THESE

Présentée à
Université de Bretagne Sud
En vue de l'obtention du

DOCTORAT

Spécialité
Sciences de l'Ingénieur en Electronique et Informatique Industrielle (**UBS**)
Ingénierie des Systèmes Informatiques (**ENIS**)

Préparée en cotutelle au
Laboratoire d'Electronique des Systèmes Temps Réel (**LESTER**)
De Université de Bretagne Sud (France)
&
L'unité de recherche Computer and Electronics Systems (**CES**)
De l'Ecole Nationale d'Ingénieurs de Sfax (Tunisie)

Fatma ABBES BEN AMOR

ENCAPSULATION DES COMPOSANTS VIRTUELS DANS UN SYSTEME SUR PUCE

Soutenue le 11 mai 2007

Composition du jury

M. .Dominique HOUZET	Rapporteur
M. Habib YOUSSEF	Rapporteur
M. Jean Luc PHILIPPE	Examinateur
M. Eric MARTIN	Examinateur/Président
M. Emmanuel CASSEAU	Directeur
M. Mohamed ABID	Directeur
M. Philippe COUSSY	Invité

DEDIACE

*A mon père « **ALI** », A ma mère « **SERRA** »*

Auxquels

Je dois ce que je suis

Que dieu vous protège Et vous prête une bonne santé
Et une longue vie

*A mon cher mari « **NADER** »*

Pour la tendre affection

Pour les sacrifices endurés et les encouragements

*A ma petite **« BALKIS »***

*A ma sœur « **DORRA**» et mon frère « **MOHAMED** »*

Pour les encouragements continus

*A mes beaux parents « **MOHAMED** » & « **NOURA** »*
A ma belle soeur
A mes beaux frères

*A l'âme de mon oncle **« ABDEFETTEH »**, A l'âme de mon oncle **« ALI»**,*

A toute ma famille

A tous mes amis
A tous ceux que j'aime Et qui m'aiment

REMERCIEMENTS

C’est avec un immense plaisir que je réserve ces lignes en signe de gratitude et de reconnaissance à tous ceux qui ont contribué de près ou de loin à l’élaboration de ce travail.

Je tiens à remercier en premier lieu Monsieur Mohamed ABID, Professeur à l’ENIS, Directeur du groupe de recherche « Systèmes sur Puces » du laboratoire CES pour m’avoir accepté dans son groupe de recherche. Je tiens à remercier également Mr Eric MARTIN Professeur des Universités et Président de l’Université de Bretagne Sud (ancien directeur du laboratoire LESTER) pour m’avoir accepté dans le laboratoire LESTER. Je tiens à exprimer ma vive gratitude à mon directeur de thèse, Monsieur Emmanuel CASSEAU, Professeur des Universités à l’Université de Rennes à Lannion pour son accompagnement et ses conseils.

Mes remerciements s’adressent aussi à Messieurs Dominique Houzet, Professeur des Universités au laboratoire LIS à Grenoble et Monsieur Habib Youssef, Professeur à l’Institut Supérieur d'Informatique et des Technologies de Communication à Hammem Sousse pour avoir accepté de rapporter mon travail. Je tiens à remercier également Messieurs Jean Luc Philippe Professeur des universités à l’Université de Bretagne Sud– LESTER pour accepter d’être membre au jury et Monsieur Philippe Coussy, Maître de Conférences à l'Université de Bretagne Sud – LESTER, pour accepter d’être invité au jury.

Je tiens à remercier mes enseignants à l’ENIS, ainsi que tout son personnel administratif et technique.

Je tiens également à remercier tous mes amis de l’ENIS. Je remercie également tous ceux qui ont contribué aux travaux présentés dans ce mémoire.

Je remercie également le personnel du LESTER pour sa disponibilité.

RESUME

Avec l'augmentation du taux d'intégration qui dépasse actuellement plusieurs centaines de millions de transistors, de nouveaux systèmes sont apparus appelés « système sur puce » ou SoCs. Ces systèmes sont assujettis à de fortes contraintes de conception : délai de mise sur le marché court, support d'applications complexes, coût réduit, etc. Une des solutions à ce défi est l'utilisation de composants prêts appelés IPs ou composants virtuels qui permettent de réduire le temps de développement et de minimiser les erreurs de conception. Ces IPs pouvant être par essence de provenance différente, leur structure de communication n'est pas forcément adaptée au reste du système. Un des problèmes de conception des SoCs est alors d'intégrer facilement et rapidement ces IPs dans le système.

Ce travail de thèse porte sur la conception automatique de l'interconnexion entre les composants IPs d'un SoC. Ces travaux ont été menés conjointement dans l'équipe « système sur puce » du laboratoire CES de l'ENIS et le groupe de recherche IP Design du laboratoire L.E.S.T.E.R de l'UBS.

Cette thèse propose une approche d'intégration/d'encapsulation d'IPs applicable pour un contexte de simulation et un contexte de synthèse. Elle est basée sur l'instanciation d'une interface de communication générique à l'aide d'une configuration à travers des graphes. Ces graphes modélisent les transferts de données entre le système et l'IP. Cette interface de communication cible les applications orientées flot de données pour un contexte de réutilisation SoC/MPSoC. La réalisation de l'interface de communication consiste alors pour la partie matérielle à instancier le ou les modules d'interface nécessaires, et pour la partie logicielle à générer le pilote de l'interface.

Afin d'automatiser cette approche, un outil a été développé. Il permet dans un premier temps de vérifier la compatibilité entre l'IP et le reste du système. Dans un second temps, il permet de générer le code SystemC de l'interface pour le contexte de la simulation, et le code VHDL synthétisable pour la synthèse (avec les pilotes et les fichiers de test nécessaires).

Pour l'expertise de cette approche, l'application « synthèse d'image 3D » a été choisie. L'IP considérée est « le produit matriciel ». L'application de notre méthode démontre que l'interface adoptée est indépendante du contexte d'utilisation et que l'approche peut être utilisée pour l'intégration automatique d'IPs.

***Mots clés** : Système sur puce, communication, interface, outil d'aide à la conception, intégration d'IPs, synthèse de haut niveau.*

ABSTRACT

In order to manage the system-on-a-chip (SoC) increasing complexity, a promising way consists of the reuse concept of preconceived hardware or software blocks. An important aspect of a core's marketability is its ability to be easily integrated into a SoC since IP must be usable in many different application contexts. Integrating Intellectual Property (IP) components into SoC design requires the use of a hardware/software interface.

This PhD thesis deals with interconnection design between IP cores (Intellectual Property) in a System on Chip. This work was undertaken jointly in the team "IP Design" of laboratory L.E.S.T.E.R of the UBS and the C.E.S Group of ENIS.

To increase reuse efficiently, quality and productivity of SoC design, we propose a design approach for packaging the cycle accurate and bit accurate (CABA) interface of hardware IPs in SoC/MPSoC design context aiming data flow emerged systems. This approach gives an interface modelling considering communication adaptation concepts/context. Graph formalism has been established to specify data traffic considering the cycle accurate behaviour at the IP interface and system requirements. Moreover, the approach is built around two main steps: checking compatibility and interface architecture generation. To realize communication adaptation, both the software part ("driver") and the hardware part (the interface) are generated.

A communication interface architecture generator has been implemented as a CAD tool called GIC. This tool is able to choose and to configure generic interface parameters according to applications constraints and system needs through graphs models. Moreover, it generates the synthesisable VHDL code for synthesis and SystemC code for simulation of the specified interface generated.

This work has been validated on multimedia application, a "pipeline 3D" application. The application of our method shows that the adopted interface is independent of use context and that the approach can be used for automating IPs integration.

Keys words: System on Chip, communication, interface, computer aided design, IP integration and high level synthesis.

Table des matières

Liste des Figures

Liste des Tableaux

CHAPITRE 1. INTRODUCTION GENERALE

Motivation & Problématique

La technologie permettant l'intégration sur silicium des circuits numériques n'a cessé d'évoluer suivant la loi de Moore [Tur03]. Avec ce développement technologique des composants submicroniques, nous assistons à un taux d'intégration supérieur à plusieurs centaines de millions de transistors ce qui a confirmé et renforcé la nouvelle aire d'implémentation des systèmes sur une seule puce SoC « *System on Chip* » ou « systèmes sur puce ». Cette nouvelle génération de systèmes numériques a ouvert la voie pour l'apparition et la large diffusion de produits performants non encombrants, efficaces, flexibles et capables de s'adapter aux changements des normes des applications. Ces systèmes sont de plus en plus utilisés dans plusieurs domaines notamment dans le domaine de multimédia et le domaine de traitement de l'information (exemple : Traitement De Signal et d'Images : TDSI) caractérisés par des applications variées et complexes devant être exécutées sur des systèmes performants, ergonomiques et de faible coût.

Par ailleurs, les applications supportées par ces systèmes sont de plus en plus complexes. Toutefois, leur temps de mise sur le marché est de plus en plus court. Les concepteurs doivent donc concevoir ces systèmes complexes en un temps minimal tout en respectant des contraintes temporelles et spatiales strictes. Afin de réduire ce temps, il est nécessaire de disposer d'outils de CAO (Conception Assistée par Ordinateur) capables d'accélérer les différentes phases de conception et profiter ainsi du potentiel d'intégration permis par l'évolution de la technologie. En effet, à l'horizon 2010, les technologies de fabrication de circuits intégrés sur silicium permettront d'intégrer des milliards de transistors sur une seule puce. La figure 1 donne les caractéristiques technologiques de la prochaine génération de systèmes. De tels systèmes sont de plus en plus construits à partir de composants développés en interne ou achetés de l'externe, dits « IPs » (***Intellectual Properties***) ou composant virtuel (***VC : Virtual Component***).

Ces IPs sont des fonctions qui permettent de réaliser un traitement donné (ex. FFT, DCT, décodeur MPEG, transformée en ondelette, estimation de mouvement etc.) et leur complexité peut atteindre celle d'un processeur. Ces composants sont conçus puis vérifiés. Ils sont souvent flexibles vis à vis des technologies ciblées, lorsqu'ils sont décrits en langage

HDL (*High Description Language* : langage de haut niveau) au niveau RTL (*Register Transfer Level* : niveau transfert de registre).

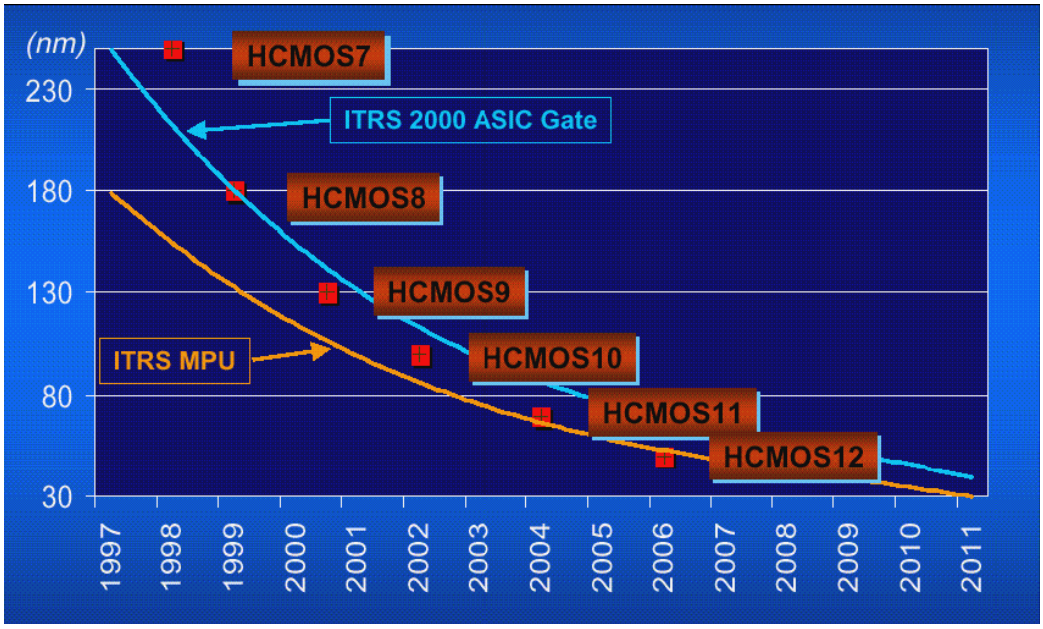

Figure 1. Prévisions d'évolutions des technologies silicium[1].

Dans la conception de cette nouvelle génération de SoC, la partie fonctionnalité n'est plus la partie la plus difficile à concevoir, mais c'est plutôt la partie communication entre les différents composants. Afin de permettre aux concepteurs des systèmes sur puce d'interconnecter facilement des composants de provenances diverses, ces composants doivent pouvoir communiquer et s'interfacer facilement entre eux. Pour une réutilisation contrainte dans un système complet, il est donc souvent nécessaire d'adapter un composant à la structure de communication existante du système grâce à un module matériel de translation. La conception de ce module de translation est à la charge du concepteur intégrateur. Le problème d'interfaçage d'IP dans ces conditions est traité selon plusieurs facettes en considérant plusieurs niveaux d'abstraction. Il existe :

- Différentes méthodologies de conception
- Différentes technologies d'implémentation (ASIC (Application Specific Integrated Circuit), FPGA (Field Programmable Gate Array) ….)
- Différents protocoles de communication
- Différentes fréquences de fonctionnement

[1] Source : The transistors international technology roadmap for semiconductors

Les solutions de ce problème apportées par la recherche académique ou industrielle peuvent être regroupées en trois familles :

- la synthèse à partir d'une spécification système fonctionnelle,
- la conception s'appuyant sur l'utilisation de plateforme.
- la conception basée sur la réutilisation de composants.

Une des solutions consiste à utiliser des protocoles de bus existants comme par exemple le bus AMBA ou le coreconnect (CoreConnect). Cette tendance a évolué pour favoriser la démarche de conception basée sur une plateforme qui impose l'emploi de bus particuliers [Reg04]. Malgré ses avantages, l'utilisation de bus standard peut ne pas satisfaire les exigences du système en terme de débits de données puisque le gap entre la fréquence du bus et la fréquence du système ne cesse d'augmenter. Pour garantir plus de performances notamment dans les applications multimédia complexes (par exemple la synthèse 3D, la compression MPEG4), il est nécessaire d'améliorer la vitesse de transfert des données en adoptant de nouvelles structures de communication. Leur protocole de communication doit être suffisamment générique pour pouvoir être interfacé facilement avec n'importe quel autre protocole ce qui permet de réduire la complexité de la tâche d'adaptation du composant. Cependant, malgré l'effort de standardisation en terme de protocole de communication VCI [VCI] (*Virtual Component Protocol*) ou OCP [OCP] (*On Chip Protocol*)), diverses applications nécessitent l'intégration de fonctionnalités particulières non supportées par ces standards. Par exemple, l'utilisation de tels protocoles n'exempte pas le « wrapper » (adaptateur matériel entre composants utilisant des protocoles différents) de la tâche de temporisation des données. Par ailleurs, l'apparition de nouvelles structures d'interconnexion (comme les Networks on Chip : NoC) pose de nouvelles contraintes dans la conception des interfaces [Ben02].

Pour toutes ces solutions, il est nécessaire d'obtenir des structures de communication adaptées aux exigences des applications cibles capables d'interconnecter des IPs de provenances diverses. Ainsi, la «*génération automatique d'interfaces* » doit être considérée pour favoriser l'interfaçage automatique des IPs dans un flot de conception SoC [SPIRIT05]. Cette pratique permet de gérer les contraintes de conception de plus en plus pressantes et de relever les nouveaux défis apparus avec les nouvelles structures de communication mises en place telle que la communication multiprocesseurs.

De plus, à présent, si un système est simulé correctement au niveau fonctionnel, un passage automatique au niveau RTL ne peut pas être garanti spécialement au niveau de la synthèse de communication [Cyr04] où le concepteur est souvent appelé à faire des

ajustements manuels souvent délicats et fastidieux. Il est donc important de disposer d'outils pour automatiser la génération des structures de communication sous contraintes d'application et de prototypage pour la simulation et la synthèse. Actuellement, la technologie du logiciel dans la conception des SoCs fournit des outils satisfaisants permettant la synthèse logique à partir d'une spécification RTL. Ces outils restent cependant incomplets et ne permettent pas une génération automatique d'architectures à partir d'une description de haut niveau. C'est pourquoi les tendances actuelles mènent vers l'établissement de méthodologies pour rehausser le niveau de la conception en utilisant de nouveaux langages et de nouvelles approches.

Objectifs

Notre objectif est de développer une approche interactive pour l'intégration ou l'encapsulation des IPs accélérateurs matériels dans un système mixte logiciel/matériel en ciblant les applications orientées flot de données. Cette approche suppose que le système est déjà partitionné en tâches logicielles et matérielles. Elle suppose également l'existence d'une architecture cible constituée par une plateforme monoprocesseur ou multiprocesseur. L'approche d'intégration proposée cible les méthodologies de conception des SoCs à base de plateforme « *platform based design* » ou à base de composants « *component based design* ». Elle permet de générer une interface de communication pour la simulation et/ou la synthèse. Les primitives d'intégration permettent la génération des schémas de communication pour l'IP à intégrer.

L'objectif à long terme est d'enrichir la bibliothèque du générateur afin de généraliser son utilisation à d'autres types d'IPs.

Contributions

Afin de faciliter la réutilisation des IPs, nous avons défini dans cette thèse une méthode pour l'automatisation d'intégration de ces composants dans un système sur puce. Cette méthodologie utilise un modèle d'interface générique pour assurer l'intégration automatique d'un bloc IP. La prise en compte de l'ordonnancement des données pour la synchronisation des entités communicantes est assurée par une formalisation sous formes de graphes. Le modèle de l'architecture de l'interface ainsi que la formalisation de l'ordonnancement du transfert des données sont définis pour mettre en place un outil informatique permettant de généraliser l'application de l'approche d'intégration pour des

contextes différents. Les fonctionnalités assurées par le modèle d'interface sont proposées pour un contexte de simulation et de synthèse.

Par ailleurs, les outils de synthèse architecturale ou de synthèse de haut niveau (SHN) accomplissent un rôle primordial pour réduire le temps de conception et les erreurs dues aux manipulations manuelles des conceptions. En revanche, ils ne permettent pas de définir clairement les communications externes du circuit. Les performances d'un circuit à l'intérieur d'un système intégré sont difficilement évaluables autrement que par la simulation du système. Pour garantir plus d'efficacité de leur utilisation dans un flot de conception de SoC à base d'IPs réutilisables, nous devons faire face aux problèmes d'intégration de leurs IPs dans des environnements de simulation et de synthèse.

Pour promouvoir l'utilisation de ces outils et pour remédier essentiellement aux problèmes liés à l'intégration automatique d'IPs dans un contexte contraint par l'architecture cible, un outil « générateur d'interface de communication » implémentant l'approche d'intégration proposée a été mis en place. Il permet dans sa première version l'interfaçage des IPs accélérateurs générées par un outil SHN développé au sein du laboratoire LESTER appelé GAUT.

Le rôle de l'interface est l'intégration/l'encapsulation de l'IP (adaptation à l'interface de l'interconnecte du système) et l'acheminement des données ordonnées vers les ports d'entrée ou de sortie correspondants. Pour notre approche d'encapsulation, nous avons traité le problème de la consommation de données aléatoires du côté de l'IP avec des données de tailles différentes pour une architecture d'IP synchronisée par les données.

Les points clés dans notre approche d'intégration d'IP sont :

1. Permettre la réutilisation systématique de composants matériels existants. La méthode doit favoriser cette réutilisation pour des environnements spécifiques.
2. Considérer une structure d'architecture d'interface générique pour des applications opérant sur des flux de données importants (flux de données ordinairement dans le traitement de signal ou les images dans le multimédia).
3. La méthodologie d'encapsulation ou d'intégration d'un composant matériel cible un contexte de simulation et de synthèse. Dans ce cadre, la plateforme SoCLIB a été adoptée pour la simulation et la cible FPGA Altera pour la synthèse.

Organisation du document

Ce rapport est organisé en cinq chapitres structurés comme le montre la figure 2.

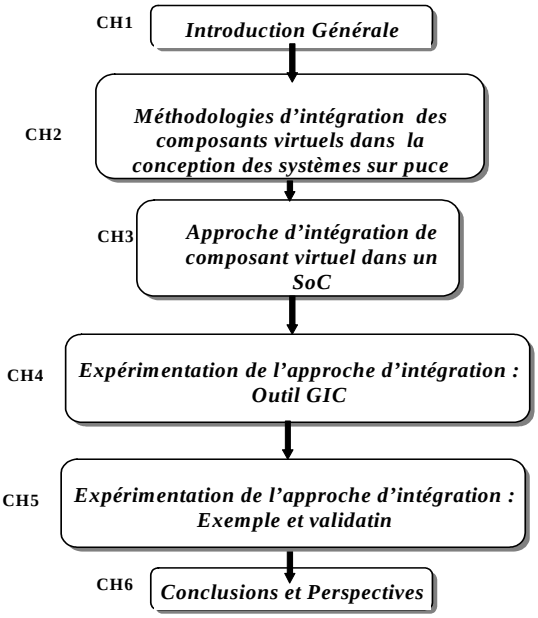

Figure 2. Structure du mémoire

Le chapitre 2 présente le flot de conception des systèmes sur puce mono/ multiprocesseurs basé sur la réutilisation des composants virtuels. Cette partie présente l'intégration d'IP dans un SoC, nous y présentons notamment quelques approches et quelques outils automatisant la génération d'interface de communication dont l'approche proposée dans le cadre de cette thèse.

Le chapitre 3 présente l'approche d'intégration proposée pour la mise en place de l'outil « générateur d'interfaces ». Nous considérons dans cette partie les détails et les hypothèses de l'application de cette approche ainsi que la formalisation des contraintes d'intégration. Cette spécification favorise l'automatisation de l'approche qui repose sur une architecture générique d'interface de communication. Cette architecture générique est constituée à l'aide de modules décrits sous forme de machines d'états finis.

Le chapitre 4 présente l'expérimentation de l'approche à travers la conception et la réalisation d'un outil de CAO automatisant les étapes de l'approche d'intégration.

Le chapitre 5 met en évidence les fonctionnalités de l'outil. Ces fonctionnalités sont illustrées pour la génération d'interfaces de communication pour des IPs accélérateurs dans un contexte de simulation, dans une première partie de ce chapitre, et de synthèse, dans la seconde partie, permettant ainsi de valider l'approche.

Le chapitre 6 présente les conclusions et les perspectives des travaux menés dans le cadre de cette thèse. Les aspects qui sont devenus d'actualité et qui n'ont pas été évoqués ou traités dans cette thèse seront évoqués dans ce chapitre.

CHAPITRE 2. METHODOLOGIES D'INTEGRATION DES COMPOSANTS VIRTUELS DANS UN FLOT DE CONCEPTION DE SYSTEME SUR PUCE

1. Introduction

Actuellement, un véritable décalage persiste entre la capacité d'intégration des transistors et la capacité de conception des systèmes sur puces. En effet, la progression du nombre de transistors par puce suit toujours la loi de Moore. Le rythme observé correspond à une augmentation de 58% par an [Bou05] (Cf. figure 3). En 1998, un microprocesseur était composé d'environ 10 millions de transistors. En 2008, un circuit de référence équivalent pourra être composé de près de milliard de transistors. Parallèlement l'augmentation de la productivité des équipes de conception est plutôt de l'ordre de 21%.

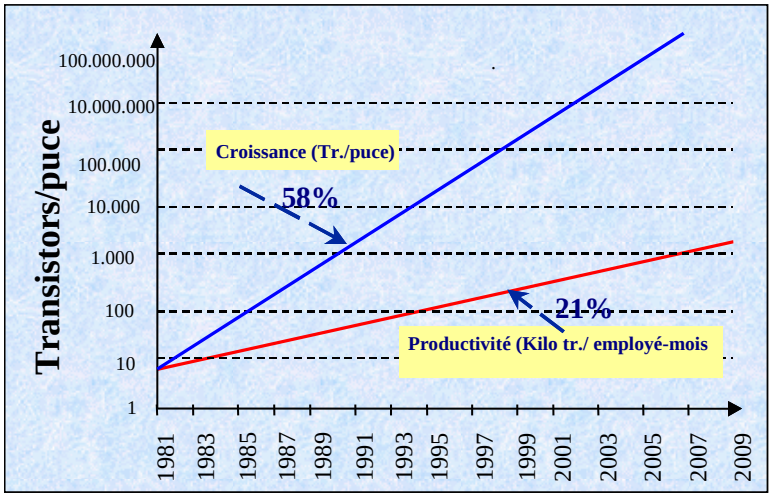

Figure 3. Prévisions des besoins de productivité[2]

Afin de pouvoir gérer la complexité croissante des SoCs, une voie prometteuse consiste à réutiliser des blocs matériels ou logiciels préconçus et vérifiés pour certaines fonctions des systèmes. La conception des systèmes intégrés fait aujourd'hui largement appel à la réutilisation de composants préexistants. C'est ainsi que nous parvenons à produire des dispositifs de grande complexité en maîtrisant l'effort de conception et de validation. Les éléments réutilisables sont matériels (CPU, mémoire,…) ou logiciels (OS,

[2] Source : Semiconductor Industry Association

applicatif,…). Cependant, puisque ces composants sont hétérogènes, leur intégration nécessite des sous-systèmes adaptateurs. Dans ce cadre, la réutilisation ou l'intégration des blocs existants constitue une phase clé de la conception des SoCs permettant de faciliter et d'accélérer le cycle de conception [Kea03].

Par ailleurs, pour intégrer des composants dans un système global, une adaptation de leur interface au bus/réseau de communication est indispensable. Toutefois, la communication entre composants est devenue un véritable goulot d'étranglement. De plus, les performances, la consommation et les coûts de développement de ces systèmes sont fortement dépendants des choix de protocoles de communication et de leur implémentation.

Dans ce cadre, le cycle de conception des systèmes hétérogènes embarqués peut être décomposé en deux phases : la phase de conception des composants et la phase d'intégration de ces composants dans le même système. Notre travail concerne le deuxième processus.

Le but de ce chapitre est de présenter les approches et les travaux associés à l'intégration d'IP dans un SoC. Selon une classification des méthodologies de conception SoC, nous analysons dans la section 2 de ce chapitre des solutions proposées pour la conception des systèmes sur puce. Cette étude focalise sur l'aspect communication dans la procédure d'intégration des composants préexistants et sur les techniques d'interfaçage d'IPs. La communication peut être prise en charge par une structure de communication complexe, allant du simple bus jusqu'au réseau de communication de paquets. L'interfaçage d'IPs est traité soit en vue de simulation et/ou de synthèse selon l'étape de la conception concernée et l'efficacité des outils utilisés. La section 3 analyse la génération automatique d'interface de communication matérielle/logicielle à travers la présentation de quelques méthodologies. La section 4 traite la résolution des problèmes d'intégration à travers la synthèse d'interface de communication. Enfin, la section 5 analyse des exemples de travaux qui traitent la génération automatique d'interface pour l'encapsulation et/ou pour l'intégration d'IPs dans un SoC.

2. Conception des systèmes sur puce

La complexité croissante des applications entraîne une augmentation considérable de la durée de développement avec la méthode de conception des circuits. Par conséquent de nouvelles méthodologies de conception sont apparues. Dès les années 90, les chercheurs se sont concentrés sur la mise en œuvre d'une nouvelle méthodologie de conception appelée : « co-design » [Ism94] [Ben95] [Gup95] [Ben97] [Abi98]. Cette méthodologie se base sur la

considération conjointe de la conception du logicielle et du matérielle et couvre les différents cycles de développement depuis la spécification jusqu'à la réalisation physique du système.

Toutefois, la complexité croissante des applications entraîne une augmentation considérable de la durée de développement avec les méthodes de conception conventionnelles [Her02]. En effet, les systèmes sur puce sont de plus en plus présents dans la vie courante. Aussi, une nouvelle gamme de produits embarqués (nomades) devient de plus en plus populaire : les téléphones portables, l'agenda électronique, les voitures, les satellites etc. Souvent, ces systèmes enfouis intègrent des composants logiciels et matériels sur un même support et leur mise en place nécessitent de nouvelles méthodologies de conception. Ces méthodologies se basent sur le concept de la réutilisation d'IPs.

Dans cette section, nous rappelons les étapes d'un flot typique de la conception des systèmes mixtes pour introduire la conception des SoCs avec la prise en considération des composants préconçus (IPs). Nous nous intéressons spécifiquement à l'aspect « intégration d'IP ».

2.1. Conception conjointe logicielle matérielle

Le « co-design » constitue le point de rencontre entre les communautés de la conception de circuits intégrés, du génie logiciel et de la spécification des systèmes. Le développement des systèmes enfouis composés d'une partie logicielle et d'une partie matérielle n'est pas un nouveau problème. La conception et la réalisation de tels systèmes nécessitent une compétence technique dans trois domaines : l'électronique analogique, l'électronique numérique et l'informatique. La nature spécifique du traitement à effectuer et le couplage du système avec son environnement nécessitent aussi des compétences complémentaires : en traitement de l'information (signal, image, parole etc.), en électronique de puissance, en réseaux et télécommunications, etc. Vu l'importante croissance de complexité que connaissent les SoCs, particulièrement dans des domaines tels que le traitement de signal intensif, il devient de plus en plus nécessaire d'unifier (standardiser) le flot de conception de ce type de systèmes dit flot de conception logicielle/matérielle.

2.2. Flot typique de la conception conjointe

La conception conjointe matérielle/logicielle (« co-design ») est une approche qui intègre, dans un même environnement, la conception du matériel et du logiciel. La conception commence à partir d'une spécification au niveau système (le niveau d'abstraction le plus

élevé) et conduit à un premier prototype d'architecture, qui respecte les contraintes de la conception. Ce type de conception admet différentes méthodologies. Les méthodologies regroupent les aspects techniques et les aspects d'organisation de la conception. Elles coordonnent l'utilisation conjointe de plusieurs outils de conception et la coopération de plusieurs aspects liés à tous les niveaux de développement d'un système électronique. Les équipes du logiciel et celles du matériel peuvent travailler en parallèle, dans un environnement de coopération et de collaboration qui réduit le cycle de développement de la conception.

La figure 4 illustre les étapes du cycle de conception d'un système.

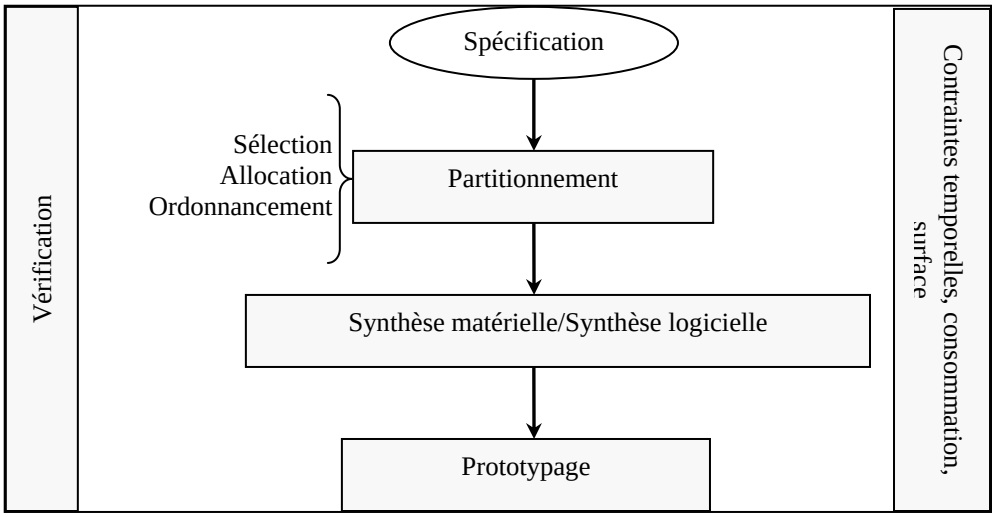

Figure 4. Flot de conception « co-design »

Les étapes principales de la conception logicielle/matérielle sont :

- la spécification et la modélisation
- le partitionnement (découpage) logiciel/matériel
- la co-synthèse (synthèse de communication)
- le prototypage

De plus, les tâches de vérification et de validation sont appliquées au cours de la conception pour vérifier le respect des contraintes imposées (temporelles, consommation, surface…) (Cf. figure 4).

Ces étapes définissent différents niveaux d'abstraction. A chaque niveau est associée une description abstraite servant de spécification pour la conception au niveau inférieur qui conduit à une description plus détaillée et plus concrète. L'abstraction est un concept basé sur le fait d'ignorer ou de supprimer, dans un modèle, certains détails du système original afin de simplifier la taille de ce modèle, d'en déduire des conclusions d'ordre général [Abi98].

2.2.1. Spécification système

Le processus de « co-design » commence par la modélisation du système à concevoir. Il s'agit de pouvoir décrire l'application comme étant un ensemble de fonctions et de contraintes, indépendamment de toute considération matérielle ou logicielle. La modélisation s'effectue par la décomposition de l'application en modules. La spécification de chaque module peut utiliser un langage différent.

Deux niveaux de spécification peuvent cependant être distingués :

- La spécification fonctionnelle : elle décrit le comportement attendu de l'application.
- Les spécifications non fonctionnelles : ce sont les contraintes externes que le système devra supporter.

Ces spécifications sont généralement écrites en langages de haut niveau ou parfois en langage de programmation logiciel pour permettre une spécification système de l'application.

2.2.2. Partitionnement logiciel / matériel

Suite à l'étape de modélisation et spécification fonctionnelle du système, le concepteur doit déterminer les parties qui seront réalisées en matériel (de types ASIC, FPGA...) et les parties qui seront réalisées en logiciel (Processeurs, DSPs, etc). Il s'agit de trouver le "meilleur" compromis entre logiciel et matériel pour chaque fonction du système.

A l'issue de cette étape, un sous-ensemble matériel, un sous-ensemble logiciel et leur communication sont spécifiés à un niveau de détail suffisant permettant leurs synthèses respectives.

2.2.3. Synthèse matérielle et synthèse logicielle (synthèse mixte ou co-synthèse)

Une fois l'affectation des tâches effectuée et validée, il faut synthétiser les tâches matérielles. Cette synthèse se fait généralement par un outil dédié à la conception matérielle. Les tâches logicielles doivent être compilées pour le processeur cible et exécutées par lui. Si plusieurs tâches sont affectées à un même processeur, soit ces tâches peuvent être fusionnées, soit elles peuvent être ordonnées statiquement, soit il faut utiliser un système d'exploitation multitâche. Ensuite, il faut intégrer ces codes dans des ROMs « Read-Only Memory » afin que le processeur physique du système embarqué puisse y accéder. Une fois les parties matérielles et logicielles générées, il faut garantir que les transferts de données et la synchronisation entre elles s'effectuent correctement. Ce problème est appelé synthèse d'interface logiciel/matériel ou synchronisation du système. La synthèse des communications

est une étape très importante dans l'interfaçage logiciel/matériel. Il s'agit de définir pour chaque transfert les protocoles de communication et le mode de communication (point à point, via mémoire partagée, etc.).

2.2.4. Prototypage

Le prototypage est divisé en deux étapes : le prototype virtuel, qui emploie les techniques de co-simulation et de co-synthèse et le prototypage physique, qui produit un premier exemplaire du système. La création d'une réalisation pour chaque composant est réalisée par l'intermédiaire des techniques de conception logicielle/matérielle classiques. Le prototypage consiste à effectuer la synthèse du matériel et la génération du code logiciel à partir des descriptions générées par les étapes précédentes. La co-simulation intervient après la phase de partitionnement. L'objectif consiste à vérifier que les spécifications matérielles et logicielles sont valides. Cela implique les tests de chaque module et de son interface de communication, l'étude de l'évolution du système en présence des contraintes (performances, coûts d'implantation etc.).

L'évolution rapide de la complexité des applications embarquées, destinées à être implantées sur des cibles de type SoC, fait face à de fortes limitations des méthodes et des outils de conception classiques. Avec l'émergence vers des SoCs, un flot de « co-design » dédié pour la conception d'un SoC repose sur la réutilisation de composants virtuels (IPs).

2.3. Flot de conception des SoCs

L'architecture des SoCs est composée de différents composants logiciels/matériels hétérogènes. Une voie prometteuse consiste en le concept de réutilisation des blocs matériels ou logiciels préconçus et vérifiés pour certaines fonctions des systèmes. L'utilisation de blocs préconçus pour les SoCs est devenue une technique de plus en plus indispensable pour concevoir un système complexe dont le temps de mise sur le marché est très court (de l'ordre de quelques mois).

La figure 5 décrit une approche typique de la conception système à l'aide de composants virtuels. Il s'agit d'un flot complet de conception conjointe logiciel/matériel (co-design) qui vise la mise en œuvre de systèmes SoCs sur une architecture hétérogène (matérielle/logicielle). Cette méthodologie est développée en plusieurs étapes qui sont :

- La définition des besoins et analyse des contraintes,
- La spécification du système et sa modélisation,
- Le partitionnement Logiciel/Matériel (L/M),

- La synthèse des composants matériels et logiciels,
- La synthèse de l'interface L/M,
- La simulation (ou la co-simulation) jusqu'à la génération du modèle d'architecture (réalisation du système).

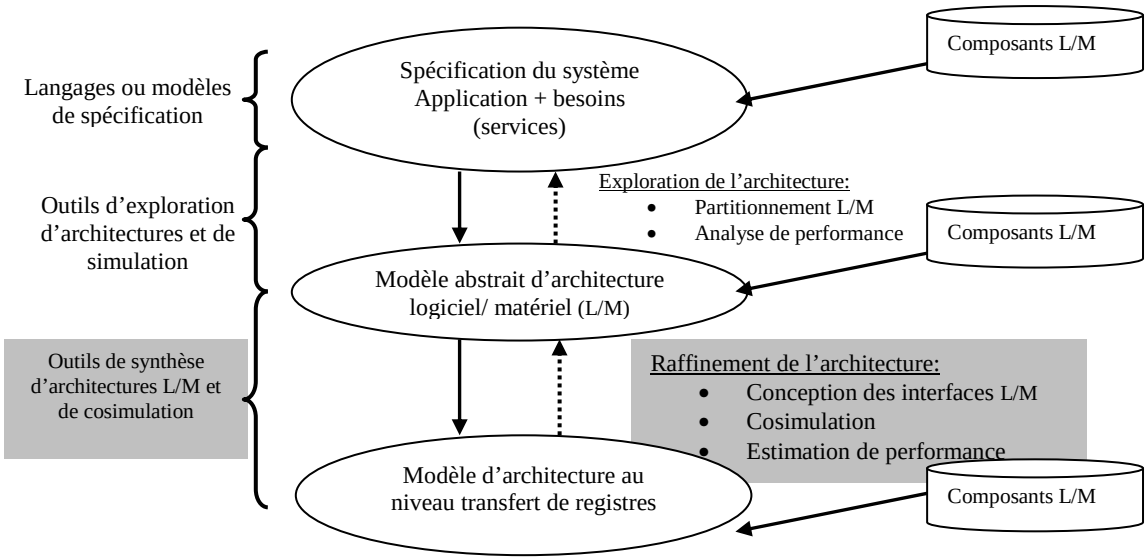

Figure 5. Flot typique de la conception des SoCs avec la prise en compte des IPs

Ce flot de conception est basé sur la réutilisation des IPs. Il permet donc d'exploiter l'adéquation entre l'application et le système grâce au prototypage rapide de ces composants virtuels au niveau d'abstraction le plus élevé. Ces composants sont mis en œuvre dans le système à l'aide d'une stratégie de réutilisation dans le but de manier la complexité croissante des SoCs. Les concepteurs des SoCs sont obligés donc d'intégrer les composants logiciels et les composants matériels tout en respectant les contraintes de performance décrites au niveau de l'architecture abstraite (modèle abstrait d'architectures dans la figure 5). Cependant, l'incorporation d'IP dans un SoC ou dans une architecture à base de plateforme sans considérer les contraintes de synchronisation aux entrées sorties peut faire échouer la conception du système. Ce qui revient à dire que la génération de toutes les interfaces logicielles et matérielles doit se faire au niveau de précision le plus bas : niveau microarchitectures. Cela n'exclut pas la possibilité de la spécification des composants et des interconnections à différents niveaux d'abstraction.

Afin de favoriser la spécification, le concept de la séparation du comportement de la communication connu sous le terme « Interface Based Design » [Raw97] est utilisé permettant ainsi l'abstraction de l'interconnexion. La séparation entre le calcul et le traitement et la communication permet de favoriser l'intégration de processeurs et de protocoles de

communication hétérogènes en utilisant des interconnections abstraites. Le comportement et la communication doivent être séparés dans la spécification système de façon que le système de communication puisse être décrit à un niveau haut et raffiné indépendamment du comportement du système.

La procédure de raffinement à partir d'un modèle abstrait est adoptée dans ce flot pour concevoir des systèmes complexes. Une classification orientée vers les concepts de la communication est présentée dans [Nic02]. Au niveau système, quatre niveaux d'abstraction de la communication peuvent être distingués. Pour spécifier des systèmes mixtes, des détails sont plus ou moins utilisés selon le niveau d'abstraction.

- *Au niveau service*, le système est modélisé sous la forme d'un ensemble de modules qui fournissent/requièrent des services. La primitive de communication typique est une requête de service, comme par exemple « imprimer (fichier) ». La notion de temps est complètement abstraite.
- *Au niveau transaction*, les différents modules du système communiquent via un réseau de canaux de communication dits actifs. Ces canaux permettent la synchronisation et peuvent inclure des comportements complexes. Mais les détails de la communication sont englobés par des primitives de communication de haut niveau (par exemple « send/receive ») et aucune hypothèse sur la réalisation des protocoles de communication n'est faite. Le langage SDL (System Description Language) [Sar87] peut être placé à ce niveau.
- *Au niveau macro-architecture ou niveau message* la communication se fait par des fils abstraits, avec des protocoles de communication pour les entrées/sorties. La modélisation à ce niveau implique par conséquent le choix d'un protocole de communication et la topologie des interconnexions. SystemC [Sys02] est un langage caractérisant le mieux un système à ce niveau d'abstraction.
- *Au niveau RTL ou micro-architecture*, le système est décrit sous forme d'un ensemble de registres, de circuits combinatoires, de circuits de contrôle, de fils et de bus physiques. La granularité temporelle est le cycle d'horloge et les primitives de communication sont du type « set/reset » sur des ports.

Ces niveaux favorisent l'aspect raffinement pour la mise en œuvre du style de communication dans un SoC. L'intégration d'un IP dans un SoC peut être traitée selon ces différents niveaux d'abstraction.

Dans ce contexte, la section suivante présente les techniques utilisées pour les différentes solutions adoptées ainsi que les méthodologies suivies pour l'intégration d'IP dans un SoC.

2.4. Intégration d'IP dans un SoC

Avec l'apparition des structures de communication SoC de plus en plus complexes tels que les réseaux sur puce (NoC) (Cf. figure 6), les difficultés dues aux limitations des bus (bande passante, nombre de composants esclaves supportés) peuvent être surmontées. En revanche, l'interfaçage de l'IP reste une issue importante [Deu02]. Ces interfaces correspondent à des adaptateurs matériels flexibles connectant l'IP au réseau de communication à l'aide de pilotes d'accès adaptant le logiciel de l'application aux processeurs cibles [She04]. La communication entre IPs est devenue un véritable goulot d'étranglement vu que ces composants :

– sont fournis par des sources différentes
– ont des domaines d'application spécifiques.

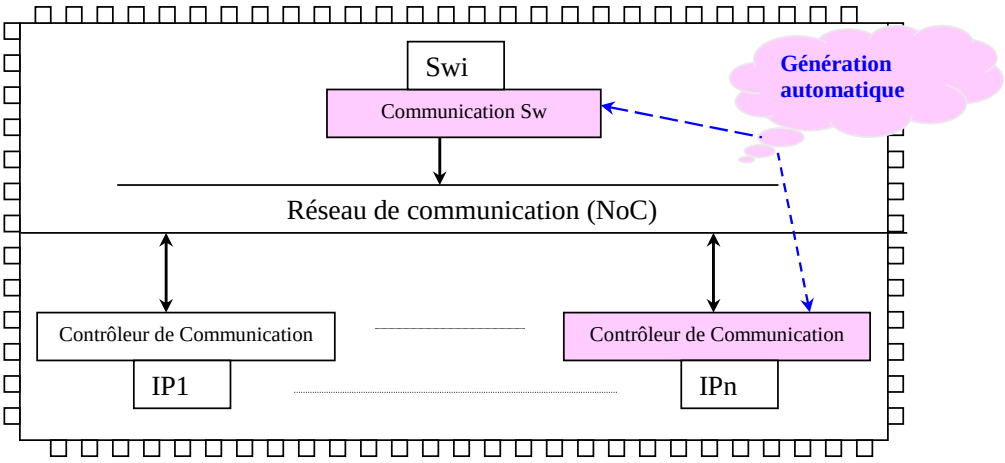

Figure 6. Communication entre IPs dans un SoC

Deux techniques sont utilisées pour l'adaptation de l'interface d'IP : la personnalisation et l'encapsulation.

La personnalisation «customisation » d'un IP est un processus de conception qui change les caractéristiques de l'IP sans modifier sa fonctionnalité originale. Elle est utilisée pour pouvoir intégrer un IP dans un SoC. Le « wrapping » est un cas particulier de la « customisation ». C'est une fonctionnalité supplémentaire qui est ajoutée à la fonctionnalité principale pour l'adaptation de l'IP aux contextes de son utilisation. Le « wrapper » est la « glue » logique nécessaire pour adapter le protocole de communication de l'IP au protocole de communication de l'interconnecte du système. Il est basé sur l'utilisation d'un bloc

matériel qui prend place entre l'IP et l'interconnecte indépendamment du protocole de communication. L'encapsulation ou le «packetage » d'IP est le processus de cacher les fonctionnalités du composant derrière une interface de haut niveau sans changer la fonctionnalité ni la structure du composant. Nous pouvons par ailleurs ajouter un « wrapper » à un IP pour qu'il soit prêt à être encapsulé.

Malheureusement, en pratique, l'assemblage d'un système sur puce en utilisant des IPs [Rei00] n'est pas évident pour plusieurs raisons parmi lesquelles nous pouvons citer :

- La conception de tels systèmes exige des résultats qui ne peuvent s'obtenir qu'après la conception entière du système.
- Certes, un IP peut fonctionner correctement tout seul, son intégration dans le système peut engendrer une dégradation des performances et même des erreurs.
- L'intégration des blocs IPs fait souvent appel à des interventions manuelles du concepteur qui peuvent conduire à des erreurs dues à la complexité des modules IPs telles que par exemple la « transformée de cosinus discrète » (appliquée surtout pour des applications de traitement d'images), ou le contrôleur de mémoire, etc., et le nombre important de connexions à manipuler d'autre part.

Pour ces raisons, ces composants doivent être interconnectés à l'aide d'une stratégie permettant leur réutilisation. Dans la section suivante, nous présentons des méthodologies de réutilisation d'IP composant virtuel.

3. Méthodologies de réutilisation de composants

Au cours des dernières années, plusieurs tentatives de standardisation visant la réutilisation des composants matériels ou logiciels sont apparues. Ci après, nous présentons une classification des méthodologies et des styles de conception utilisant des IPs hétérogènes.

3.1. Approche de conception basée sur les IPs (IP based design)

Cette approche propose l'assemblage de composants matériels/logiciels réutilisables. L'étape d'intégration utilise des composants pour implémenter une architecture abstraite. Le résultat est une microarchitecture où les composants matériels sont décrits au niveau RTL. Quant aux composants logiciels, ils sont décrits en utilisant un langage de programmation approprié qui peut être compilé directement par le/les processeur(s) de l'architecture cible choisie.

Les composants sont directement connectés entre eux ou à la structure de communication. Mais d'une manière générale, le concepteur est amené à :

- Adapter les différents composants : dérivation de composant ou « IP derivation »
- Synthétiser des adaptateurs pour les composants programmables ; bien que la dérivation soit faite facilement par une re-programmation de la fonctionnalité désirée, le concepteur a besoin de développer et de synthétiser des adaptateurs « wrapper » logiciels (des pilotes de périphériques et/ou de bus) pour adapter l'application logicielle à l'infrastructure de communication et matériels pour l'interconnexion de ces composants.

3.2. Approche de conception basée sur une plateforme (plateform based design)

Cette approche utilise un modèle architectural, souvent spécifique à un domaine d'application [Keu00]. Ce modèle inclut une plateforme matérielle, qui se compose :

- d'une structure de communication donnée
- de plusieurs composants matériels (processeurs, mémoires, bloc matériel)
- d'une plateforme logicielle se trouvant sous forme d'une API [3] (Application Programming Interface) de haut niveau

Dans ces conditions, le système est conçu par une dérivation de ce modèle : les composants sont spécifiés pour satisfaire des besoins particuliers de l'application. Par ailleurs, puisque la communication est figée, les composants préconçus sont spécifiques et leur intégration est ciblée.

3.3. Approche de conception basée sur un bus (bus based approach)

Cette approche est basée sur une architecture utilisant un bus. Il existe une multitude de bus adoptés par les constructeurs, comme par exemple, le bus AMBA qui équipe les processeurs ARM [ARM99]. Différents bus peuvent être utilisés dans un même système. L'adaptation entre les différents bus se fait à l'aide de ponts « bridges » de bus (exemple l'APB et l'AHB du bus AMBA). Comme la spécification du bus est connue, des composants ayant des interfaces directement compatibles sont développés par des compagnies comme par exemple :

[3] Ensemble de commandes externes publiées par un éditeur et permettant de recourir aux fonctions d'un logiciel depuis un autre logiciel (antécédent de l'orienté objet).

- Peripheral Interconnect Bus (PiBus) d'OMI [PIBus] ;
- AMBA d'ARM Inc [ARM99]
- CoreConnect d'IBM [IBM02] ;
- OpenCore de Sonics (SON02).

La spécification d'adaptateurs est nécessaire si les composants ne peuvent s'adapter qu'avec la spécification du bus [Ber00]. Cependant, le succès d'une norme de protocole exige que tous les IPs puissent adhérer à ce bus. Bien que cette approche tient compte efficacement du choix d'une structure de bus pour un ensemble de processus, elle ne permet pas d'assurer la portabilité des IPs. En raison de la multitude des protocoles de bus propriétaires, le consortium VSIA, qui a dans un premier temps proposé de définir un bus standard, a trouvé que cette solution était trop complexe et a décidé qu'elle ne serait pas retenue [VSIA].

3.4. Approche de conception basée sur des cœurs d'IP (core/component based approach)

L'idée de cette approche est de rehausser le niveau d'abstraction au moment de la conception des interconnections entre les composants. Si cette approche ne favorise pas une aide à l'automatisation de l'exploration architecturale, elle permet néanmoins une réduction considérable du temps de conception pour le raffinement des communications logicielle/matérielle pour l'intégration de composants et pour la réutilisation d'IP. Le point clé d'un flot utilisant cette approche de conception est l'utilisation d'une architecture abstraite où la communication est séparée du traitement du côté matériel (core) et des tâches (tasks) du côté logiciel. Cette architecture abstraite est utilisée par le programmeur du côté logiciel comme un API. Ce qui assure la séparation entre la communication et le traitement pour l'approche « core based design » [Raw97].

Les composants respectent une norme d'interface indépendante du bus appelée protocole de communication. Bien qu'une norme puisse supporter des fonctionnalités variées, chaque composant peut avoir une interface qui ne contient que les fonctions qui lui sont utiles. Interconnecter ces composants à travers un bus revient à adapter leurs interfaces au bus de communication. Divers efforts ont été déployés afin de faciliter cette interconnexion. Nous citons par exemple les propositions de protocoles configurables et non spécifiques à un bus tels que VCI, OCP et « IP Interface (IPI)» [MOT]. Par ailleurs, l'approche orientée objet a été utilisée pour la modélisation haut niveau et l'adaptation des composants réutilisables (Bar99). VSIA a décidé d'approuver la tendance de transfert de sa norme d'interface de bus depuis sa propre interface VCI de VSIA à « OCP-IP » (Open Core Protocol International Partnerchip)

d'OCP. De cette façon, ce consortium tend à réduire le nombre de standards dans l'industrie [VSIA]. VSIA exige l'incorporation des caractéristiques du protocole VCI non figurant dans OCP-IP dans les futures révisions de celui-ci [VSIA]. Certes, l'adoption au début de l'an 2000 du standard VCI par les grandes « firmes » du secteur résout en partie le problème de la communication. Cependant, VCI possède des limitations. Il ne permet pas, par exemple, de dire à partir de quel moment le contenu d'une adresse peut être lu, ou réécrite, ni par qui [Pet03].

Les approches développées dans les sections 3.3 et 3.4 permettent d'interconnecter des composants en respectant un standard sans avoir recours à développer des adaptateurs complexes. Le problème est que plusieurs standards provenant de plusieurs organismes coexistent. Ceci empêche un vrai échange de bibliothèque de composants développés pour des standards différents. Cependant, ces deux approches peuvent être adaptées pour connecter les composants au réseau de communication.

3.5. Approche de conception basée sur des réseaux de communication

Pour des raisons de flexibilité, les systèmes hétérogènes peuvent être conçus autour d'un réseau de communication embarqué (NoC) [Mic02]. Afin de faciliter l'intégration de la communication, certaines approches favorisent l'utilisation d'un NoC prêt. Ces approches exigent que les IPs soient compatibles avec l'interface du NoC [She04]. Sonics propose « SOCworks » [SOC], un site « Web » interactif pour la construction et l'évaluation d'un SoC basé sur son micro réseau (µNetwork) sur lequel peuvent être connectés des composants de plusieurs producteurs.

En utilisant un protocole standard dédié pour les composants comme VCI, le concepteur peut choisir le protocole OCB (On Chip Bus) et ensuite, concevoir des « wrappers » pour les composants et présenter des méthodes de communication pour un réseau de commutation dans le cas des SoC multiprocesseurs.

Nous avons présenté les approches de conception basées sur l'intégration de composants matériels. Nous détaillons dans la suite les approches pour l'intégration des composants logiciels.

3.6. Intégration des composants logiciels

Les composants programmables sont très importants dans une plateforme architecturale réutilisable, puisqu'ils permettent de tailler cette plateforme pour des

applications différentes. Manuellement, ce processus représente un travail très fastidieux et une source d'erreurs. Pour faire face à ce problème, une technique automatique pour la synthèse du logiciel s'avère une solution concrète.

Actuellement, des efforts similaires aux efforts de standardisation visant la réutilisation des composants matériels sont nécessaires pour la réutilisation des composants logiciels. Par exemple, ce besoin guide certaines recherches intéressantes dans le domaine de la conception basée sur les langages pour faire des évolutions significatives pour améliorer les méthodologies de conception associées à ces langages. En effet, il existe un grand débat et même des confusions concernant la variété des nouveaux langages de conception apparus récemment : SystemC, SystemVerilog, Verilog-2005, e, Vera, PSL/Sugar, UML etc.

SystemC dans sa version 3.0 [SYS] inclut la modélisation du logiciel ainsi que sa simulation. Toutefois, il n'est pas un environnement de développement logiciel.

Puisque SystemC n'est pas un langage optimal pour la description du matériel ainsi que pour le niveau logique, de nouveaux langages sont apparus tel que Verilog-2005 et SystemVerilog mais ils ne sont pas non plus des langages pour la modélisation au niveau système avec une haute performance [SYSV].

Quand UML est apparu, il était encore trop tôt d'envisager qu'il supporte la modélisation des systèmes embarqués temps réel. En effet, UML « classique » ne peut pas représenter une vue architecturale avec les attributs tels que la 'hiérarchie, la structure, la connectivité ou la taille des bus. Les développeurs d'outils de CAO considèrent qu'UML a besoin encore d'extensions pour qu'il puisse supporter la modélisation des systèmes matériels. Un modèle causal est ajouté à ce langage offrant ainsi la conversion automatique des exigences en termes de vecteurs de test.

UML 2.0, la plus récente évolution du langage de modélisation UML pour la modélisation logicielle, promet de grandes capacités dans la modélisation et la génération de code spécialement pour les systèmes embarqués temps réel. Ceci pourra minimiser le temps du processus de conception d'un tiers [Yve05]. Dans ce sens, des travaux sont en cours pour définir une approche avec UML pour le « co-design » des systèmes [Yve05].

Cette étude montre qu'il n'existe pas encore un langage unifié permettant une description logicielle abstraite et en même temps performant dans les descriptions du matériel. Un flot entre langages peut permettre une productivité meilleure et minimise le risque de conception avec un langage unique pour satisfaire toutes les exigences d'une conception

basée sur la réutilisation d'IPs de provenances diverses. En effet, dans cette guerre de langages, certains langages coexistent dans un seul flot de conception.

3.7. Bilan : méthodologies de réutilisation de composants

Actuellement la technologie du logiciel fournit des outils satisfaisant permettant la synthèse logique à partir d'une spécification RTL. Mais les tendances mènent vers l'établissement des méthodologies pour rehausser le niveau de la conception ou en utilisant de nouveaux langages tels que SystemC et des nouvelles méthodologies basées sur la spécification UML. Beaucoup de conceptions de SoC comptent sur les protocoles de bus, par exemple, le bus AMBA ou « core-connect ». Cette tendance est poussée par la méthodologie de conception basée sur des plateformes qui impose l'emploi de bus particuliers sur la puce.

Les efforts de standardisation ne répondent toujours pas aux exigences de l'application en terme de protocoles de communication. En plus, les interfaces de communication sont parfois trop générales et utilisent des protocoles trop complexes dépassant les besoins réels de l'application.

Bien que la réutilisation de composants soit améliorée par ces approches, l'intégrateur système doit fournir encore plus d'effort de conception très important, et la performance des composants est plus difficile à prévoir.

3.7. Synthèse de communication

Divers travaux traitent l'intégration des composants à travers une synthèse de la communication logiciel/matériel. Cette synthèse « transforme » les communications à travers des protocoles abstraits en communications sur une architecture cible. Elle permet donc de raffiner les interfaces de sous-systèmes communicants.

La section suivante traite de la résolution des problèmes d'intégration à travers la synthèse d'interface de communication.

4. Synthèse d'interface de communication : SIC

Des travaux pour la synthèse automatique des adaptateurs de communication connectant les composants matériels ayant des interfaces incompatibles ont été déjà proposés. De nombreux travaux ont été menés dans le cadre du « co-design » dans les années 90 : Chinook [Cho95], Vulcan [Gup96], Polis [Chi96], Coware [Rom96], Cosyma [Ern93] etc. Certains avaient pour cible des architectures mono processeurs (Polis, Vulcan, Cosyma) et

d'autres les architectures distribuées avec plusieurs processeurs (Spec Syn, Coware etc.) [Hom01]. Tous ont cherché à automatiser le « co-design » du système complet à partir d'une spécification de haut niveau. Toutefois pour ces outils, les interfaces permettant la communication entre la partie logicielle et la partie matérielle et les interfaces implémentant des protocoles de communication de bas niveau tels que des interruptions, des écritures dans des registres etc. sont conçues manuellement ou bien synthétisées [Cho95].

Actuellement, l'intégration automatique des composants est un aspect clé dans la synthèse des systèmes. La synthèse automatique d'adaptateurs de communication est proposée par divers outils. Pour toutes ces méthodes, le système subit une décomposition fonctionnelle, puis un ordonnancement et une synthèse des interfaces. Cette décomposition est utilisée pour le traitement du système entier. Le système découpé en fonctionnalités est partitionné en logiciel et en matériel.

4.1. SIC dans les outils de « co-design »

Nous nous intéressons ici aux outils qui cherchent à intégrer des composants dans un flot de « co-design ». Trois types d'outils existent : des outils qui essayent de couvrir un flot de conception complet, des outils dédiés à l'exploration d'architectures et d'autres pour la conception d'architectures.

1. L'outil PIG [Pas98] : l'interface des composants dans cet outil est décrite à l'aide d'expressions régulières pour générer une MEF (machine à états finis) correspondante.
2. L'outil POLARIS [Smi98] : il génère un adaptateur basé sur une MEF permettant de convertir le protocole du composant en un protocole standard interne avec des tampons d'envoi et des tampons de réception.
3. L'outil IPchinook [Cho99] : c'est un outil de synthèse pour les systèmes embarqués distribués. Il est orienté vers la réutilisation de composants. L'entrée de IPchinook est une description comportementale de l'architecture cible et une fonction d'allocation qui définit les relations entre les deux descriptions. La description comportementale contient plusieurs modules concurrents et communicants. La description de l'architecture cible décrit les processeurs, les entrées/sorties, les bus de communication et la topologie du système cible. La fonction d'allocation décrit l'affectation des modules aux processeurs du système. Les protocoles de communication abstraits sont synthétisés en des protocoles de bus de bas niveau selon une architecture cible.

Cette description structurée en trois parties permet de changer une partie indépendamment des deux autres.

Toutefois, ces outils ne s'adressent pas aux composants logiciels.

4. L'outil TEReCs [Bok00] synthétise la communication logicielle. Cette approche est associée à la synthèse automatique d'un système d'exploitation configuré selon l'application à partir d'un assemblage de composants.
5. L'outil Paradise [PARA] permet la construction d'un RTOS dédié à partir d'une bibliothèque de composants de base, et la synthèse de communication basée sur des composants et selon une architecture de communication choisie.

Les solutions les plus récentes traitent uniformément les interfaces logicielles et les interfaces matérielles entre les composants.

6. Dans le cadre du projet ESPRIT/OMI-COSY qui résulte d'une collaboration entre l'université de Pierre-Marie Curie de Paris et les laboratoires de Philips, un processus de raffinement de la communication logiciel/matériel est proposé en partant d'un modèle de réseau de processus de Khan étendu pour la spécification des systèmes. Le standard d'interfaces VCI est utilisé pour la conception d'adaptateurs matériels génériques. Dans COSY [Bru00], le système fonctionne par une séparation explicite entre les blocs fonctionnels et l'architecture. Ensuite, les blocs fonctionnels sont affectés aux composants architecturaux. Les interactions entre les blocs fonctionnels sont représentées par des transactions de haut niveau et sont par la suite affectés à des schémas de communication entre les parties logicielles et les parties matérielles. Une bibliothèque fournit un ensemble fixe d'adaptateurs de composants et contient des implémentations en logiciel et/ou en matériel pour des schémas de communication donnés.
7. Prosilog's IP creator [PRO] : la partie Magillem de cet outil vise l'intégration et la réutilisation d'IPs non compatible VCI. Un « wrapper » est généré pour adapter les structures d'IPs communicants. Cet outil permet la génération de « wrapper » à partir d'une description VHDL/RTL de l'interface de l'IP. Le concepteur doit décrire l'interface de l'IP :
    - Il doit lire la documentation technique (datasheet) de l'interface de l'IP
    - Il doit avoir une expérience pour choisir une méthode de description (langage et méthode).

8. Fast prototyping : cet outil [Pog99] utilise un flot de conception système innovateur combinant différentes technologies : la modélisation en langage C, l'émulation, l'outil CowareN2C, le VC hard.

Par ailleurs, les outils de synthèse architecturale sont différents selon le domaine d'application, la classe d'architectures ciblées et les techniques d'optimisation employées. Parmi ces outils, certains ont traité le problème d'interfaçage des modules matériels générés. Nous citons :

9. CoWare Napkin-to-Chip (N2C) : [Ver96] est un environnement de conception et de simulation des systèmes hétérogènes. La communication entre les modules est assurée par des canaux de communication reliant les ports des modules. L'implémentation de cette architecture abstraite consiste à :
   - transposer les modules sur un modèle de processeur et des modèles de composants matériels déjà existants dans la bibliothèque.
   - synthétiser la communication entre les différents blocs.
   - générer les adaptateurs de communication.

   La synthèse de communication dans cet outil consiste à raffiner les canaux de communication abstraits et à choisir un scénario de communication pour chaque canal. La bibliothèque des protocoles de communication contient plusieurs scénarios restreints au protocole « poignée de main » (handshake). CoWare propose une approche basée sur la connexion des composants de l'architecture au réseau de communication à travers des adaptateurs de communication. Ces adaptateurs sont générés automatiquement en sélectionnant dans une bibliothèque les bons éléments à partir des caractéristiques du réseau et du composant à connecter. Toutefois, l'architecture cible de ce flot est fixe. Elle est composée d'un seul processeur connecté à plusieurs composants matériels via un réseau de communication.
10. O'Nils [Nil99] présente une méthodologie de conception d'architecture matérielle à base de processeurs communicants par des protocoles différents. Ces travaux visent la spécification et la génération d'adaptateurs de communication. Mais cette génération n'est pas globalement automatisée. Cependant un environnement de co-synthèse et de prototypage est proposé. Dans cet environnement, la génération d'adaptateurs matériels de composants est adressée par ProGram (Obe99). ProGram présente une nouvelle approche

pour la spécification d'interface matérielle de communication basée sur la modélisation du comportement de l'interface par une grammaire régulière.

11. GAUT [GAUT] : c'est un outil de synthèse comportementale. Il permet aussi la génération des interfaces matérielles appelées Unités de Communication (UCOM). Une nappe de connexions « point à point » reliant un unique processeur à plusieurs accélérateurs matériels est mise en œuvre. La sortie de GAUT est un réseau de modules matériels synthétisables connectés au microprocesseur via des parties logiques ou des adaptateurs. Ces adaptateurs interfacent chaque port de périphérique au port du microcontrôleur. Ils sont sélectionnés à partir d'une bibliothèque matérielle tout en respectant les paramètres de la spécification d'entrée. Bien que cette approche puisse être étendue à l'assemblage de plusieurs IPs, elle se limite à un seul processeur et elle suppose que la spécification de l'application d'entrée ne contienne pas de tâches ou de processus concurrents. Il est basé sur une analyse formelle des communications de l'application et sur leur affectation à des protocoles faisant partie des spécifications.

4.2. Bilan : SIC dans les outils de co-design

Toutes ces approches traitent l'intégration et l'adaptation des protocoles de bas niveau pour intégrer les composants virtuels. Ils proposent des modèles différents et s'appuient sur des techniques différentes de modélisation. D'autre part, un grand nombre d'outils existent. Cependant, ces outils essayant de couvrir un flot de conception complet et des domaines divers d'applications, utilisent des modèles de SoC très simples et très limités. De plus, ces outils permettent d'accomplir avec efficacité seulement des étapes de conception bien particulières telles que l'exploration d'architectures, la conception d'architectures, la simulation, etc.

Par conséquent, outre l'amélioration au niveau outils, il est nécessaire de réduire le gap entre la fréquence du bus de communication (relativement lent) d'une part et la fréquence des éléments de calcul du système. La vitesse à laquelle les données peuvent être consommées dans un système devient l'atout pour garantir plus de performance. Par conséquent, une bonne conception doit être fournie avec un protocole d'interface utilisant la largeur de bus efficacement. La ***« génération automatique d'interfaces »*** doit donc être considérée en prenant en compte les contraintes de transfert des données.

Dans la section suivante, un ensemble de travaux les plus récents qui cherchent à rehausser le niveau de traitement de la communication est présenté afin de favoriser l'interfaçage automatique des IPs dans un flot de conception SoC.

5. Génération automatique d'interfaces de communication : exemples

La conception à base de composants et la conception basée sur une plateforme [Wol02] deviennent obligatoires afin de maîtriser les nouveaux défis qui apparaissent dans les SoCs tels que les NoCs, la communication multiprocesseur etc.

Dans ce contexte, les concepteurs font des choix en limitant le flot de conception SoC ou une étape de ce flot à une classe d'applications (exemple : les applications de traitement de données, les applications de communications, les applications de contrôle etc.) et/ou en considérant des hypothèses sur les architectures cibles, sur les protocoles de communications, etc. Cela permet de proposer des méthodes plus efficaces qui prennent en compte des besoins spécifiques à ce domaine et par conséquent conduisent à un environnement plus réaliste.

Ci après, nous nous limitons à présenter des exemples de travaux ayant la même problématique que cette thèse : la génération automatique d'interface pour l'encapsulation et/ou pour l'intégration d'IPs dans un SoC. Ces travaux reliés à la synthèse de communication permettent l'intégration de composant virtuel en considérant trois aspects : (1) la synthèse d'interface, (2) l'optimisation de la communication, et (3) la conception d'unité de communication (composée d'une partie contrôle et d'une partie de mémorisation) pour l'intégration de composant virtuel. Ils visent à adapter les caractéristiques du système de communication, à savoir, l'aspect protocole et/ou l'aspect de synchronisation et d'ordonnancement, aux exigences du coeur de l'IP. Cela revient à prendre en considération les conditions d'exécution d'IP et les contraintes de son intégration.

5.1. Architecture d'un module d'interface générique

Une définition d'une architecture du module d'interface générique pour une réalisation matérielle est présentée, analysée, et validée dans [Hom01].

Le système considéré est constitué de modules comme des processeurs, des coprocesseurs, des mémoires, des passerelles dont l'interface est précise au bit près. Ces composants sont interconnectés par des signaux dans une architecture à bus partagée. Entre le bus physique et le module d'interface se trouve un « wrapper » permettant l'accès au bus en

mode esclave ou en mode maître. Cette interface matérielle, à interfaçage VCI, intègre les sous-modules suivants comme le montre la figure 7 [Hom01] :

- un module concentrateur d'interruption regroupant les interruptions émises par les modules FIFO (First In First Out) du module d'interface et des interruptions externes permettant de n'utiliser qu'une ligne d'interruption par module d'interface.
- Un module de registres de configuration. Ces registres sont écrits par le processeur via le bus et lus par le coprocesseur sur son interface
- Un module de registres d'état. Ces registres sont lisibles par le processeur. Ils ne sont inscriptibles que par le coprocesseur. Ils sont utilisés pour connaître l'état du coprocesseur.
- Un module FIFO maître d'entrée permettant au coprocesseur de lire des données,
- Un module FIFO maître de sortie permettant au coprocesseur d'écrire des données,
- Un mécanisme d'autorégulation est utilisé entre deux modules d'interfaces communicantes.

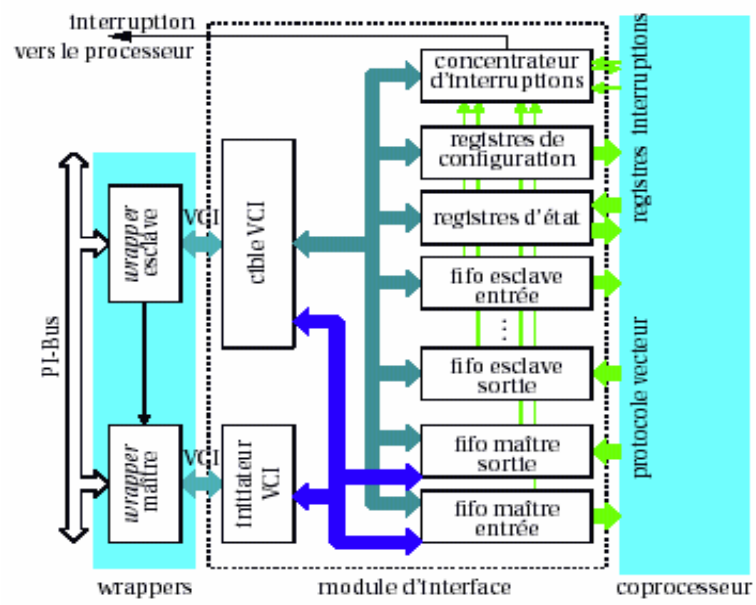

Figure 7. Architecture interne du module d'interface

Il s'agit dans ce travail d'un module d'interface générique synthétisable et optimisé selon les besoins d'un composant coprocesseur à interface défini par un « protocole vecteur ». Chaque sous module de l'interface n'est instancié que si les services utilisés par le coprocesseur l'exigent. L'interfaçage est testé pour le protocole de bus PIbus.

Cette approche présente une solution pour l'intégration de coprocesseur à travers la synthèse de communication pour une approche « Platform Based Design ».

5.2. Adaptation d'IP fonctionnel dans un SoC basé sur un NoC

[Oua04] présente une méthode d'adaptation de l'interface d'IP fonctionnel en considérant un système d'intégration ayant un système d'interconnexion NoC. Cette approche considère l'existence de « wrappers » adaptant le bus au protocole VCI (Cf. figure 8). Dans cette approche, des adaptateurs SystemC conformes au protocole VCI (basique et périphérique) sont définis. Ces adaptateurs sont dédiés pour assurer l'intégration automatique d'IP initiateur ou cible. Ils sont valables pour les différentes couches d'abstraction définies dans le modèle OSI adapté au NoC.

Ce travail favorise un flot de conception « top down » avec le langage SystemC. L'approche utilise la chaîne d'outils CoCentric de Synopsys qui favorise l'utilisation de SystemC en tant que standard unifié de modélisation et de spécification matérielle et logicielle multiniveaux.

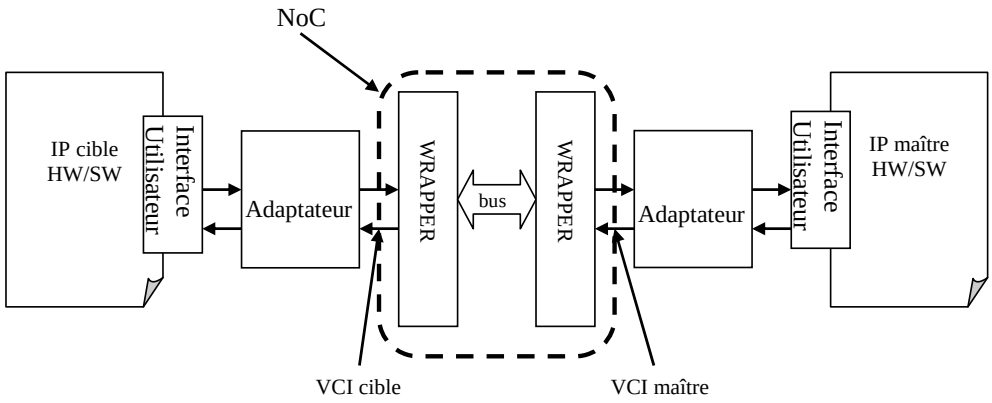

Figure 8. Couches entre des interfaces hétérogènes

Cette approche à base de bibliothèques d'adaptateurs existe dans d'autres approches de « co-design » telles que l'approche COSY. L'application de cette idée est faite pour gérer les couches supérieures dans un NoC pour une approche d'intégration « Core/Component Based Design ».

5.3. Encapsulation automatique d'IP dans l'environnement ROSES

Dans [Yan04], un environnement de conception de SoC est défini où une attention particulière est donnée à la communication entre les composants d'un SoC. L'aspect communication est traité avant le partitionnement.

Un adaptateur de processeur aussi appelé MA pour « Module Adapter » connecté à un bus interne lui-même connecté avec des adaptateurs de canaux aussi appelés CA pour «Channel Adapter». Les adaptateurs de canaux sont spécifiques aux services de communication implémentés. Ils implémentent la partie matérielle du comportement des services de communication. Le composant « interface matérielle » généré par cet outil (Cf. figure 9) est indépendant du processeur utilisé ; il dépend par contre fortement du réseau de communication. Le bus interne est une partie fixe de l'interface matérielle. Il permet de connecter n'importe quel « CA » avec n'importe quel « MA ».

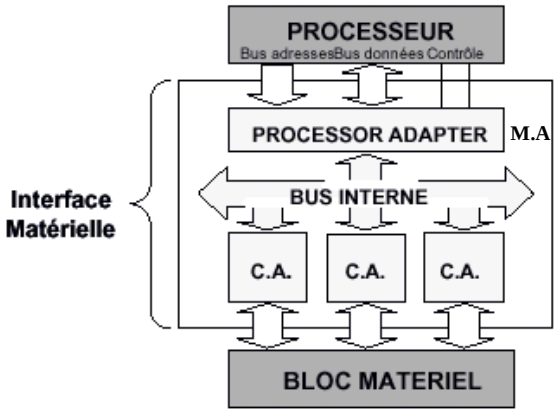

Figure 9. Architecture de l'interface matérielle

Cette architecture favorise une communication point à point et multi points. Toutefois, l'interface est spécifique à une intégration d'IPs dans un SoC dont l'architecture (bus interne et architectures des composants du SoC) est définie par l'outil « ROSES ».

5.4. Génération d'interface pour un système multiprocesseur

Le travail présenté dans concerne un outil de génération automatique d'interface matériel/logiciel dans le contexte du SoC multiprocesseur.

L'IP est considéré comme une entité de trois parties :

1. le modèle de niveau transactionnel (TL : Transaction Level) décrivant sa fonctionnalité,
2. un support du mécanisme d'événements qui fournit un protocole portatif pour l'intégration du cœur de l'IP.
3. une interface de bus agissant en tant que couche d'adaptation pour connecter le cœur de l'IP à une architecture à bus spécifique.

Dans le meilleur cas, si une architecture à bus spécifique change, seule l'interface de bus devrait changer, alors que les deux autres couches demeurent inchangées (Cf. figure 10).

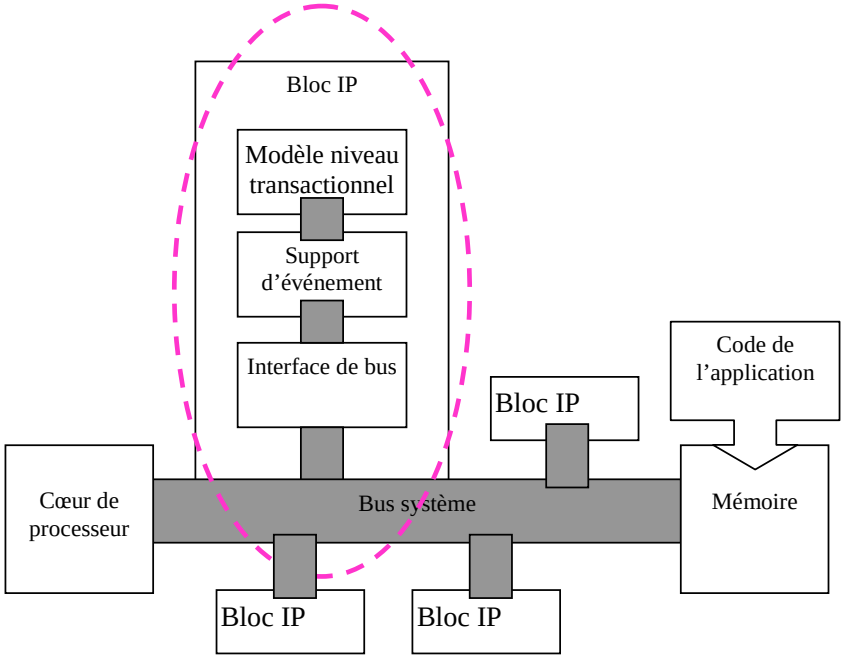

Figure 10. Structure de l'interface d'adaptation de l'IP à une architecture SoC

Cette approche est expérimentée sur un système basé sur le bus du processeur NEC V850. Dans [Laj03], la synthèse automatique des interfaces matériel/logiciel pour ce genre d'architecture est détaillée. Dans [Reg04], cette méthodologie d'intégration est adaptée dans le contexte des SoCs multiprocesseur.

La solution proposée concerne un environnement de conception basé sur des plateformes pour établir un système flexible avec des cœurs d'IPs réutilisables et des unités centrales de traitement.

Toutes ces approches ciblent explicitement l'intégration des composants virtuels en se basant sur la synthèse des communications. Les approches développées dans les paragraphes 5.1, 5.2 cherchent à rehausser le niveau de traitement de l'intégration des composants virtuels tout en utilisant le standard VCI. Les approches 5.3, 5.4 ciblent l'intégration pour un contexte multiprocesseur sans être liés à un protocole de bus. Toutefois l'IP à intégrer possède une architecture spécifique.

Par ailleurs, toutes ces solutions sont de type universitaires. La sous-section suivante présente un outil industriel : l'outil PixelStreams de Celoxica et analyse la solution d'intégration d'IP qu'il propose.

5.5. Intégration d'IP dans Celoxica DK Suite

Celoxica [CXA] délivre des solutions industrielles pour la conception des systèmes électroniques. Pour cela, Celoxica fournit des outils inter-opérables pour favoriser la conception système des systèmes électroniques (Electronic System Level (ESL) design). Les outils de Celoxica couvrent la conception basée sur le langage C (C based design : L'approche technologique et méthodologique incorpore le langage Handel-C et le langage SystemC.) et la synthèse comportementale, le développement matériel sur carte FPGA ou sur des SoCs programmables, des librairies d'IPs et d'APIs au niveau système pour la modélisation co-design, la vérification et l'abstraction de plateforme. Pour cela Celoxica dispose d'une série d'outils tels que :

- Agility Compiler **:** il permet la conception et la synthèse comportementale en SystemC.
- DK Design Suite [DKDS05] : c'est l'environnement de conception système complet pour la simulation et la synthèse permettant l'implémentation de systèmes électroniques. Il favorise un flot de conception SoC à partir d'une description haut niveau basé sur le langage C (C, C++, ou handel C)
- PixelStreams : c'est l'outil qui permet à DK de supporter la conception basée sur la réutilisation (core/ component based design) et l'intégration d'IP pour le développement rapide de système de traitement d'images et de vidéo.

L'outil PixelStreams [PixS06] assure :

- L'importation de code de composant IP RTL issues d'autres fournisseurs sous différents formats (EDIF/VHDL/Verilog) ; l'environnement permet l'implémentation en ciblant aussi bien les technologie FPGA (en générant des fichiers au format EDIF) que les ASICs (en générant des fichiers en VHDL RTL).
- La création des IPs en utilisant des bibliothèques génériques. Les bibliothèques génériques sont des IPs décrites en Handel-C qui ne visent pas un format de sortie particulier. Elles se composent de code compilé qui peut être employé dans un autre programme. PixelStreams dispose d'une bibliothèque de cœurs d'IP paramétrables (filtres, ..) pour la vidéo complexe et les systèmes à traitement d'images permettant de recueillir, de manipuler et de produire des flux de pixels de données visuelles.
- L'emploie de bibliothèques de fonctions de « wrapping » (contrôle de flux, mémorisation intermédiaire, synchronisation, multiplexage/démultiplexage, division en Split etc.), pour simplifier l'interfaçage de ces IPs (code importés, IP

générés à partir des bibliothèques génériques de PixelStreams) pour un prototypage fonctionnel et cycle près.

PixelStreams, utilisé conjointement non parallèlement avec Celoxica DK Design Suite favorise une solution complète pour le développement des systèmes de vidéo basé sur la réutilisation d'IPs.

5.6. Approche proposée

La problématique majeure dans un flot de conception d'un SoC est la définition et la mise en place d'une approche permettant l'implémentation complète d'un système avec toutes ses composantes hétérogènes à partir d'une description de haut niveau.

L'approche proposée cible la génération automatique d'une architecture d'interface pour la simulation et la synthèse en tenant compte du comportement des données à l'interface de l'IP et des contraintes de son intégration ainsi que de l'architecture cible [Abb06a]. L'approche est intégrée dans un outil CAO permettant son automatisation. Il s'agit d'ajouter des blocs de communication pour les entrées et pour les sorties afin de pouvoir simuler le fonctionnement d'un composant matériel dans un environnement de communication intelligent « plug-and-play ». Ce composant peut être synthétisé par un outil de synthèse de haut niveau, implémenté manuellement ou délivré par un fournisseur d'IP. Cette interface est élaborée pour l'intégration d'IPs dans le cas des systèmes dont l'activité est dominée par le transfert et le traitement régulier des données (multimédia, traitement de signal, imagerie, etc.). Ce modèle est suffisamment générique pour permettre une adaptation aux différents protocoles de communication liés à l'interconnecte dans un contexte monoprocesseur ou multiprocesseur.

Par ailleurs, cette approche repose sur un ensemble d'hypothèses pour son application. Une modélisation en graphes est considérée pour la spécification haut niveau de l'interface physique à adapter. L'adoption d'une modélisation à l'aide de graphes d'une part et l'utilisation d'une structure générique de l'interface d'autre part favorisent l'automatisation du flot d'intégration associé à cette approche. Ce flot sera présenté dans le chapitre suivant.

Dans cette thèse, le problème de la génération automatique d'interfaces dans la conception des SoCs est étudié sans être lié à un protocole spécifique ou à un interconnecte spécial ni à une architecture dédiée. L'idée est de résoudre des problèmes de synchronisation et d'ordonnancement liés à l'intégration d'IP considérant les exigences des applications orientées traitement de données. L'approche cible à la fois la simulation et la synthèse. Dans ces conditions, cette approche peut être considérée dans le contexte des méthodologies

d'intégration dites « Component Based Design » et «Platform Based Design» se basant sur la génération automatique d'interface dans un flot de conception système.

6. Conclusion

L'intégration d'IP, considérée l'une des étapes les plus importantes dans la conception des SoCs, exige de tenir compte des contraintes de communication (des caractéristiques d'architecture de communication etc.) et de synchronisation. Ce chapitre a décrit plusieurs méthodologies proposées pour l'intégration des IPs dans le cadre de co-design et de la conception des SoCs.

Une solution importante pour l'intégration [Cou06] des IPs matériels se base sur la synthèse de communication, qui se rapporte à la conception de l'interconnexion entre les composants d'un système et qui couvre un ensemble de tâches. En fait, la synthèse de communication dépend non seulement du flot de conception auquel elle est impliquée, mais du niveau d'abstraction, et peut se rapporter ainsi au choix de la topologie de communication (point à point, « bus based design », etc.), à la définition des modes de transfert (avec ou sans DMA (Direct Memory Access), à la mémoire partagée, etc., ou à la minimisation des coûts de transfert (« overhead » etc.). Dans ce contexte, ce chapitre a décrit également plusieurs solutions adoptées pour l'intégration d'IPs dans un SoC. Ces approches, considérant des hypothèses sur les architectures cibles, sur les protocoles de communications, sur des domaines particuliers, sur les nouvelles orientations architecturales dans les SoCs monoprocesseur ou multiprocesseur etc. permettent de proposer des méthodes orientées mais efficaces.

Dans ce cadre, cette thèse propose une approche d'intégration d'IP dédiée constituée de trois phases principales : (1) la modélisation des contraintes d'intégration, (2) l'analyse des contraintes d'intégration pour la vérification de la faisabilité, et (3) la génération d'une architecture d'interface de communication dédiée pour la simulation et la synthèse.

Dans le chapitre suivant, nous présentons les contraintes liées à l'approche d'intégration proposée. Nous considérons les architectures SoC que peut cibler cette approche, les détails et les hypothèses de son application ainsi que la formalisation des contraintes d'intégration favorisant son automatisation pour la simulation et la synthèse. Basée sur une architecture générique d'interfaces de communication, cette approche permet la génération automatique d'interfaces de communication pour l'intégration d'IP.

CHAPITRE 3. APPROCHE D'INTEGRATION DE COMPOSANT VIRTUEL DANS UN SOC

1. Introduction

Dans le domaine de la conception à l'aide d'IPs, les solutions proposées pour l'intégration des IPs par la communauté scientifique souffrent de plusieurs inconvénients. Il reste toujours des applications pour lesquelles les approches existantes restent inapplicables. Bien que ces approches permettent de réaliser les interconnexions nécessaires, elles conduisent à une consommation excessive en temps et en surface ce qui pénalise le système et le rend incapable d'avoir les performances requises par l'application. En effet, la majorité des méthodes existantes ne permettent pas la génération automatique des structures des schémas de communication. Dans le cas général, certaines méthodes nécessitent une intervention manuelle du concepteur afin d'obtenir des modèles synthétisables par les outils existants que ce soit pour les méthodologies de conception SoC dites « component based design » ou « plateform based design ».

Dans ce chapitre, nous présentons une approche d'intégration d'IPs. Le flot d'intégration associé à cette approche cible la simulation et la synthèse. Il repose sur deux étapes : la modélisation de l'ordonnancement du transfert des données et la génération de l'architecture d'interface correspondante à cette modélisation. Nous présentons dans la section 2 les architectures SoC que peuvent cibler cette approche. Nous avons opté pour l'utilisation d'architecture flexible que ce soit pour un contexte de simulation ou de synthèse. Une telle architecture peut être d'une complexité variable allant d'une simple plateforme correspondante par exemple à une plateforme monoprocesseur, jusqu'à une plateforme complexe hétérogène et multiprocesseur. La section 3 présente les hypothèses adoptées par l'approche proposée. Nous y présentons également les modèles de graphe utilisés pour la modélisation de l'ordonnancement spatio-temporel du transfert des données. Cette étude a permis de définir un modèle architectural générique adéquat pour la génération de la structure de l'interface de communication. Les détails conceptuels des différents modules de cette architecture sont décrits dans la section 4.

2. Flot d'intégration d'IP dans un SoC

L'approche que nous proposons cible la réutilisation de composants matériels existants pour la simulation et la synthèse en s'appuyant sur un modèle d'architecture générique. Une

modélisation en graphes est considérée pour la spécification haut niveau de l'interface physique à adapter. Le but de cette spécification est l'abstraction de la communication du système à un niveau qui est indépendant du protocole et des détails de la technologie, tout en étant expressif et facile à utiliser pour un contexte de simulation ou de synthèse. À ce niveau, tout ce qui est connu concerne les signaux aux entrées/sorties des modules en interaction ainsi que leurs conditions intermodulées du transfert des données et de commande, ce qui est équivalent au comportement du transfert des données.

Les applications orientées flot de données peuvent être spécifiées par un graphe de tâches s'échangeant des données. Le point d'entrée de ce flot est une application cible modélisée par un graphe de tâches cible partitionnées sous forme de tâches matérielles et de tâches logicielles. Les tâches matérielles sont exécutées par des IPs (sous forme d'accélérateurs matériels). Les tâches logicielles sont exécutées par « le reste du système » (Cf. figure 11). La structure interne de ces IPs est le plus souvent inconnue et inaccessible au concepteur système. Seul le comportement spatio-temporel est connu à l'entrée et à la sortie de l'IP. Il décrit comment l'IP échange les données avec le reste du système.

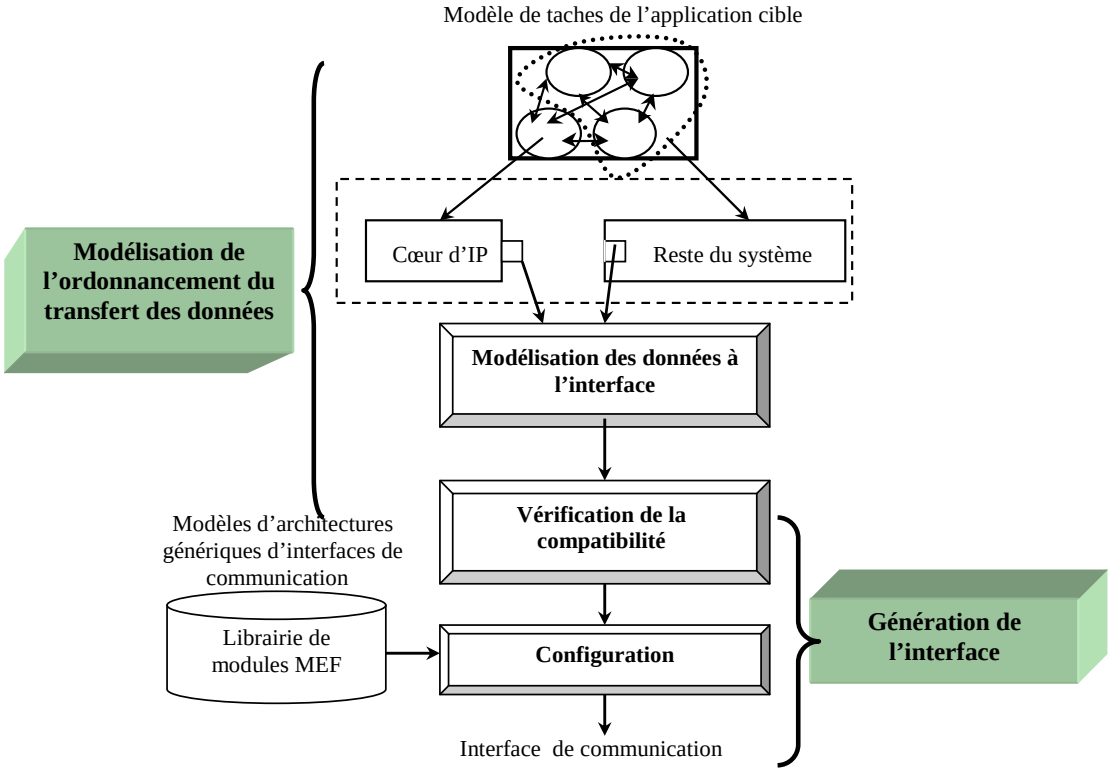

Figure 11. Flot d'intégration d'IP

Afin de pouvoir faire fonctionner correctement l'IP avec le reste du système, nous proposons une approche d'intégration s'articulant sur deux étapes principales (Cf. figure 11 : cube gris) :

1. Modélisation de l'ordonnancement du transfert des données.
2. Génération de l'interface.

2.1. Modélisation de l'ordonnancement du transfert de données

L'ordonnancement des données consiste à identifier sans ambiguïté les données en entrée (resp. en sortie) à chaque cycle de l'horloge de chaque port d'entrée (resp. port de sortie). Il n'est pas obligatoire que le système respecte parfaitement l'ordonnancement imposé par l'IP. Il suffit de respecter certaines conditions. Une interface est ajoutée entre l'IP et le reste du système. Elle assure un échange de données selon les modèles d'ordonnancement de l'IP d'une part et du reste du système d'autre part. L'ajout de cette interface n'est possible que si l'IP et le reste du système sont compatibles. Cette vérification de compatibilité permet de réaliser un compromis entre la complexité de l'interface d'une part et le nombre d'IP qu'elle peut intégrer d'autre part. En effet, comme l'IP et le reste du système respectent déjà certaines conditions pour l'échange de données, l'interface ne sera pas d'une complexité élevée. D'autre part, vu que le reste du système peut ne pas suivre l'ordonnancement des données de l'IP, l'intégration d'une large gamme d'IPs est possible.

2.2. Génération de l'interface

L'intégration d'un IP de l'approche proposée est un processus de génération d'interface assisté. Elle est basée sur la compatibilité entre le système intégrant et l'IP à intégrer. L'analyse de la compatibilité est valide si les hypothèses considérées par la méthodologie sont vérifiées. Une fois la compatibilité de l'intégrité vérifiée, la deuxième étape du flot consiste à la génération de l'architecture de l'interface. Cette architecture est définie par des modèles de MEFs paramétrables inclus dans une librairie. Sa génération consiste alors à :

- sélectionner et configurer les MEFs sélectionnées selon la modélisation des données à l'interface.
- générer le pilote logiciel correspondant qui permet de gérer l'interface matérielle.

Ces étapes d'intégration sont détaillées dans les sections suivantes. Dans la sous-section suivante, nous détaillons les architectures que cible l'approche proposée.

2.3. Architectures cibles

A la différence des systèmes classiques, les systèmes monopuces sont conçus pour des applications spécifiques (télévision numérique, GPS, etc.) et sont taillées sur mesure pour satisfaire les besoins de l'application. Pour assurer son autonomie, un système mono puce, tel que nous le définissons, est composé de plusieurs éléments logiciels et matériels dont les fonctionnalités sont très complexes. Un ou plusieurs processeurs intégrés exécutent la partie logicielle. La partie matérielle est constituée, en plus des processeurs, de différents blocs opérationnels qui sont soit développés soit achetés tels que les IPs. Ces composants exécutent une partie de l'application beaucoup plus rapidement et en consommant moins d'énergie que si elle était exécutée par le processeur. Comme le montre la figure 12, ces IPs peuvent être des processeurs de natures différentes (des microprocesseurs, des processeurs de traitement de signal (DSP), des coprocesseurs ou des accélérateurs matériels). Nous distinguons la définition d'un accélérateur matériel et d'un coprocesseur [Aou04] :

1. La notion de coprocesseur consiste à ajouter une instruction de plus au jeu d'instruction prédéfini du processeur hôte. Il est connecté directement à l'unité arithmétique et logique (UAL) du microprocesseur. Avec ces instructions spécialisées (Custom Instructions), les concepteurs peuvent réduire une séquence complexe d'instructions standard à une instruction simple implantée en tant que matériel.
2. Les accélérateurs sont considérés comme des boîtes noires pour le système. Ils opèrent parallèlement avec l'ensemble du système communiquant à travers l'interconnecte. Cet interconnecte est le moyen de communication permettant de relier le processeur, l'IP, la mémoire et aux autres modules du SoC. Par conséquent, pour utiliser un accélérateur, le microprocesseur envoie les données nécessaires à ce dernier et attend le résultat. Le calcul est fait en dehors du microprocesseur ; et ainsi celui-ci peut continuer d'exécuter le code tandis que l'accélérateur fonctionne.

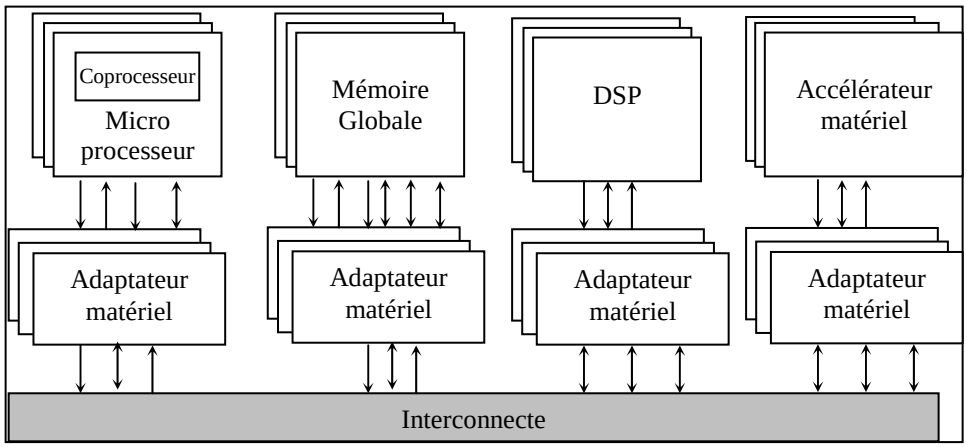

Figure 12. Architecture d'un système mono puce

La différence entre l'accélérateur et le coprocesseur est qu'un accélérateur matériel peut accéder aux autres accélérateurs ou à la mémoire dans le système sans intervention du microprocesseur. Le coprocesseur communique avec le microprocesseur en utilisant des registres spécifiques internes sans passer par les bus externes du microprocesseur comme le fait l'accélérateur.

Un SoC comporte également des mémoires, des réseaux de communication complexes et des adaptateurs matériels de communication. La partie logicielle est composée du programme d'application et du système d'exploitation temps réel. Ce système d'exploitation est formé d'un ensemble de primitives logicielles qui facilitent la gestion et le partage des ressources matérielles. Cette architecture hétérogène permet d'obtenir des systèmes performants pour les diverses applications cibles (orientés flot de données ou de contrôle).

La complexité relative des architectures des SoCs a poussé les chercheurs à définir de nouvelles méthodes de conception en s'inspirant de celles développées pour la conception de circuit. En fait, les méthodes de conception des circuits intégrés ne sont pas adaptées aux systèmes monopuce [Abi00] : elles ne visent, en effet, que la réalisation de la partie matérielle. Des méthodes de conception des SoC ont été proposées. Ces méthodes utilisent les techniques de conception des circuits intégrés pour la partie matérielle. Elles les précèdent, par contre, par des étapes visant, à partir de l'application et de ses contraintes, à déterminer ce qui doit être fait en matériel et ce qui doit être fait en logiciel, à concevoir l'architecture de la partie matérielle et à permettre la réalisation des communications entre la partie matérielle et la partie logicielle.

Actuellement les travaux de conception de SoC peuvent être classés en trois scénarios [Zit01] (Cf. figure 13). Le premier consiste à sélectionner tout d'abord une architecture puis à chercher le partitionnement optimal. Le deuxième assure en premier lieu le partitionnement

puis cherche l'architecture optimale. Le dernier assure le partitionnement et la sélection de l'architecture d'une manière interactive.

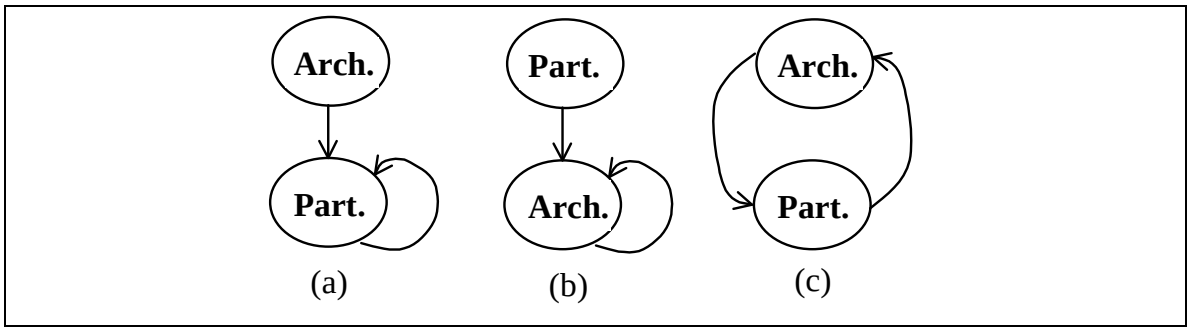

Figure 13. Différents scénarios de la conception de SoC

Pour couvrir les différents contextes de l'application de l'approche d'intégration (mono et multiprocesseur), nous supposons que le modèle d'architecture cible peut supporter des applications partitionnées en des tâches logicielles et des tâches matérielles. Il permet également différents mécanismes de communication entre les tâches (par registre, par bus, par mémoire partagée). La figure 14 montre un exemple monoprocesseur dans lequel quatre tâches (T1, T2, T3, T4) communiquent entre elles selon diverses partitions :

– trois tâches logicielles et une tâche matérielle (figure 14 (a)),

– deux tâches matérielles et deux tâches logicielles (figure 1' (b), figure 15 (a, b))).

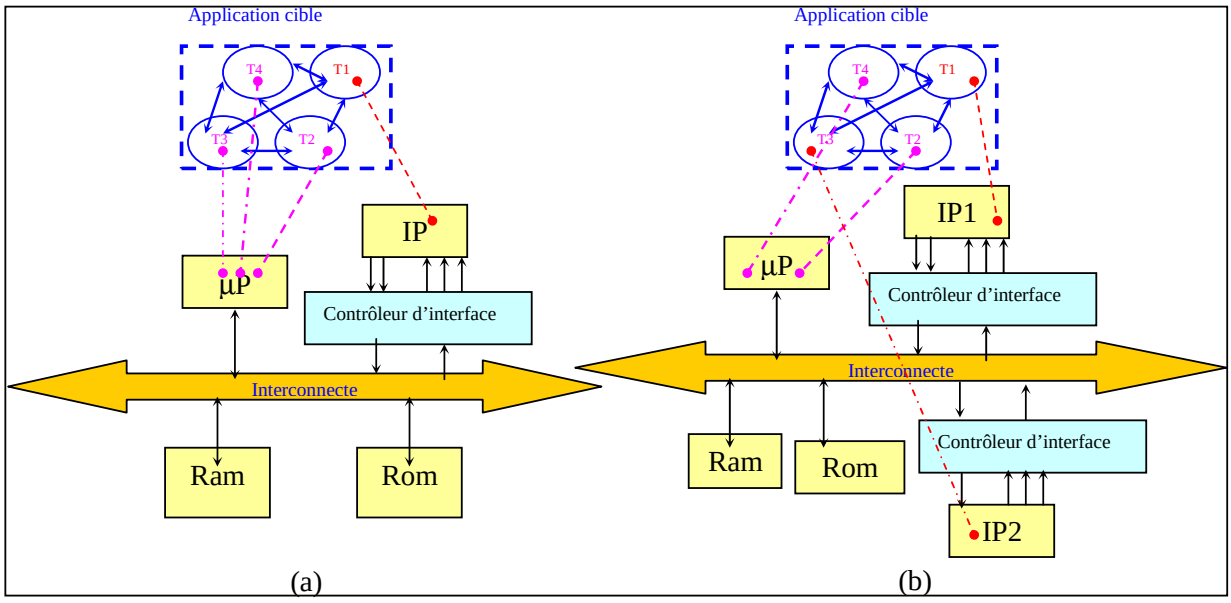

Figure 14. Modèles d'architecture Monoprocesseur

Le contexte multiprocesseur mono puce est considéré dès qu'une application (ensemble de tâches communicantes entre elles) peut s'adapter à une architecture contenant au moins deux processeurs. Le contexte multiprocesseur est traité selon le degré d'indépendance des différentes tâches de l'application (en terme de partage de ressources

matérielles ou logicielles, entre tâches, entre IPs en général) (Cf figure 15.a, figure 15.b). L'efficacité d'une topologie de multiprocesseur dépend du degré de parallélisme dans l'application (c'est à dire le nombre de tâches indépendantes parallèles).

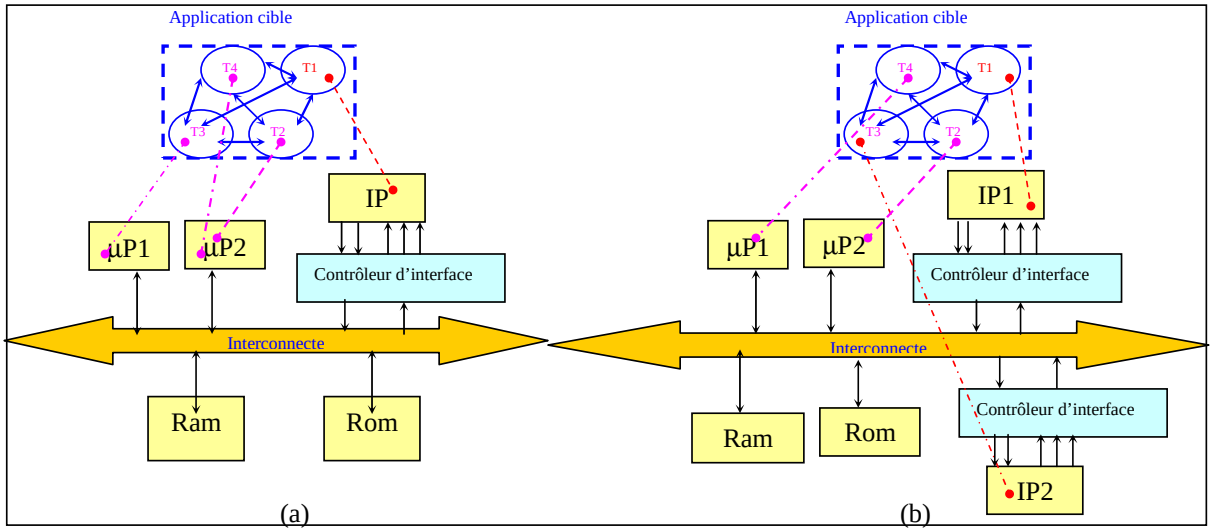

Figure 15. Modèles d'architecture Multiprocesseur

Diverses architectures possibles peuvent correspondre au contexte multiprocesseur. Si aucun espace d'adressage partagé n'est utilisé, les processeurs peuvent communiquer à travers une FIFO par exemple. Dans le cas de la figure 16, une mémoire FIFO est utilisée entre les deux microprocesseurs si nous supposons que le microprocesseur 1 utilise une information traitée par le microprocesseur 2 (figure 16 (a)). La même interprétation est valide pour la communication entre les deux IPs (figure 16 (b)).

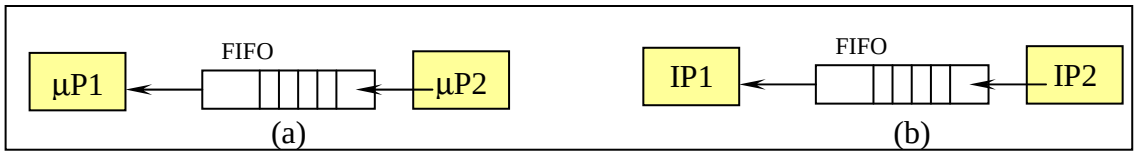

Figure 16. Communication à travers des mémoires FIFOs

Vu la diversité des architectures cibles, l'approche d'intégration d'IPs proposée doit être générique pour pouvoir s'adapter au maximum des cas possibles. Elle utilise un niveau d'abstraction intermédiaire entre le haut niveau et le niveau transfert de registre ou le niveau micro architecture dans le flot de conception. Elle permet de transposer une spécification de haut niveau sur une architecture cible. L'interface utilisée par notre approche supporte l'adaptation au niveau « cycle près bit près» à l'interface. L'ordre d'envoi et de récupération des données définit la structure générale de l'architecture de l'interface.

L'approche proposée fait partie du deuxième scénario (figure 13 b) (après le partitionnement). En revanche, le fait qu'elle peut cibler la simulation et la synthèse la rend modulaire et flexible. Elle peut donc supporter d'éventuelles extensions pour minimiser le surcoût de communication qui peut être introduit par le partitionnement.

L'avantage de cette approche consiste, par ailleurs, en l'obtention des structures RTL des schémas de communication afin de permettre la synthèse par les outils existants. Malgré la difficulté de trouver une plateforme à usage universel, nous avons opté pour l'utilisation d'architecture modulaire et flexible que ce soit pour un contexte de simulation ou de synthèse de l'interface de communication. Une telle architecture peut correspondre à une simple plate-forme, comme à une plateforme complexe hétérogène et reconfigurable. Chaque plateforme sert à l'intégration du système global mixte logiciel/matériel et peut supporter plusieurs variétés d'applications complexes.

Nous analysons dans la suite de ce chapitre les deux étapes du flot : l'étape de modélisation de l'ordonnancement des données et l'étape de la génération de l'architecture de l'interface.

3. Modélisation du transfert des données

L'architecture SoC cible inclut des composants matériels qui représentent les unités de calcul sans logique de commande externe pour être autonome.

Il existe trois familles d'IPs matériels suivant leurs niveaux de flexibilité (adaptation aux besoins de l'utilisateur) (Cf. figure 17) : [VSIA]

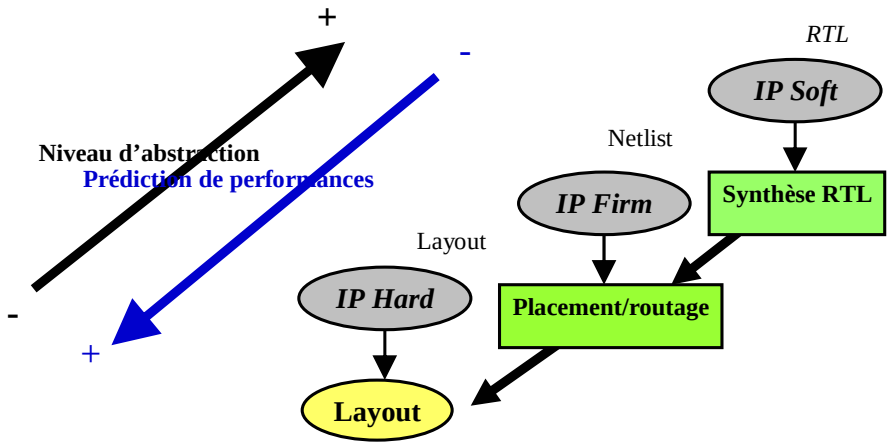

Figure 17. Familles d'IP et caractéristiques associées

1. IP logiciel « soft » : composant virtuel formé par un code HDL synthétisable ce qui assure à ces IPs une indépendance totale vis à vis de la technologie (ASIC ou FPGA). Ils sont très peu prédictibles, et très flexibles.

2. IP « firm » : composant virtuel optimisé structurellement et typologiquement mais non routée. Ils reposent sur l'emploi de technologies génériques et sont donc indépendants de la taille des transistors et de la technologie du fondeur. Ils sont peu prédictibles, car l'optimisation au niveau logique n'a pas encore eu lieu, et supportent un degré de flexibilité qui autorise la personnalisation non seulement de la technologie cible et des contraintes d'optimisation logique, mais aussi de paramètres architecturaux.
3. IP matériel « hard » : composant virtuel fourni sous la forme d'un réseau de portes logiques pouvant être optimisé pour une bibliothèque technologie générique. Ils ne sont pas flexibles mais très prédictibles.

Nous ciblons les applications opérant sur des flux importants de données telles que les applications TDSI et multimédia. Ces applications sont généralement spécifiées comme un ensemble de tâches avec des contraintes temporelles globales aux entrées et aux sorties. Les IPs accélérateurs pour ces applications sont donc synchronisées par les données. L'ordre dans lequel les données sont transférées est alors très important pour piloter le composant virtuel de manière efficace.

Comme les IPs sont de provenances diverses, leurs ordonnancements sont variables et ne correspondent pas forcément à celui du système intégrant. Afin de pouvoir interfacer un IP dans un système, il est nécessaire donc de respecter l'ordonnancement spatio-temporel des données aux entrées/sorties de l'IP et de l'adapter à celui du reste du système. Une interface externe est intercalée entre les deux pour cette finalité. Toutefois, pour pouvoir l'appliquer, il est nécessaire de respecter certaines hypothèses sur l'ordonnancement des données utilisées pour la définition de la structure de l'architecture de l'interface. Ces hypothèses sont détaillées dans la sous-section 3.1.

3.1. Hypothèses sur l'ordonnancement des données

L'ordonnancement des données entre l'IP et le reste du système peut être faite de diverses manières. Il dépend de l'IP, du protocole de l'interconnecte et du reste du système. Afin de limiter la complexité de l'interface, il est nécessaire d'introduire certaines hypothèses. Ces hypothèses, bien que certaines parmi elles soient restrictives, ne limitent en rien la généricité de l'approche comme nous allons l'expliquer. Ces hypothèses sont les suivantes :

– L'interface possède un seul port d'entrée défini par un ordre d'envoi et un seul port de sortie définissant l'ordre de réception du côté système (Cf. figure 18). Un

IP peut avoir plusieurs ports d'entrées et/ou de sorties. Dans la figure 18 par exemple, le module IP possède deux ports d'entrées et un port de sortie.

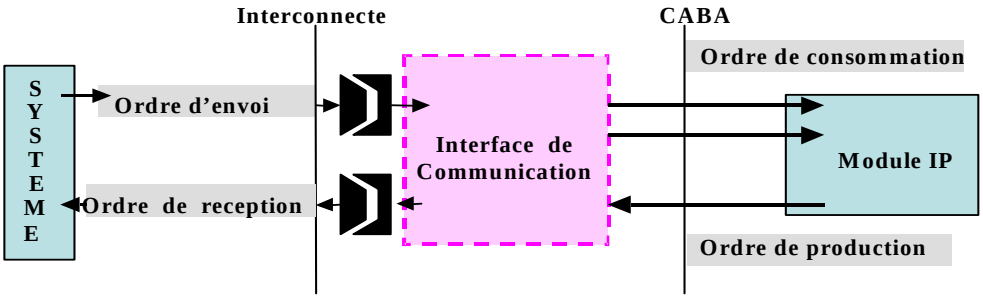

Figure 18. Chemin de communication

– L'ordre d'envoi et l'ordre de consommation sont les mêmes pour un flux transitant sur un port de données : les données envoyées par le système pour l'IP doivent suivre un motif. Dans les applications ciblées, les données sont regroupées généralement par ensemble (ou structure). La compression vidéo par exemple se fait sur des macro-blocs 8x8 ou 16x16 pixels, le filtrage d'image se fait sur des masques 3x3 ou 5x5 pixels. Pour l'application « produit matriciel L x C » (L : ligne, C : colonne), une structure de donnée est une matrice de donnée (le transfert de donnée est considéré avec trois structures de données : deux structures d'entrée et une structure de sortie). Si le système peut ne pas respecter l'ordre exact des données requis par l'IP, il doit néanmoins respecter l'ordre d'envoi des structures de données (macro bloc par macro bloc ou matrice par matrice : ce que nous appelons motif ou « pattern »). Les données sérialisées par le système prennent la répartition spatiale parallèle suivant ce motif. Une itération de calcul définit une exécution pour une séquence de donnée de base pour le traitement de l'IP. Une itération de calcul commence avec le début de la consommation de la première donnée à traiter et se termine avec la production de la dernière donnée de la matrice à sortir pour l'exemple produit matriciel.
– L'IP à intégrer est esclave tandis que le reste du système (formé essentiellement par le processeur) est le maître du système. C'est lui donc qui envoie les données et réclame les résultats de l'IP.
– Les données ont une largeur de la forme 2^n (où n est un entier) qui divise la largeur de l'interconnecte du reste du système.
– Les entrées et les sorties sont ininterrompues, l'IP n'arrête le traitement que si les données ne sont plus disponibles. L'échange des données entre l'IP et le reste du système suit un comportement asynchrone. Nous introduisons un instant de début

et un instant de fin virtuels : la différence de ces instants donne le temps nécessaire à l'IP pour le traitement des données relatif à une itération de calcul. Le temps d'envoi des données par le système dépend de son fonctionnement (déclenchement d'interruption, exécution de tâches ordonnancées).

Nous définissons une interface spatio-temporelle par :

- une interface spatiale définissant la succession spatiale des données en entrée ou en sortie.
- une interface temporelle définissant les temps associés à cette interface spatiale.

Le problème de la modélisation de l'interface spatio-temporelle associée à l'ordonnancement des entrées/sorties peut être donc traité suivant la cadence du système.

3.2. Ordonnancements des données aux entrées sorties de l'IP

L'IP ciblé est à interface CABA (Cycle Accurate Bit Accurate). Pour qu'un IP fonctionne correctement, il est nécessaire de lui fournir les données adéquates aux instants appropriés.

Comme l'IP peut travailler à une fréquence différente du reste du système, nous différencions dans cette étude :

- l'ordre de consommation et l'ordre de production des données à l'interface de l'IP.
- l'ordre d'envoi et l'ordre de réception des données par le système.

Il est possible que ces deux ordres coïncident. Ils peuvent également être différents même si le concepteur du système l'exige comme par exemple dans le cas d'un système multiprocesseur sans «timer ». Toutefois, dans ce genre de système l'ordre dans lequel un flux de données transite sur un port de l'IP doit être connu par le concepteur système.

Pour mieux expliquer comment se fait l'adaptation entre les ordres, nous considérons un exemple didactique simple à travers lequel nous expliquons notre solution. Le système à considérer possède 4 entrées « a », « b », « c » et « d », les sorties du système sont « x », « y » et « z ». Les traitements effectués par le système sont représentés par le système d'équations suivant :

$$\begin{cases} x=a+b \\ y=c+d \\ z= a+y \end{cases}$$

Dans la configuration de la figure 19, l'interface temporelle ne pose pas de contraintes sur la cadence du système. En effet, le calcul des sorties s'effectue avant l'arrivée de

nouvelles entrées (« a' », « b' », « c' » et « d' »). Alors que dans le deuxième cas de figure (figure 20), il y a la possibilité d'écraser la valeur de « a » avant que la sortie «z » ne puisse l'utiliser. La solution consiste à utiliser une structure à base de FIFOs pour la mémorisation de la succession des variables d'entrée.

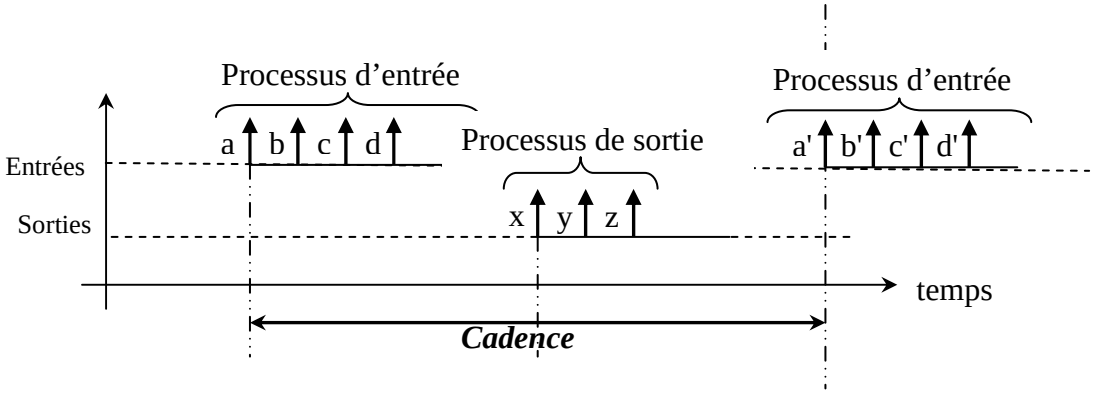

Figure 19. Configuration d'ordonnancement 1

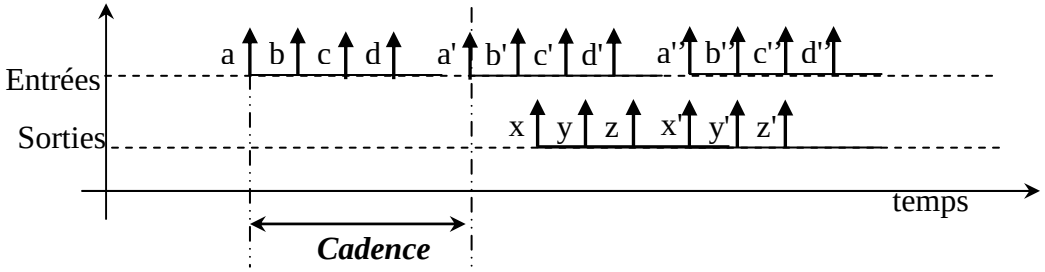

Figure 20. Configuration d'ordonnancement 2

Dans ces deux configurations (figure 19, figure 20), si nous voulons unifier la solution, des FIFOs peuvent être intercalées entre l'interface de communication de l'interconnecte (structure de communication du système) et l'Unité de Traitement (UT) de l'IP afin de pouvoir gérer la différence temporelle. Ces FIFOs sont directement connectées aux ports du composant virtuel.

Dans un troisième cas illustré par la figure 21, nous avons besoin de présenter les entrées selon l'ordre voulu par le traitement. Prenons par exemple l'entrée « a » : la séquence que l'UT de l'IP a besoin est « a, a', a'', a, a', a''…..», le problème figure dans la manière d'accéder aux données dans les FIFOs (figure 21) : comment gérer l'accès aux données dans cette configuration ? Comment gérer l'interface temporelle « donnée fournie par le système, donnée consommée par l'UT » ?

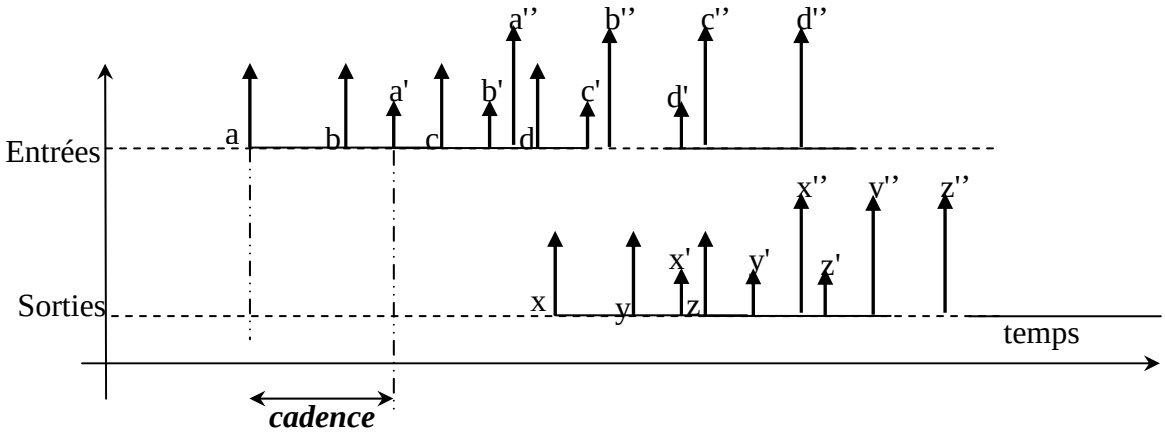

Figure 21. Configuration d'ordonnancement 3

Par ailleurs, pour les données en entrée ou en sortie, nous devons définir leur ordre de rangement dans la FIFO (figure 22).

Structure FIFO entrée	
@	a" → a' → a
@	b" → b' → b
@	c" → c' → c

Figure 22. Structure FIFO

Dans ce cas, pour satisfaire les contraintes sur les informations en consommation ou en production des données, une machine d'état associée à l'interface de l'IP permettra d'accéder aux données en entrée sauvegardées dans les FIFOs selon le besoin de l'UT.

A travers cet exemple, nous allons présenter les caractéristiques de l'interface de communication afin de pouvoir gérer la différence entre l'ordre d'envoi/réception des données par le système d'une part et l'ordre de consommation/production de l'IP d'autre part.

3.3. Interface de communication

La prise en compte de toutes les possibilités d'ordonnancement des données signifie une complexité trop grande de l'interface générée. Ceci peut générer une architecture d'interface "trop programmable" difficile à implémenter en matériel. Nous avons opté plutôt pour l'utilisation d'une interface générique configurable selon les besoins applicatifs. Elle possède deux côtés : le côté matériel qui représente l'architecture de l'interface et le côté logiciel formé par son pilote.

Nous avons montré qu'il est nécessaire pour la définition de la structure de l'interface de communication (cf. figure 23) :

– d'intercaler entre l'interface du protocole de l'interconnecte et l'UT de l'IP des FIFOs afin de gérer la différence temporelle entre l'ordre de données fournies par le système et l'ordre des données consommées par l'UT. Ces FIFOs sont directement connectées aux ports du composant virtuel.
– une unité de contrôle permet d'accéder aux données d'entrée (de sortie) mises en FIFO.

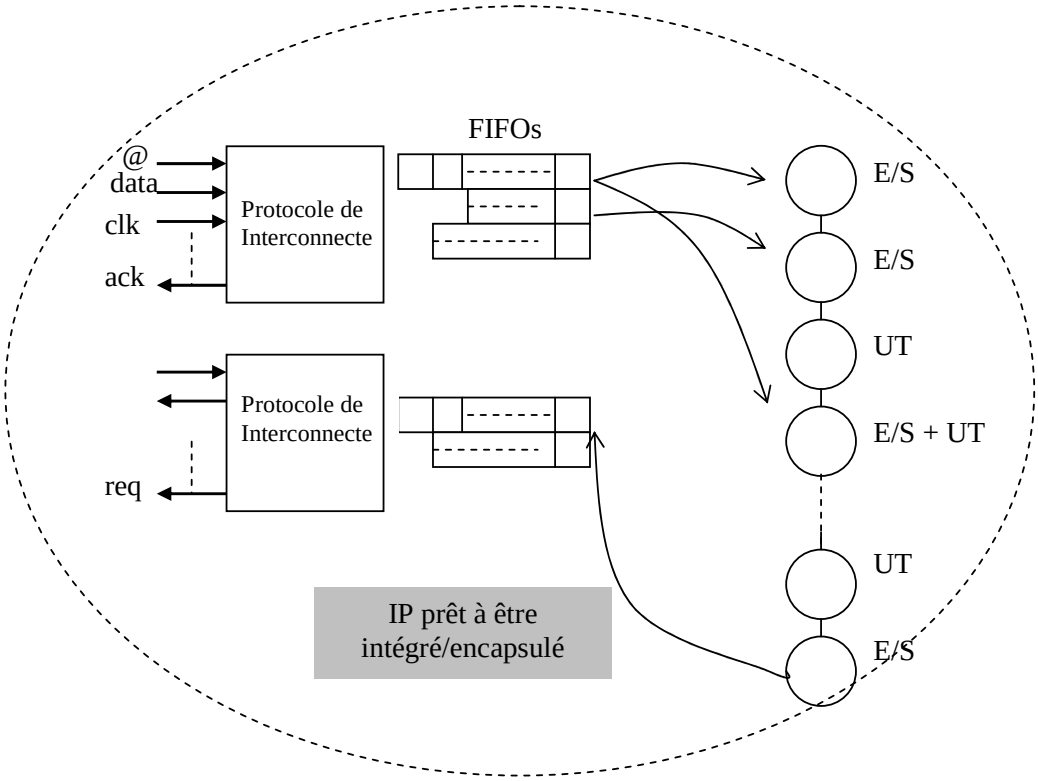

Figure 23. Structure du modèle d'intégration d'IP

En conséquence, l'ordre de lecture des données peut être aléatoire du côté de l'IP. Les données sérialisées à l'entrée de l'interface peuvent être consommées dans un ordre aléatoire : c'est l'interface qui ordonne ces données (aiguillage de chaque donnée vers le port associé) suivant la caractérisation du transfert des données à l'interface de l'IP. Grâce à cette hypothèse, l'interface générique peut s'adapter aux différents types d'IPs ayant des caractéristiques différentes. Les données à traiter sont de tailles variables : un même IP peut être capable de traiter des données de différentes tailles. Cela permet d'utiliser un seul IP pour des applications complexes multimédia telle que la compression vidéo où les données audio sont sur 8 ou 16 bits, les données représentant des pixels d'images sont sur 16, 24 ou 32 bits.

Grâce à ce module d'interface de communication, il est possible de réutiliser des IPs systématiquement (sans un effort supplémentaire d'adaptation) dans un environnement

intégrant spécifique dont l'interconnecte est connu (comme par exemple un système utilisant le bus AMBA).

Afin de modéliser les échanges des données entre le système et le composant virtuel, nous définissons des modèles de graphes qui seront présentés dans la sous-section 3.4. Cette modélisation en graphes est utilisée pour la spécification abstraite de l'interface.

3.4. Modélisation en graphes des contraintes sur les entrées/sorties de l'IP

L'idée principale de l'approche proposée est de modéliser les contraintes sur le transfert des données sous forme de modèles de graphes abstraits en considérant le système intégrateur et l'exécution d'IP.

La sémantique des modèles de graphes abstraits adoptée dans notre approche est inspirée des travaux de Ku et De Micheli [Deu92]. Elle se base sur la notion de graphe de contraintes d'entrées/sorties IOCG (IO Constraint Graph) et sur le modèle d'IPERM (IP Execution Requirement Model) proposé par [Cou03b]. Ces modèles permettent, entre autre, d'exprimer :

- les contraintes temporelles liées au transfert des données comme la spécification des temps et les délais de temps,
- les variations temporelles des dates de transfert des données,
- du séquencement des données échangées (contraintes spatiales)

Cette spécification est adoptée pour la formulation et la définition des modèles de graphes. Ces modèles de graphes sont abstraits et permettent d'exprimer en totalité le comportement aux entrées/sorties à l'interface de l'IP pour la configuration de la partie contrôle de l'architecture de l'interface de communication et du modèle de transfert des données du système.

Nous détaillons dans la suite les différents modèles de graphes développés.

3.4.1. Graphe d'Ordonnancement des Entrées/Sorties : GOES

GOES est un graphe d'ordonnancement des entrées/sorties (Cf. figure 25). Il représente le comportement des entrées/sorties de l'interface de l'IP. Ce modèle de graphe noté par GOES (V,E) est hiérarchique (contenant des nœuds hiérarchiques) pondéré (contenant des transitions) polaire orienté. Il est formé par un ensemble de nœuds V et un ensemble d'arcs E reliant ces nœuds.

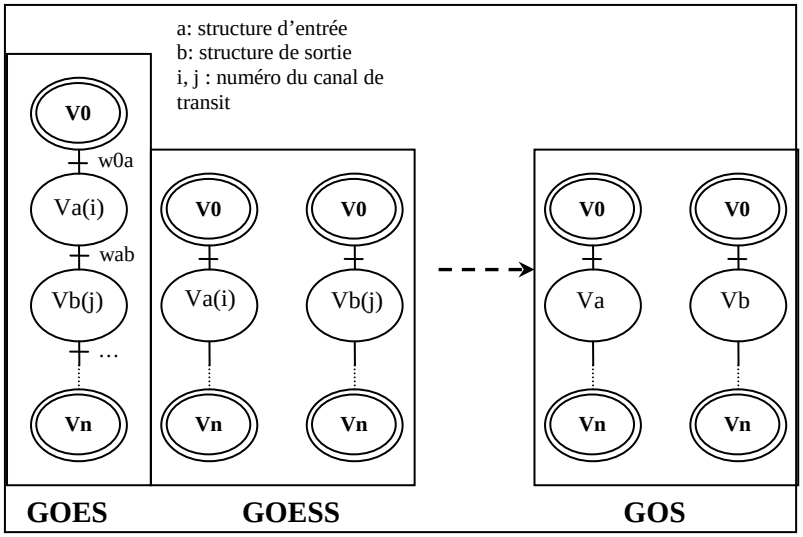

Figure 24. Modèles de graphes

V est l'ensemble des nœuds : V = {v0, ..., vn}. V représente les étapes de transfert de données. Chaque élément de l'ensemble des données consommées/produites est doté du numéro de port de transit. v0 et vn sont respectivement le noeud source et le noeud puits et symbolisent le début et la fin d'une séquence de transfert.

E est l'ensemble des arcs du graphes, E= {(va,vb)}. Il représente le séquencement des transferts spécifiés par les délais correspondants aux différents temps de transfert. A chaque arc « Eab » est associé un poids « wab ». Il représente le délai séparant deux états entre deux transferts de données va et vb. Le délai est exprimé en unité de temps. Le graphe GOES est défini en supposant que le système traite les données sans interruption.

3.4.2. Graphe d'Ordonnancement aux Entrées/Sorties par Structure : GOESS

A partir du graphe GOES, nous pouvons déduire « m » graphes d'ordonnancement des entrées/sorties par structure où «m » désigne le nombre de structures. En éliminant la contrainte sur les délais pour chaque structure de données du graphe GOES, nous définissons un deuxième type de graphe appelé : GOESS. Ce type de graphe sert à la configuration de la partie contrôle de l'architecture de l'interface de communication.

3.4.3. Graphe d'Ordonnancement aux entrées/sorties du Système (GOS)

Du côté système, nous proposons un graphe déduit du modèle de graphe GOESS appelé GOS. Ce type de graphe définit l'ordre selon lequel le système doit envoyer ou consommer les données de chaque structure. Le GOS est déduit à partir des graphes GOESS en éliminant les contraintes sur les délais, sur l'information de l'indexage du canal de transit. Le nombre de nœuds du GOES et celui du GOS peuvent être différent.

Ces modèles de formulation en graphes peuvent cibler l'intégration des architectures à IPs synchrones ou asynchrones. En effet, la contrainte de temps à l'interface de l'IP est exprimée par un délai représentant un temps de transfert exprimé en unité de temps.

Ces graphes définissent la modélisation générique permettant l'automatisation de la configuration et le paramétrage des éléments de l'interfaçage d'un IP accélérateur dans un SoC à savoir le « driver » et les sous modules de l'architecture de l'interface matérielle décrite dans la section 5. Cette automatisation sera détaillée dans les sections qui détaillent l'étape de vérification de la compatibilité entre graphes et l'étape de configuration pour la génération de l'interface.

3.5. Vérification de la compatibilité

Nous notons que les modèles des graphes de types GOESS, et GOS sont définis à partir du même modèle de graphe : GOES (cf. figure 25). Ils sont donc compatibles. Cette compatibilité valide la possibilité d'appliquer la deuxième étape de l'approche d'intégration.

La vérification de la compatibilité (cf. figure 26) réside simplement dans la vérification d'un motif caractérisant l'ordre d'envoi et l'ordre de réception des données par le système (GOS). Ce motif doit vérifier un séquencement qui satisfait au moins l'ordre par structure définissant le comportement des entrées sorties à l'interface de l'IP. Pour cela, deux côtés sont considérés : le comportement à l'interface de l'IP (modèle de graphe GOES) et le transfert des données à partir du système (modèle de graphe GOS). Pour cela, une base de graphes de type GOS est générée automatiquement décrivant la totalité des possibilités qui peuvent être compatible avec le modèle de graphe GOES.

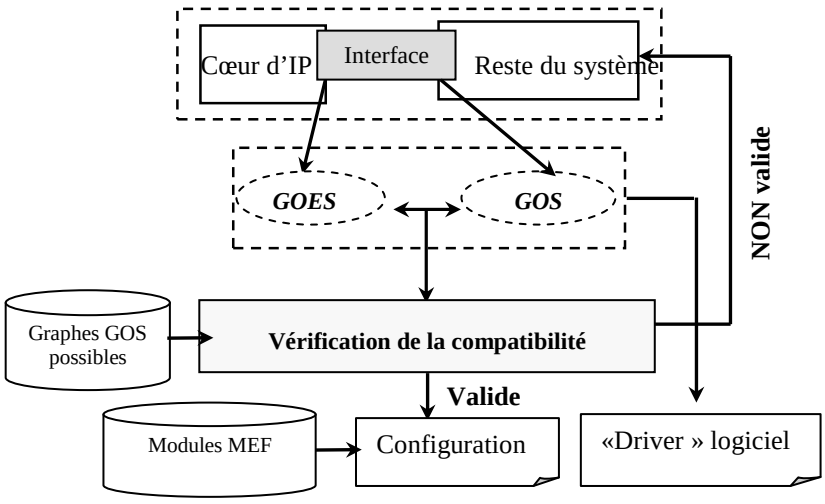

Figure 25. Vérification de la compatibilité

La vérification de la compatibilité consiste à comparer le GOS du système un à un avec chacun des éléments de la bibliothèque des graphes GOS possibles qui peuvent garantir l'intégration de l'IP considéré dans le système cible.

Exemple :

Nous considérons le motif de pseudo ordre (A,B,C,C,B,C,C,B,C,C,B,C,C) pour une IP à deux entrées A et B et une sortie C.

Si le motif est fixe pour toutes les itérations de calcul, la bibliothèque des graphes GOS possibles contient un seul graphe : le graphe correspondant à ce motif.

Si le motif n'est pas fixe (pseudo ordre considérant l'ordre par structure), alors la bibliothèque des graphes GOS possibles contient les graphes correspondants au motifs où A se répète une fois, B se répète 4 fois, C se répète 8 fois soient :

Nombre de graphes possibles= $C_{13}^{8} \times C_{5}^{4} \times C_{1}^{1}$

Dont :

(A,B,B,B,B,C,C,C,C,C,C,C,C),

(A,B,C,B,C,B,C,B,C,C,C,C,C),

(B,B,B,B,A,C,C,C,C,C,C,C,C),

(B,B,B,B,A,C,C,C,C, C,C, C,C) etc.

Si le GOS proposé par le système ne figure pas dans la base des motifs possibles alors la compatibilité entre les graphes n'est pas valide. Dans ce cas, l'envoi et la récupération des données par le système doivent être re-étudié suivant le besoin de l'IP. GOS est employé pour définir l'ordre dans lequel le système envoie/reçoit les données. Selon cet ordre, le « driver » logiciel est généré (cf. section 5).

Une fois la compatibilité vérifiée, l'étape suivante du flot d'intégration est la configuration des modules génériques de l'architecture de l'interface. Pour cela une bibliothèque de modules est définie en utilisant des modules décrits par leurs modèles MEF. Nous détaillons, dans la section suivante, les éléments de cette bibliothèque permettant de spécifier la structure d'une architecture générique de l'interface de communication pour l'intégration des IPs accélérateurs. L'interaction entre les sous modules de l'architecture de l'interface et les modèles de graphes définis est analysée pour introduire l'étape de configuration dans le flot d'intégration.

4. Modèle Générique de l'architecture de l'interface matérielle

Le chemin de données de l'interface inclut des modules de stockage des données d'entrées/sorties de type FIFO. Ces FIFOs permettent de gérer la différence temporelle relative à l'utilisation des données entre l'IP d'une part et le reste du système d'autre part. Elles permettent en particulier de stocker temporairement les résultats produits par l'IP et qui ne sont pas encore exploités par le système.

L'interface de communication présente une « unité de transfert de données » formée par une unité de « tamponnage » instantanée (FIFO dans notre approche) et une « unité de contrôle » permettant le transfert et l'aiguillage des données (sérialisation ou parallélisation) (Cf. figure 27).

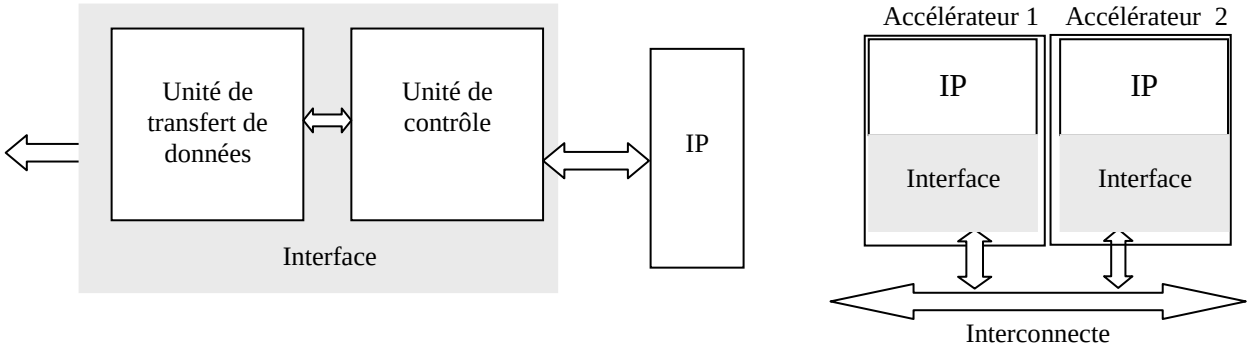

Figure 26. Interfaçage d'IP

L'interface matérielle est composée de deux blocs de base. Le premier bloc sert à contrôler les entrées du système vers l'IP (contrôleur_in). Le second sert à contrôler les sorties (contrôleur_out). Ils sont symétriques et fonctionnent indépendamment l'un de l'autre. L'unité de contrôle et l'unité de transfert sont modélisées par un ensemble de machines d'états communicantes qui sont décrites dans la sous section 4.1. La bibliothèque de communication est formée par ces machines d'états finis qui serviront à la construction de la structure de l'architecture de l'interface de communication. Dans la section 4.1, les détails techniques qui sont expliqués concernent uniquement la partie contrôle du côté des entrées puisque nous utilisons la même démarche de conception pour la partie de contrôle de sortie. Nous présentons par ailleurs deux versions de l'interface ; une pour le cas monoprocesseur et l'autre pour le cas multiprocesseur.

4.1. Modules de l'interface

L'interface est formée par un ensemble de modules synchrones et séquentiels. L'idée est de spécifier ces différents modules au niveau macro architecture ou RTL afin de faciliter leur synthèse et leur transposition sur l'architecture cible en utilisant des outils de synthèse/placement routage commerciaux. Ces modules RTL sont générés par une configuration à partir d'une bibliothèque de modèles génériques à l'aide des différents graphes d'ordonnancement. L'étape de configuration sera détaillée dans la section 6.

La spécification des éléments de cette bibliothèque est basée sur une modélisation formelle utilisant des machines à états finis (MEFs). Ce qui permet de représenter les interfaces indépendamment du langage de description.

Le modèle des MEFs [Har96] est très connu et bien maîtrisé. Ceci est dû au fait que les propriétés de terminaison, de séquencement des actions et de déterminisme sont bien formulées. Une MEF est un automate caractérisé par des états (dont l'un est l'état initial) et des liens entre les états qui représentent les transitions possibles. Pour qu'une transition soit effectuée, trois conditions doivent être vérifiées :

- l'automate doit être dans l'état de départ,
- il doit y avoir un front actif du signal d'horloge,
- les entrées de commande, autres que l'horloge, doivent autoriser la transition

L'utilisation de MEFs pour la modélisation permet d'assimiler le modèle général d'un système à une fonction à trois paramètres : le contrôle, la fonction, et les données. Intuitivement, il est possible de dire que les fonctions transforment l'information dans le système, que les données représentent les entrées et les sorties des fonctions et que le contrôle active les fonctions suivant la séquence voulue.

L'interface de communication est formée par quatre modules :

- un module FIFO
- un module contrôleur à l'entrée à l'entrée de l'IP (CTRL_IN dans la figure 28)
- un module FIFO_Enable, et
- un module Enable

La figure 28 illustre le schéma de la partie de l'interface responsable de l'ordonnancement des données à l'entrée de l'IP seulement. La partie responsable des sorties est symétrique à celle des entrées.

Le fichier de caractérisation d'un IP est un ensemble d'informations qui caractérisent au cycle près et au bit près le comportement spatio-temporel des données à l'interface de cet IP pour une itération de calcul.

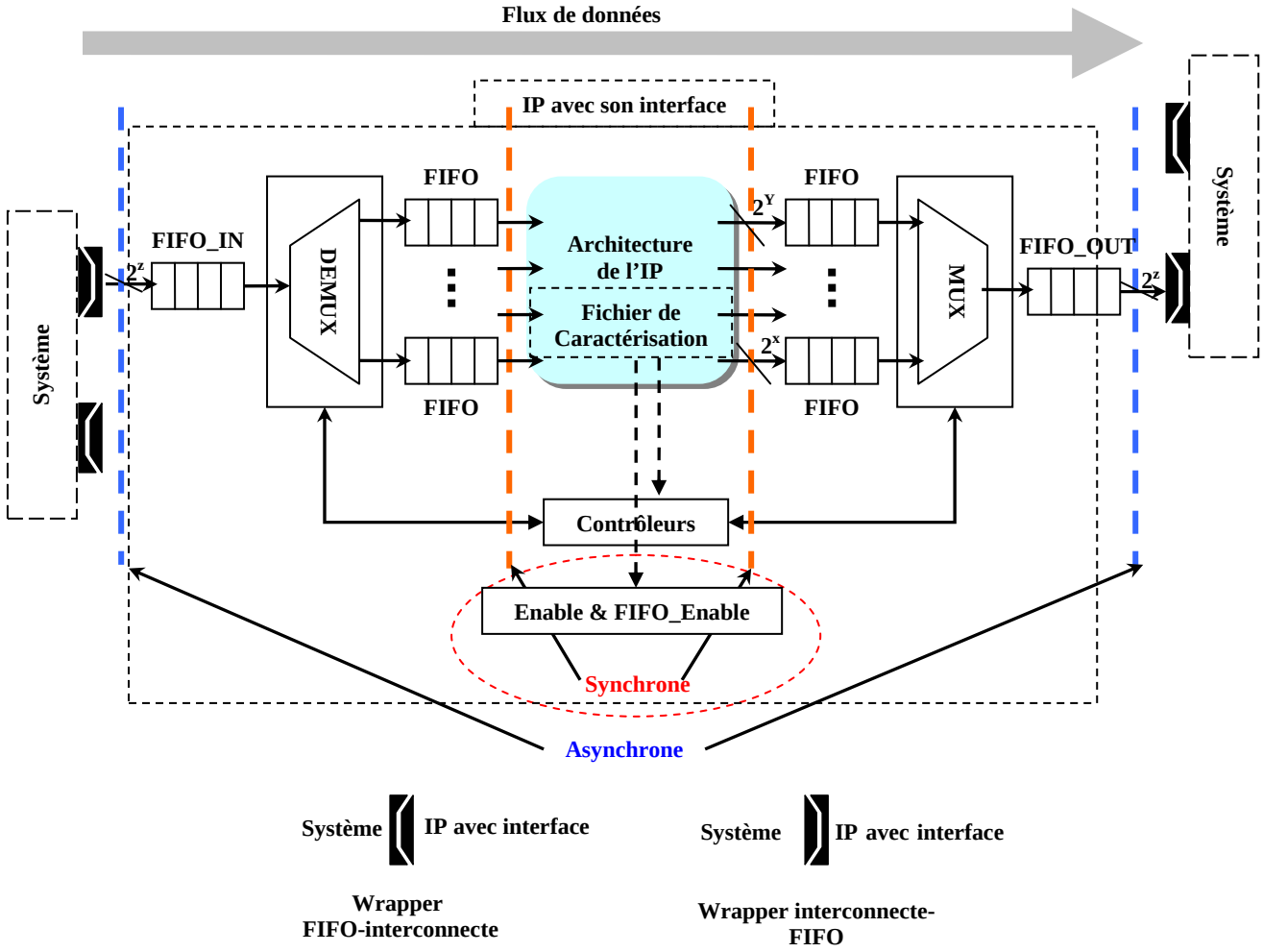

Figure 27. Conception du module de l'interface

Les MEFs correspondant à ces modules sont utilisées pour la spécification de leurs systèmes de contrôle. Elles permettent une description comportementale simple, en exprimant les états discrets de l'entité et les conditions de changement d'état.

Nous détaillons dans la suite les MEFs de chaque module de l'interface.

4.1.1. Module FIFO

Le protocole de base de la FIFO séquentielle est implémenté selon la machine d'états finis représentée par la figure 29.

Elle possède trois états. L'état « Full » indique que la FIFO est pleine et ne peut plus recevoir de données. L'état « Empty » indique que la FIFO ne peut pas fournir des données.

L'état « FIFO_DATA » indique que la FIFO peut recevoir ou délivrer des données selon l'état dans lequel se trouve la FIFO (lecture (R) ou écriture (W)).

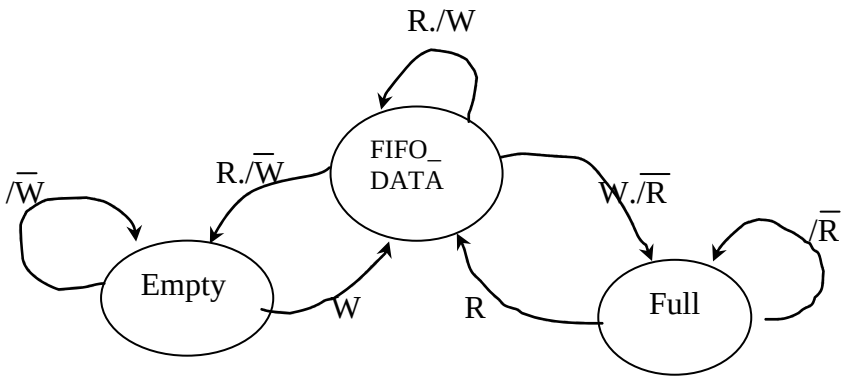

Figure 28. MEF de la FIFO

L'adaptateur entre le protocole de l'interconnecte et le protocole FIFO constitue la partie unité de transfert de l'interface matérielle. Cette adaptation entre ces deux protocoles est spécifique à l'architecture cible d'intégration.

4.1.2. Module Contrôleur à l'entrée de l'IP (CTRL_IN)

Le contrôleur de l'interface est l'automate qui permet d'acheminer les données à traiter aux ports d'entrées de l'IP. Il existe deux contrôleurs : un contrôleur à l'entrée et un contrôleur pour la sortie. Pour le contrôleur en entrée, il permet le démultiplexage des données à l'entrée de l'IP alors que pour la sortie, il s'agit du multiplexage de données résultantes du traitement de l'IP.

Le module CTRL_IN, piloté par le graphe GOESS, fonctionne suivant la machine d'états finis représentée par la figure 30.

Comment il est piloté, par le driver

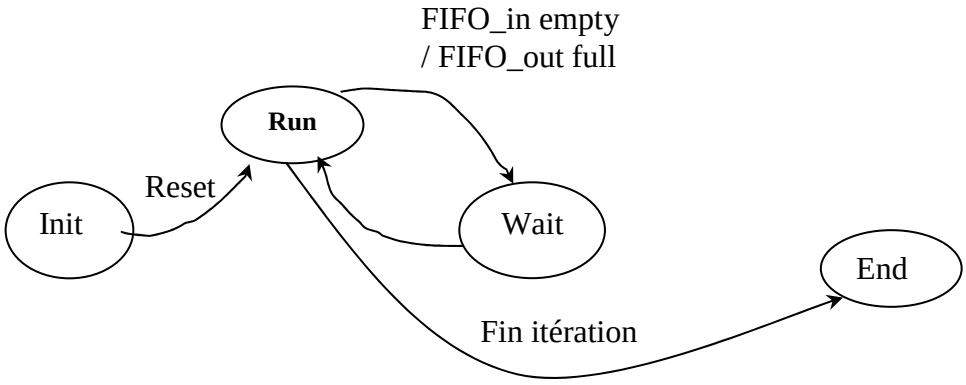

Figure 29. MEF du CTRL_IN

Cette machine d'états est formée par quatre états :

- L'état « Init » permet au contrôleur d'initialiser ses registres de configuration pour une nouvelle itération de calcul.
- L'état « Wait » permet au contrôleur d'attendre les données issues des FIFOs.
- L'état « Run » permet de dispatcher les données avec la bonne taille des structures dans les ports d'entrées correspondants de l'IP.
- L'état « End » permet au contrôleur de retourner à état de reset.

4.1.3. Module FIFO_Enable

Une fois que les données sont rangées dans le bon ordre dans les ports d'entrées correspondants, le module « FIFO_Enable » se charge de fournir les données correspondantes aux cycles correspondants pour l'IP tout en synchronisant l'IP et les FIFOs en amont des ports de l'IP. Le module « FIFO_Enable » est schématisé par le diagramme bloc de la MEF représenté par la figure 30.

Figure 30. Diagramme bloc de la MEF du module FIFO_Enable

Les « R [i] » sont les signaux qui indiquent qu'une lecture est possible à partir de la FIFO numéro « i » en amont du port « i » ; le signal « Enable » permet d'éveiller la MEF du module Enable. Ce module est un simple test « ET logique » entre les différents R[i] correspondant à un point de synchronisation permettant l'autoriser le fonctionnement de l'IP. Le module « FIFO_Enable » peut connecter ou déconnecter les « FIFO » (FIFO en amont des ports d'entrée de l'IP) au module de l'IP selon leur utilisation dans les points de synchronisation liée au trafic des données. Une fois la connexion réalisée, le module « Enable » se charge de fournir la donnée correspondante au cycle correspondant pour l'IP.

4.1.4. Module « Enable »

Le module « Enable » est piloté par le graphe GOES et fonctionne selon la MEF représentée par la figure 32. Cette MEF est constitué de trois états. L'état « INIT » permet d'initialiser le module pour commencer le traitement des données. L'état « Wait »permet au module d'attendre l'arrivée des données adéquates des FIFOs ou du signal à partir du module « FIFO_Enable ». L'état « Calcul » caractérise le module pendant le traitement des données. Il fonctionne selon les paramètres de configuration génériques suivants :

- le nombre de ports pour les entrées/sorties.

- la taille des entrées/sorties.
- le nombre de cycles pour une itération : ce paramètre est utilisé pour le fonctionnement du compteur virtuel correspondant à l'horloge de synchronisation virtuelle de l'IP pilotant le module « Enable ».

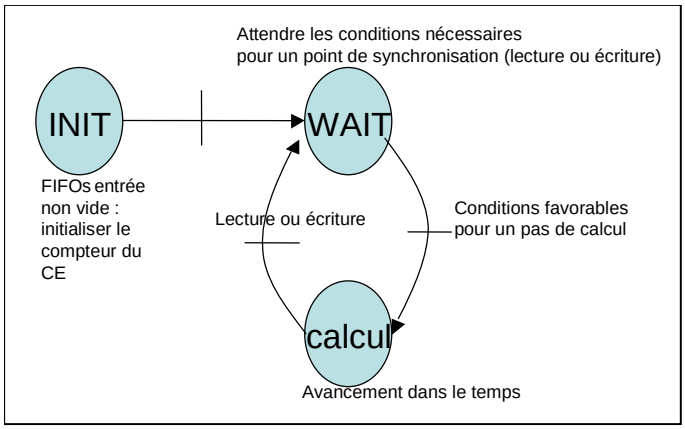

Figure 31. Automate Enable

La MEF du module « Enable » est cadencée par une horloge virtuelle (compteur CE) pour contrôler l'exécution temporelle à l'interface de l'IP. « CE » est remis à zéro pour une nouvelle itération de calcul et peut geler le comportement de l'IP si les données ne sont pas encore prêtes (par exemple, si le système n'a pas encore fourni les données nécessaires pour le traitement des données).

Les modules précédemment décrits sont utilisés pour construire la chaîne en entrée de la structure de l'architecture de l'interface de communication d'un IP à intégrer. Il en est de même pour la chaîne de communication en sortie de l'IP.

4.2. Architecture de l'interface de communication

Nous envisageons d'étudier deux conceptions architecturales de l'interface de communication : une pour le cas mono processeur et l'autre pour le cas multiprocesseur en ne tenant compte que de l'ensemble des informations qui caractérisent au cycle près et au bit près le comportement spatio-temporel des données à l'interface de l'IP pour une itération de calcul.

4.2.1. Première conception (dédiée pour le cas monoprocesseur)

Cette première conception est dédiée au cas d'un système mono processeur. Dans ce type de système, le transfert des données entre l'IP et le reste du système suit un ordre imposé par l'IP. Cet ordre est déduit à partir de son fichier de caractérisation. Il est appelé motif ou « pattern » de cette commuication. Ce motif exprime le modèle de séquencement des données

pour une itération de calcul. Il est sauvegardé selon la forme de spécification du « driver » logiciel (routines read() et write()). Le système intégrant doit respecter cet ordre. Dans le cas de l'application « produit de deux matrices A et B dont le résultat est C», un motif possible est : « matrice A, matrice B, matrice C ». Le contrôleur d'entrée de l'interface dissocie suivant cet ordre les données vers les bons ports et selon l'ordre exigé (motif). Dans ce cas, cet ordre est le même que celui de l'envoi des données par le système.

4.2.2. Deuxième conception (dédiée pour le cas multiprocesseur)

Cette deuxième conception est dédiée au cas d'un système multiprocesseur. Les données sont issus de deux ou de plusieurs processeurs (appelés initiateurs), chacun envoie des données relatives à une structure particulière de données sur laquelle l'IP effectue le traitement. Par exemple, pour le cas d'un produit de deux matrices A et B géré par deux processeurs distincts, l'initiateur n°1 envoie les éléments de la matrice A, l'initiateur n°2 envoie ceux de la matrice B. Pour pouvoir différencer ces données et déterminer le port associé, chaque donnée est envoyée avec l'adresse de l'inititiateur. Les données en sortie de l'IP doivent être également fournies avec l'adresse de l'initiateur correspondant. Le transfert des données est réalisé en décodant le champ « adresse » de la donnée à l'entrée de la FIFO. L'adresse considérée est celle de l'initiateur de la communication. Elle ne correspond pas à l'emplacement physique en mémoire des données. Dans ce cas, l'échange des données peut être spécifié selon différents motifs respectant uniquement l'ordre exigé par le port de l'IP. Un motif d'ordre possible est appelé dans ce cas « pseudo ordre ». Le système reconnaît l'information qui concerne la succession des données dans chaque port mais il ignore l'ordre et le temps d'envoi des coefficients d'une structure en entrée ou en sortie à l'interface de l'IP.

4.2.3. Comparaison des deux conceptions

Les deux conceptions peuvent s'interfacer avec différents protocoles de communication, ce qui permet une indépendance par rapport au protocole de communication de l'interconnecte.

Dans la deuxième conception, l'ordonnancement des données utilise l'adresse de l'initiateur ce qui permet plus de flexibilité par rapport à la première conception en ciblant le contexte d'intégration dans un SoC multiprocesseur. Cette conception a aussi l'avantage de générer facilement le pilote logiciel vu la flexibilité de l'échange des données que cette conception permet de faire. En revanche, elle a l'inconvénient d'être non optimale en terme d'unités de mémorisation puisque la cellule FIFO_IN (resp. FIFO_OUT) en entrée (resp. en

sortie) contient la donnée et l'adresse du composant maître transmettant la donnée. Le contrôleur de l'interface est plus complexe que celui de la première conception puisqu'il doit démultiplexer les données selon les adresses de l'initiateur.

D'autre part, la première conception a plusieurs avantages. C'est une solution optimale qui nécessite moins de cellules FIFOs que la première conception. Cependant, son « driver » est difficile à générer : il n'y a pas de flexibilité dans sa génération puisque le système doit générer une séquence dans un ordre fixe bien défini. Elle ne permet pas aussi de cibler le contexte multiprocesseur sans arbitre d'ordonnancement [Zit01].

Dans cette section, nous avons présenté deux conceptions différentes de l'interface de communication : une dédiée pour le cas monoprocesseur et l'autre pour le cas multiprocesseur. Le principe de fonctionnement de l'interface est le même dans les deux cas. Elle contient une unité de contrôle et une unité de transfert des données pour la mémorisation intermédiaire dans des FIFOs.

Comme diverses IPs connectés sur un même système peuvent consommer des données de largeurs différentes (8, 16 ou 32 bits etc.), et comme le système peut imposer une taille maximale des données qui transitent à travers son interconnecte, la gestion des cellules FIFOs devient indispensable pour assurer une communication efficace. Nous détaillons dans la section 4.3, l'influence de la gestion des cellules des files de données en entrées et en sorties de l'interface (FIFO_IN et FIFO_OUT) sur le fonctionnement des MEFs du FIFO_IN, FIFO_OUT, CTRL_IN et CTRL_OUT.

4.3. Contrôle de la gestion des cellules FIFO_IN

Les cellules FIFO_IN (resp. FIFO_OUT) doivent être utilisées en intégralité quelle que soit la taille de la donnée à sauvegarder. Par exemple si nous n'adoptons pas la gestion des FIFOs, une seule FIFO de largeur 32 bits peut contenir une seule donnée de 8 bits, le reste de la cellule reste inexploité. Par ailleurs, un interconnecte véhiculant des données de 64 bits ne peut pas communiquer avec le protocole FIFO de l'interface de communication si une cellule FIFO_IN (resp. FIFO_OUT) est définie de largeur 32 bits.

La gestion des cellules de mémorisation revient à ajouter de l'intelligence dans la chaîne de communication d'entrée/sortie de l'interface que ce soit pour la première conception ou pour la deuxième conception de l'interface de communication. Cette intelligence implique une complexité de la partie contrôle des MEFs des modules constituant l'interface de communication. Cette complexité se traduit par l'ajout de fonctionnalités

pouvant générer deux versions du même module. Particulièrement, cette gestion touche la partie contrôle du module FIFO_IN (respectivement FIFO_OUT) et également du module CTRL_IN (respectivement CTRL_OUT).

Pour des raisons de symétrie, nous allons détailler le fonctionnement de l'interface en entrée de l'IP (FIFO_IN et CTRL_IN).

4.3.1. Fonctionnement du Module FIFO

Nous envisageons de détailler dans cette section la fonctionnalité permettant de gérer le stockage des données dans la FIFO_IN en entrée à l'interface. Cette fonctionnalité doit prendre en considération la taille des données en entrée et les répartir en conséquence dans les cellules des FIFO_IN. Par exemple, si la taille de la donnée est plus grande que celle des cellules FIFO_IN, ce module doit répartir la donnée sur deux ou plusieurs cellules FIFOs.

Deux cas se présentent pour chaque conception de l'interface en considérant la taille des données véhiculées par l'interconnecte :

- la taille de la donnée système (tds) est supérieure à la taille de la donnée de l'IP accélérateur (tda).
- tds est inférieure à tda.

La gestion de l'empilement des données au sein d'une cellule FIFO_IN est traitée de la même manière pour les deux conceptions de l'interface de communication. La seule différence est la considération de l'adresse du composant initiateur pour la deuxième conception. Comme nous l'avons précisé, deux cas se présentent :

i) Cas où tds <tda

Afin d'optimiser l'utilisation des FIFO_IN, son contrôleur doit inclure :

- Un masque pour chaque structure de données : il est utilisé pour distinguer la donnée utile dans une requête véhiculée par l'interconnecte qui contient plusieurs autres informations. Ce masque permet de sauvegarder uniquement les données utiles pour l'IP.
- Un mécanisme interne dans la MEF de la FIFO_IN qui permet de concaténer dans une seule cellule les données qui se suivent (du bit poids fort au bit de poids faible). Dans le cas de la deuxième conception (dédié au cas multiprocesseur), les cellules concaténées doivent avoir la même adresse de l'initiateur comme le montre la figure 33.

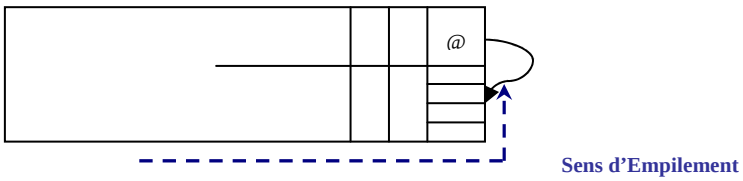

Figure 32. Sens d'empilement dans la FIFO_IN (tds>tda)

– L'adresse dans une cellule FIFO_IN correspond à l'adresse de la première donnée dans une cellule FIFO_IN. Le décodage de l'adresse pour une cellule FIFO_IN se fait une seule fois pour l'ensemble des données concaténées dans cette cellule. Nous supposons que les données dans une cellule FIFO appartiennent à une même structure de données. Dans le cas de la figure 32, l'empilement correspond à envoyer 4 coefficients (dans le bon ordre) appartenant à une même structure (même adresse).

ii) Cas où tds >tda

La partie contrôle de la MEF du module FIFO_IN doit permettre :

– L'empilement des données selon chaque argument «taille des données » pour chaque structure. Par exemple, si l'initiateur envoie une donnée sur 64 bit et que la FIFO_IN possède des cases de 32 bits, le contrôleur doit diviser cette donnée sur deux cellules FIFO_IN avec une même adresse d'initiateur comme le montre la figure 34.

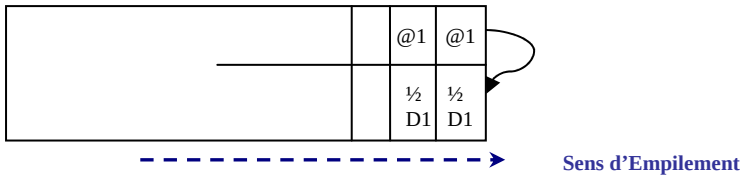

Figure 33. Sens d'empilement dans la FIFO_IN

– Pour gérer des données de taille plus que 32 bits chacune, la FIFO_IN permet d'empiler les données successivement.

4.3.3. Fonctionnement du Module contrôleur

L'ajout de la fonctionnalité « gestion intelligente des cellules FIFO» nécessite des modifications dans la MEF du module CTRL_IN. Nous supposons que l'interconnecte du système véhicule une seule adresse pour une structure d'entrée correspondant à l'élément initiateur d'envoi dans la composition du SoC. Cette hypothèse fait que le CTRL_IN décode les adresses et différencie les données selon «l'adresse de début de la structure».

Nous prenons par exemple l'accélérateur «produit de deux matrices ». Nous supposons que le transfert se fait à travers une RAM. Nous supposons que l'application est partitionnée sur une architecture de façon que (Cf. figure 35) :

– un processeur génère la matrice A,
– un deuxième processeur génère la matrice B
– et un troisième utilise le résultat de la multiplication

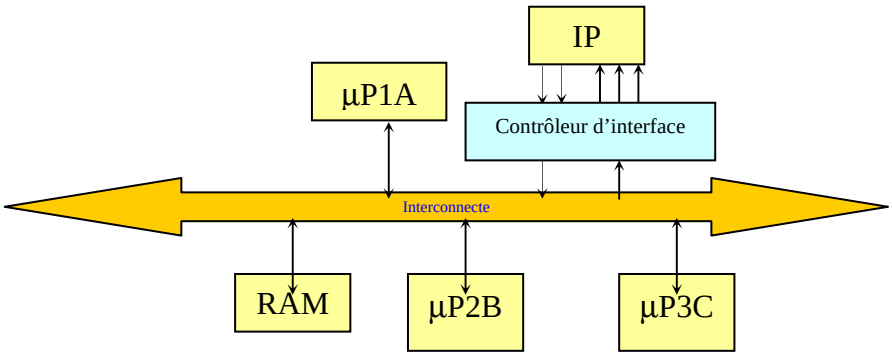

Figure 34. Architecture cible proposée

Si nous travaillons avec une mémoire partagée. Cette mémoire présente un segment noyau contenant les instructions de « boot » de la plateforme et un segment utilisateur pour le traitement des données. Nous supposons que les données sont successivement rangées dans la mémoire dans un segment utilisateur (Cf. figure 36).

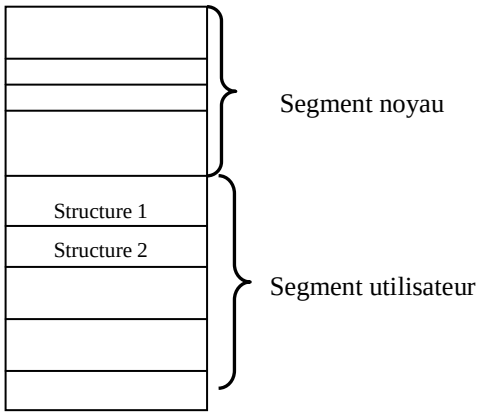

Figure 35. Rangement des données dans la mémoire RAM

Pour cet exemple, la dissociation des données selon les structures de données se fait selon la conception considérée de l'interface.

Pour la deuxième conception (avec adresse), le module CTRL_IN dissocie les données selon les adresses initiatrices suivant le cas de la figure ci-dessous (Cf. figure 37). Nous supposons que le champ « donnée » et le champ « adresse » sont sur 32 bits chacun et que les données à traiter sont sur 16 bits.

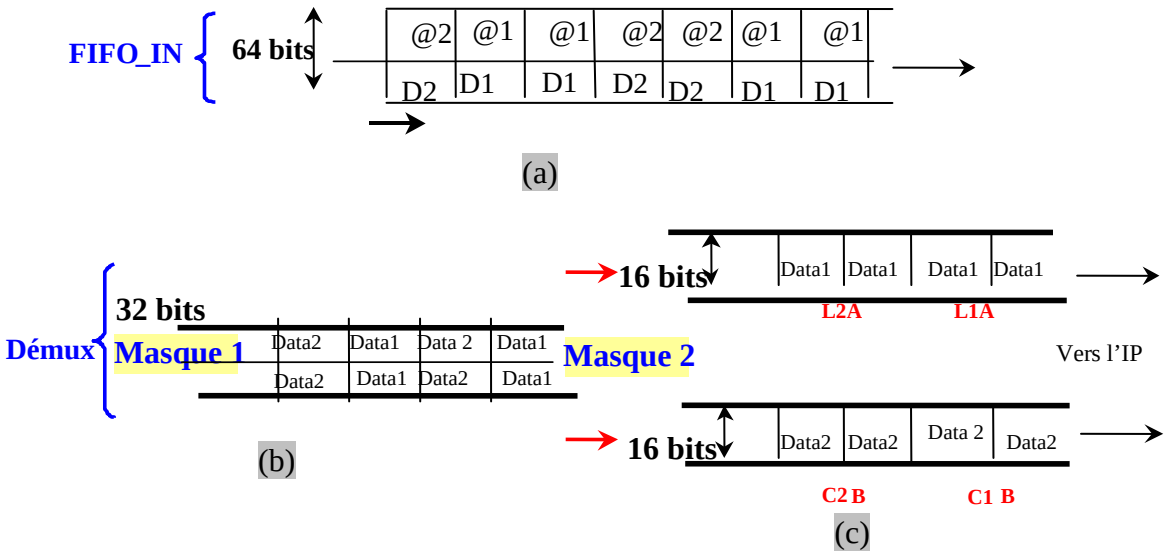

Figure 36. Fonctionnement du module CTRL_IN (conception 2)

La première étape permet la dissociation des données selon le champ adresse avec un masque de 32 bits (figure 37 (a)) et un décodeur d'adresse interne différenciant les données selon les structures. Le rangement successif dans une cellule FIFO_IN se fait à condition d'avoir la même source (adresse de l'initiateur) des données consécutives.

Nous considérons :

- Data 1 (D1) correspond aux données de la structure 1 (matrice A dans notre exemple)
- Data 2 (D2) correspond aux données de la structure 2 (matrice B dans notre exemple)
- L1A désigne la première ligne de la matrice A
- L2A désigne la deuxième ligne de la matrice A
- C1B désigne la première colonne de la matrice B
- C2B désigne la deuxième colonne de la matrice B.

Le masque 1 (figure 37(b)) permet de dissocier les données selon la structure. La deuxième étape permet de dissocier les données consécutives dans une structure selon la bonne taille et le bon ordre à l'aide du masque 2 (figure 37(c)).

La partie donnée dans le module « CTRL_IN » (de la première conception et de la deuxième conception) est traitée avec l'algorithme traitant les deux cas suivants :

i) Cas où tds >tda

- La partie donnée est fragmentée selon la taille du masque.

- La taille du masque prend la taille de la donnée à récupérer : nous dissocions les masques selon les structures.
- Selon l'ordre de succession (pour chaque structure de donnée) de la réception à partir de la FIFO_IN, le contrôleur peut dispatcher les données vers les ports correspondants.

ii) Cas où tds <tda

- Le masque sur les données dans la cellule FIFO_IN ne permet pas de récupérer les coefficients des données. Si nous supposons que l'IP traite des données de taille appartenant successivement à un même paquet de donnée fragmenté dans les cellules FIFOs, elles sont alors concaténées selon la taille réelle des données à traiter par le CTRL_IN.

L'interface ainsi décrite est générique, paramétrable et configurable, ce qui lui permet de se connecter avec différents types d'interconnectes et différents types d'IPs accélérateurs. En effet, l'interface est capable d'assurer le transfert de données entre un interconnecte ayant une taille de données « m » et un accélérateur ayant une taille de données « n » quelque soit n et m. Les tailles de données sont ainsi configurables.

Avec la définition des modules constituants l'interface de communication, nous pouvons réaliser la configuration des MEFs. L'étape de génération d'interface de communication (architecture matérielle et « driver » logiciel) respecte ainsi une méthodologie d'intégration opérant sur l'ensemble des hypothèses.

Nous détaillons, dans la suite, la structure du pilote logiciel qui permet de gérer cette interface.

5. Interface logicielle : Pilote de l'interface

Comme toute interface de communication, nous associons un pilote logiciel appelé aussi « driver » à l'interface matérielle [Cyr04]. Il est exécuté par le processeur qui commande l'accélérateur matériel (l'IP) via l'interface matérielle. Cette partie logicielle de l'interface doit commander la communication des données entre l'IP et la mémoire du système [Cyr04].

Pour la réutilisation de l'interface, seule la partie logicielle est modifiée ; quant au contrôle matériel, il reste le même vu le caractère générique de la conception. En effet, le but est de fournir une interface générique paramétrable qui peut être re-appliquée pour d'autres IPs orientés flot de données sans connaître le comportement interne de l'IP.

La partie logicielle exécutée par le/ les microprocesseur/s se compose de :

- une représentation textuelle de l'activité des entrées et des sorties de l'IP.
- un fichier pilote «driver » pour le contrôle de l'IP. Il spécifie l'ordre d'envoi et de réception des données par le système.

Le pseudo ordre est l'ordre selon lequel le « driver » fonctionne. Ce pseudo ordre spécifie l'ordre à l'interface de l'IP. Ce pseudo ordre est traduit par le graphe GOS pour son abstraction.

Nous considérons dans la suite un exemple de génération de pilote pour un module matériel simple constitué par trois registres qui retardent les données en entrées de 3 cycles d'horloge. Deux types d'instructions sont utilisés dans le développement de la partie logicielle :

- la commande (read ()) pour l'envoi des données de la mémoire vers l'accélérateur
- la commande (write()) pour la récupération des données traités et l'écrire dans la mémoire.

Après un Reset, les N données de «a » sont envoyées, une donnée par cycle d'horloge et les N résultats de b commencent à être générées après 3 cycles d'horloges. Ces cycles d'horloges correspondent aux cycles d'horloges virtuels : ce sont des cycles introduits pour pouvoir modéliser l'ordonnancement des données et ne correspondent pas forcément à une horloge réelle. A cette description, l'ordonnancement des entrées et des sorties de l'IP peut s'exprimer par l'utilisation de boucles. Ces boucles décrivent le comportement du « driver » :

For t=1 to N+3, où t correspond au compteur d'incrémentation temporelle
If (1<=t<=N) read(a)
If (4<=t<=N+3) write(b),

Nous supposons maintenant qu'à chaque port, par exemple «a », les données successives correspondent aux éléments successifs d'un vecteur linéaire. Ce vecteur est virtuel dans le sens où il ne correspond pas nécessairement à un « layout » physique de la donnée dans la mémoire du SoC intégrant l'IP. En fait, C'est une représentation qui permet d'identifier chaque donnée en entrée à un vecteur. Pour cet exemple, les N entrées au port «a » correspondent à un vecteur a[N] indexé de 1 à N. Nous aboutissons à une deuxième représentation algorithmique du pilote qui tient compte de ces nouvelles conditions :

```
For t=1 to N+3
        If (1<=t<=N) read(a[t-1])
        If (4<=t<=N+3) write(b[t-4])
```

Le comportement est observé de l'intérieur du module de l'IP, donc les cycles d'horloges correspondent aux cycles d'horloge virtuels. En effet, si nous intégrons cet IP dans un système, et si pour une raison la donnée «a » n'est pas prête dans un cycle d'horloge, alors l'horloge réelle continue à s'incrémenter alors que le compteur de l'horloge virtuelle reste constant tant que l'horloge virtuelle n'est pas réveillée. Pour ces représentations, nous considérons que le compteur d'horloge (t) correspond à celui de l'horloge virtuelle.

Dans les deux cas précédents, aucune précision n'est donnée sur les instants d'échange des données. Si nous introduisons la notion de cycles d'horloges et en faisant abstraction de l'existence physique du vecteur «a », nous pouvons aboutir à la boucle d'implémentation suivante :

```
For coef = 1 to 3
        read(a)
        wait(1)
For coef = 4 to N
        read(a)
        write(b)
        wait(1)
For coef = N to N + 3
        write(b)
        wait(1)
```

Cette implémentation correspond à N échantillons d'entrée «a » décalés de 3 cycles d'horloges des N échantillons de la sortie «b ». Si l'IP a plus d'une entrée, ses entrées sont multiplexées dans l'interconnecte et démultiplexées dans les contrôleurs de l'interface. Jusqu'à maintenant, l'index de l'échantillon est analogue au numéro de l'échantillon traité. Il est égal dans cet exemple particulier au compteur de l'horloge virtuelle (car chaque échantillon est traité dans un cycle virtuel). L'exécution d'une instruction dure 0 cycle du temps d'exécution (la lecture et l'écriture se font en parallèle). Le « wait(1) » correspond à la sémantique des routines « wait » dans VHDL : chaque exécution du cœur d'une boucle consomme au moins un cycle d'horloge. Dans ce cas, les « wait » indiquent combien de cycles d'horloge virtuels consomment les échantillons.

Avec un ordonnancement plus exigeant qui traite des données plus variées, le comportement des données à l'interface d'un nouveau composant matériel peut nécessiter un traitement plus compliqué sur chaque échantillon. Par exemple, le comportement de l'IP

nécessite trois cycles d'horloges entre deux échantillons consécutifs pour exécuter ses calculs. Après un reset, les N entrées sont envoyées successivement, une par cycle d'horloge virtuel et après trois cycles d'horloges, la première sortie est produite suivis par les N-1 autres sorties.

Les boucles utilisées pour exprimer ce comportement sont les suivantes :

```
For coef = 1 to 1
        read(a)
        wait(3)
For coef = 2 to N
        read(a)
        write(b)
        wait(3)
For coef = N to N + 3
        write(b)
        wait(3)
```

Ces boucles sont exactement l'information dont la partie contrôle dans l'interface a besoin pour véhiculer correctement les entrées et les sorties de l'IP à l'interconnecte du système intégrant. En revanche, l'adresse des données est absente dans la boucle. Cette adresse dépend des paramètres extérieurs à l'IP (la cartographie mémoire ou « data layout »).

Associé à l'information de la nature de l'architecture du SoC/MPSoC cible, le code du « driver » est séparé en un ou plusieurs « drivers ». En effet, nous supposons que la répartition des taches entre initiateurs maîtres dans une architecture multiprocesseurs pour l'utilisation d'un accélérateur matériel se fait suivant le nombre des flux de données. Les « drivers » sont séparés suivant les entrées et les sorties. Un initiateur maître peut envoyer un flux d'entrée ou bien récupérer un flux de sortie comme résultat de traitement ; ce qui est souvent le cas. Pour un IP à deux entrées et une sortie, le « driver » initial peut être divisé au maximum en trois « drivers » : un « driver » pour la première entrée, un « driver » pour la deuxième entrée et un troisième « driver » pour la sortie.

En conclusion, le principe retenu est que les données rentrant / sortant successivement sur un port sont rangées successivement en mémoire. Ceci autorise d'utiliser le mode «rafale» pour transmettre les données. Si cette condition n'est plus respectée et que nous voulons tout de même faire du « rafale», il faut envisager un mécanisme de tamponnage plus complexe qu'une FIFO du côté de l'IP [Ian02]. En effet, si l'ordre d'envoi et l'ordre de consommation sont différents lors d'une communication de type point à point entre le producteur de données et son consommateur, une FIFO simple dans ce cas ne suffit pas pour une transformation

d'une structure de donnée à N dimension (par exemple une matrice est une structure à deux dimensions) en une structure de donnée à une dimension (un vecteur de donnée). Il faut prévoir dans ce cas un mécanisme de tamponnage plus complexe comme celui présenté dans [Ian02].

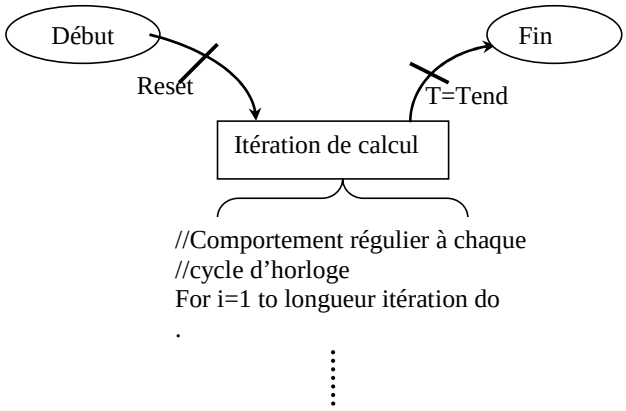

Figure 37. Comportement du « driver »

Outre la description textuelle de l'échange de données aux entrées sorties de l'IP, le pilote contient aussi le programme ou le « script » de l'interface logicielle qui permet d'initialiser les registres de configurations du contrôleur de l'interface matérielle. Les registres de configurations contiennent les informations nécessaires pour l'itération en cours d'exécution (figure 24). Nous supposons que les registres sont suffisamment larges pour supporter les informations de toutes les communications.

6. Etape de génération de l'interface

La génération de l'interface de communication se fait par un processus qui permet de sélectionner et de configurer la description des sous modules de l'interface de communication à partir d'une bibliothèque de composants selon le besoin applicatif en suivant un certain nombre d'étapes (cf. figure 38) :

- Etape de sélection : elle consiste à sélectionner les modules nécessaires pour construire le schéma de l'interface selon le choix applicatif et le contexte d'intégration (architecture mono ou multiprocesseur, simulation ou synthèse, motif ou pseudo ordre etc.)
- Etape de configuration : elle consiste à configurer et à paramétrer les modules concernés, à affecter des valeurs aux variables génériques (taille des FIFOs, bandwidth, nombre de ports, tailles des données etc.)
- Etape de génération d'interface de communication :

- Etape de génération de l'architecture de l'interface : elle consiste à assembler les modules sélectionnés en un schéma d'interface de communication configuré interagissant avec les graphes de données pour assurer le bon déroulement de la véhiculation des données et de synchronisation du transfert.
- Etape de Génération du Driver : Le « driver » doit être généré automatiquement à partir de la spécification des entrées sorties à l'interface de l'IP parallèlement avec la génération de l'interface matérielle.

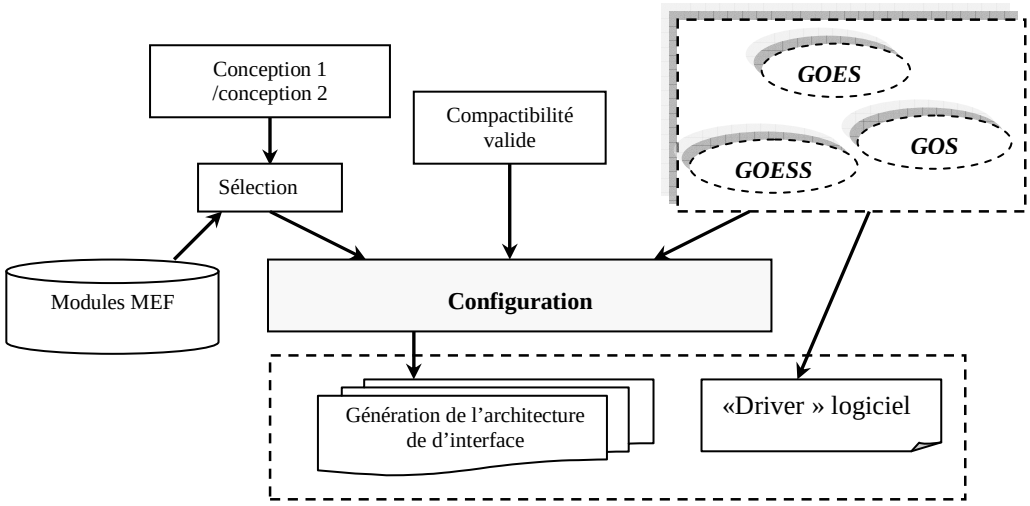

Figure 38. Étape de génération d'architectures d'interface

L'architecture de l'interface de communication est modélisée par l'interconnexion des machines d'états finis décrites dans la section 4.1. (Cf. figure 39). Elle respecte la synchronisation entre les sous modules communicants, le séquencement des données et les contraintes sur l'information spatio-temporelle du transfert des données. D'autres informations sont également utiles pour l'intégration de l'IP. En particulier, la taille des données véhiculées par l'interconnecte (tds), la taille d'une cellule FIFO et la profondeur de chaque FIFO. Il y'a autant de FIFOs que de ports d'entrées et de sorties. Le module CTRL_IN dispatche la donnée selon sa taille réelle et sa structure correspondante. L'ordre dans lequel les données sont transférées est très important pour piloter l'architecture d'une manière efficace. L'information de cet ordre est enregistrée dans le contrôleur d'entrée et celui de sortie sous la forme d'un ensemble de registres de configurations. A partir des modèles de graphes, ces registres sont initialisés avant le fonctionnement de l'interface.

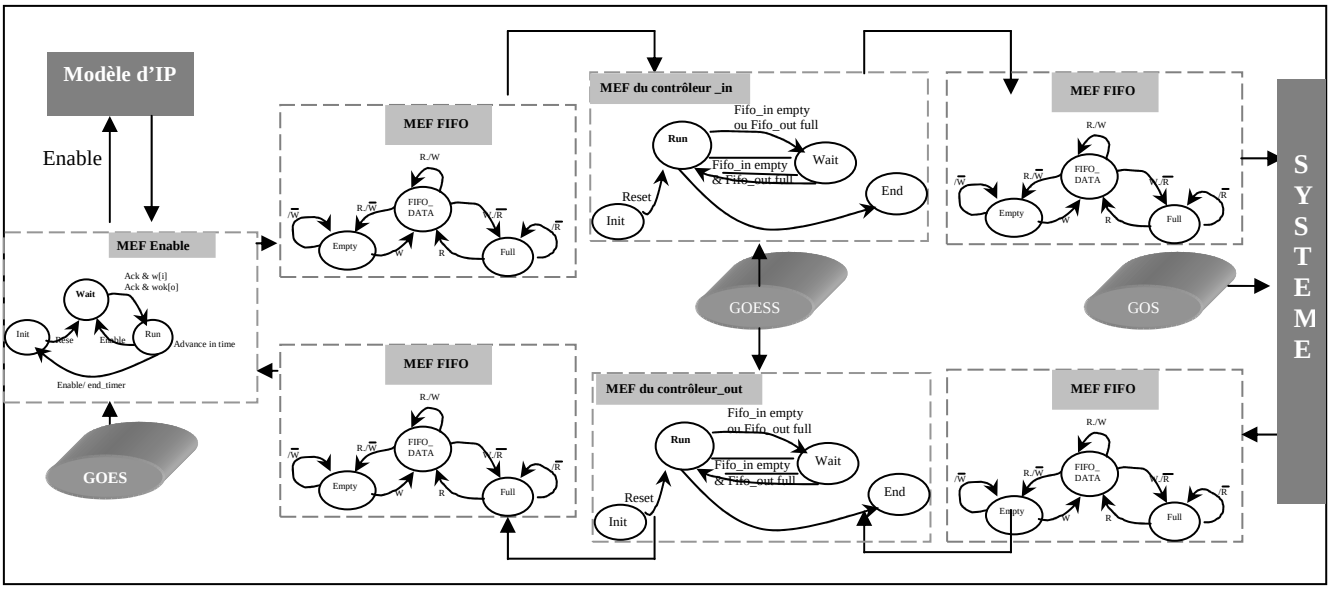

Figure 39. Interaction entre les sous modules de l'interface et les modèles de graphes

L'interaction entre les modules de l'architecture de l'interface et la spécification des graphes est illustrée dans la figure 39. L'interface logicielle ou le « driver » logiciel s'exécute par le processeur qui pilote l'IP via l'interface matérielle. Pour une réutilisation de l'interface, seule la partie logicielle est régénérée quant au contrôle matériel il reste le même. Le graphe « GOS » définit l'ordre selon lequel ce « driver » manipule les données. L'ordre des données dans ce graphe respecte l'ordre par structure (GOESS). Le GOES sert à déterminer l'ordre d'aiguillage des données à l'interface de l'IP.

Ainsi nous passons d'une approche de génération d'interface de communication à une configuration de l'interface de communication. Ceci a l'avantage de permettre au concepteur d'assurer la communication sans qu'il doive avoir à chaque fois une connaissance profonde des protocoles de communication et du séquencement des tâches. La génération automatique de l'interface de communication respecte une méthodologie d'intégration opérant sur l'ensemble des hypothèses favorisant son application. Les modèles de graphes assurent l'automatisation de l'adéquation architecture d'interface de communication et composant matériel donné à intégrer. Cette méthodologie repose sur une architecture générique de l'interface définie par une librairie de modules MEF paramétrables/ configurables.

L'automatisation des étapes de ce flot ainsi que la génération automatique de l'interface est confiée à un outil informatique. L'implémentation et l'expérimentation de cet outil sont illustrées dans le chapitre suivant.

7. Conclusion

Le domaine d'application que ciblent les travaux de cette thèse est celui des systèmes dominés par les données. Dans ce type de systèmes, les applications supportées sont de plus en plus complexes et le temps de mise sur le marché est de plus en plus court. Pour satisfaire ces contraintes de plus en plus pressantes, une solution prometteuse consiste à intégrer des IPs dans le système à concevoir. Pour pouvoir utiliser cette solution, deux conditions sont nécessaires :

- Avoir des structures de communication adaptées aux exigences des applications cibles capables d'interconnecter des IPs de provenance diverses.
- Avoir des outils permettant l'automatisation de l'intégration des IPs à travers ces structures de communication.

Nous avons présenté dans ce chapitre une approche permettant la réutilisation systématique d'IPs existants dans un environnement spécifique. Elle s'appuie sur un modèle d'interface générique défini selon deux conceptions une pour le cas multiprocesseur et l'autre pour le cas monoprocesseur. Les modules de cette interface sont décrits sous forme de machines à états finis.

Une modélisation en graphes est considérée pour une spécification abstraite de l'interface physique à adapter. Les modèles de graphes ainsi que la structure générique de l'interface favorisent l'automatisation du flot d'intégration appliqué pour une architecture cible dans un contexte de simulation et de synthèse. L'interface peut être générée si la compatibilité entre le transfert des données à l'interface de l'IP et à l'interface du reste du système est vérifiée.

Le chapitre suivant présente l'expérimentation de l'approche à travers la conception et la réalisation d'un outil de CAO « générateur d'interfaces de communication » automatisant les étapes de l'approche d'intégration. Les fonctionnalités de l'outil sont illustrées pour la génération de l'interface de communication pour des IPs accélérateurs dans un contexte de simulation et de synthèse permettant ainsi la validation de l'approche. Deux environnements ont été mis en œuvre : SoCLiB pour la simulation et l'environnement « Quartus » de la société Altéra pour la synthèse. Le domaine d'application est le secteur multimédia où l'application « synthèse 3D » est considérée.

CHAPITRE 4. EXPERIMENTATION DE L'APPROCHE D'INTEGRATION : OUTIL GIC

1. Introduction

Aujourd'hui, face à la complexité croissante des applications à concevoir, aux contraintes de coût et de temps de développement de plus en plus sévères, les concepteurs utilisent de plus en plus d'outils automatiques ou semi-automatiques efficaces et basés sur une ou plusieurs méthodologies de développement. Le développement de ces supports informatiques nécessite la maîtrise de plusieurs technologies (base de données, communication inter-outils, interface utilisateur, etc.). Le but de ces méthodologies est d'accélérer les différentes phases de conception des systèmes basés de plus en plus sur la réutilisation des IPs.

Afin de répondre à ces contraintes, un flot pour l'intégration d'IPs a été développé dans cette thèse. Il permet de vérifier dans un premier temps la compatibilité de l'IP et du reste du système puis dans un deuxième temps de générer l'interface et son pilote. Ce chapitre est consacré à la présentation de l'environnement d'expérimentation supportant ce flot.

Dans la section 2, nous détaillons le contexte d'application de la méthodologie proposée. Nous y présentons également l'outil de synthèse architecturale « GAUT » utilisé pour la génération des IPs à intégrer. Dans la section 3, nous présentons l'outil informatique «Générateur d'Interface de Communication : GIC ». L'outil de CAO GIC permet de générer l'interface de communication à partir d'une modélisation en graphes de l'ordonnancement du transfert des données à l'interface de l'IP à intégrer et du système cible.

2. Contexte d'application de la méthodologie d'intégration

Le processus de conception des systèmes hétérogènes embarqués peut être vu sous deux aspects : la conception des composants et leur intégration dans le même système. Notre travail concerne l'aspect d'intégration des composants. Un IP ou un composant virtuel peut être spécifié manuellement, acheté ou synthétisé par un outil de Synthèse de Haut Niveau (SHN). Dans le domaine de la conception assistée par ordinateur des circuits intégrés, l'intérêt porté à la SHN réside essentiellement dans la réduction des temps de conception des SoCs puisqu'une description comportementale est toujours plus simple à faire qu'une description structurelle RTL. Elle facilite également le travail de l'exploration architecturale puisqu'elle accélère la production d'essais successifs pour une architecture. L'intégration de ces IPs

générés dans un SoC peut entraîner un coût d'ingénierie très important. Seule l'automatisation de la génération d'interface peut amener à une solution en un temps raisonnable dans un flot de conception SoC basé sur des outils SHN. Pour promouvoir l'utilisation des outils de SHN, un outil de génération d'interface de communication est mis en œuvre pour l'intégration d'IP généré par GAUT dans un contexte contraint par une architecture cible de prototypage. Nous présentons dans cette section le flot de conception basé sur les outils SHN. Nous détaillons également l'outil SHN adopté dans le cadre de cette thèse : l'outil « GAUT ».

2.1. Flot de conception basé sur des outils SHN

L'une des évolutions récentes du flot de conception SoC est l'utilisation systématique des outils de SHN (Cf. figure 40). La SHN (synthèse de haut niveau) consiste à traduire une description comportementale séquentielle en un circuit intégré sous des contraintes architecturales [Cou03a]. Durant plusieurs années, les chercheurs travaillaient sur les outils de SHN pour réduire le temps de conception et les erreurs dues aux manipulations manuelles des conceptions. Malgré ces efforts, beaucoup de problèmes persistent lors de la SHN [Don04].

Les outils de SHN diffèrent selon le domaine d'application, la classe d'architectures ciblées et les techniques d'optimisation employées. Ils sont souvent spécifiques à des applications [Don04]. Ils proposent en effet des modèles différents et s'appuient sur des techniques différentes de modélisation. La synthèse de l'IP est spécifique selon des contraintes architecturales bien précises ce qui conduit à une diversité de l'interface aux entrées sorties de ces IPs. En effet, l'interface supporte plusieurs configurations de communication selon les mécanismes de synchronisation utilisés et les modes de consommation de données spécifiques à une modélisation au niveau cycle près/ bit près. Par ailleurs, comme le système impose un ordre particulier de transfert des données, l'étape d'intégration est en général faite à la main y compris pour les IPs synthétisés avec un outil de synthèse comportementale ou SHN. En effet, ces outils ne permettent pas de définir clairement les communications externes du circuit et la génération d'interfaces de communication spécifiques demeure le goulot d'étrangement dans un flot de conception SoC basé sur l'utilisation d'un outil SHN (CatapultC, GAUT, SystemC Compiler, Agility compiler de Celoxica, MMAlpha, etc…).

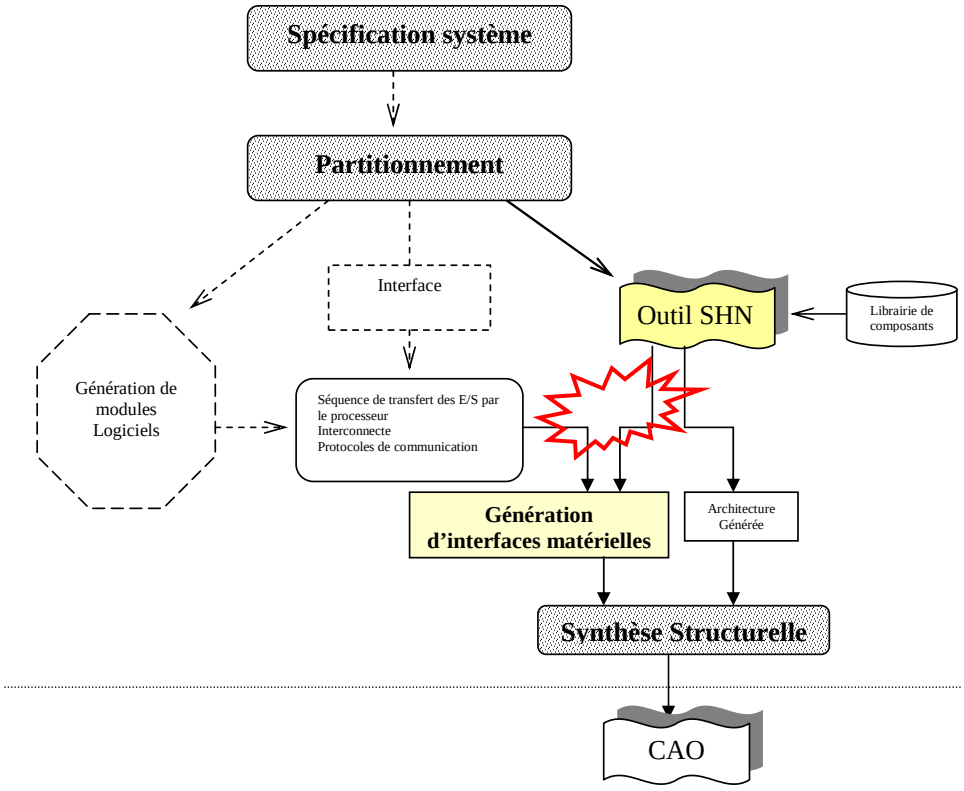

Figure 40: Flot de conception avec des outils de SHN

L'interface externe d'un IP est difficile à déterminer et son interprétation par la SHN est encore moins évidente. Ainsi, ces outils doivent faire face aux problèmes d'intégration de leurs IPs dans des environnements de simulation et de synthèse pour garantir plus d'efficacité. Par ailleurs, les performances d'un circuit à l'intérieur d'un système intégré sont difficilement évaluables autrement que par la simulation du système. Il faut donc construire un modèle au niveau matériel pour pouvoir évaluer le circuit dans son contexte d'utilisation. Le concepteur doit tenter plusieurs architectures matérielles de son circuit afin de pouvoir choisir l'architecture offrant le meilleur compromis entre les performances et le coût de son circuit. D'où l'utilité de l'automatisation de la génération d'interfaces de communication pour l'intégration saine et rapide des architectures générées par ces outils et pour la valorisation de l'utilisation d'un outil SHN dans un flot de conception SoC.

Nous présentons, dans la section 2.2, l'outil SHN considéré qui est l'outil GAUT. Les caractéristiques techniques de l'architecture de l'IP générée par cet outil seront détaillées pour pouvoir analyser le comportement à l'interface des entrées et des sorties et les hypothèses à considérer lors de l'intégration de l'IP GAUT.

2.2. Outil GAUT

Il existe différents types d'outils de synthèse selon le domaine d'application, la classe d'architectures ciblées et les techniques d'optimisation employées. Particulièrement, GAUT (Génération Automatique d'Unités de Traitement) est un outil universitaire de synthèse d'architectures pipelines développé depuis 1988 au LASTI (Université de Rennes) et poursuivi au LESTER (Université de Bretagne) depuis 1994. Il est destiné exclusivement à des descriptions avec un flot de données régulier. Il repose sur une méthode de conception dédiée aux applications de traitement de signal et d'image exécutées sous contraintes de temps réel [Sen95]. Il fonctionne indifféremment sur PC (sous Windows ou Linux) et sur station Sun (sous SunOs ou Solaris). Il est implémenté en C et supporte une interface en JAVA. Il cible principalement la conception sur FPGA mais peut aussi produire des ASICs.

GAUT se charge, sans l'intervention de l'utilisateur, du :

- Choix des données à mettre dans la mémoire,
- Ordonnancement des entrées/sorties et des accès à la mémoire et
- Dimensionnement du bus interne

Donc, l'outil assume une part importante des décisions d'implantation qui peuvent être en contradiction avec les contraintes d'intégration. De plus, GAUT requiert une bibliothèque d'opérateurs caractérisés. Ce sont des opérateurs arithmétiques, logiques et séquentiels. [Jeg99] montre que plus l'architecture du circuit est complexe, plus la surface prévue par GAUT s'écarte de la surface du circuit réel. Pour diminuer les variations, l'outil essaie de minimiser les connexions (qui sont l'un des principaux facteurs d'imprédictibilité) grâce à sa stratégie de partage des opérateurs [Jeg01]. Par ailleurs, les boucles non bornées ne sont pas acceptées par GAUT. Cela limite le champ d'application de l'outil.

Toutefois, cet outil présente une forte capacité d'être mis à jour selon les besoins du marché. L'outil essaie de minimiser les connexions (qui sont l'un des principaux facteurs d'imprédictibilité) grâce à sa stratégie de partage des opérateurs [Jeg01]. Par ailleurs, des travaux sont en cours au sein du laboratoire Lester pour élargir ses capacités au profit des applications TDSI d'une part et l'adapter à la synthèse des IPs comportementaux d'autre part [Cou03a]. Le travail dans le cadre de cette thèse tend à le rendre applicable dans sa version actuelle à une synthèse architecturale réutilisable.

2.2.1. Architecture de l'IP synthétisée par GAUT

Le modèle de l'architecture générée par GAUT repose sur un modèle générique de processeur de traitement de signal composé d'une unité de contrôle (MEF), d'une unité de

traitement (UT), d'une unité de mémorisation (UM) externe et d'une unité de communication (UCOM) comme le montre la figure 41. Le circuit généré est un circuit synchrone de niveau RTL avec une seule horloge. L'architecture présente des signaux de contrôle : reset et horloge.

L'architecture présente également des signaux d'entrées/sorties qui sont reliés à des bus externes au circuit. Les bus n'appartenant pas au circuit synthétisé sont présents pour véhiculer les données entre le circuit et l'unité de mémoire (UM) externe (figure 41).

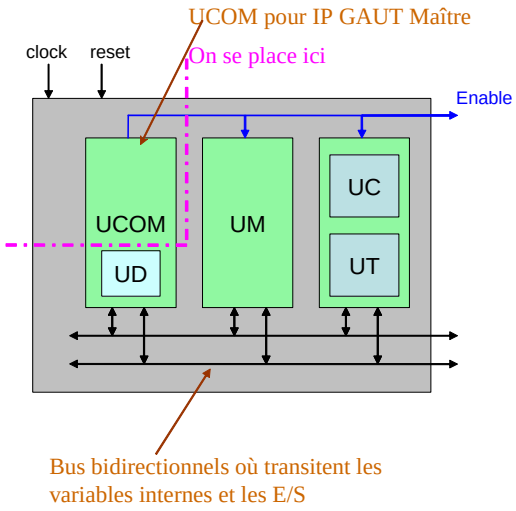

Figure 41 : Structure du circuit synthétisé par GAUT

Pour des raisons temps réel (la nature de l'algorithme et le résultat de l'ordonnancement), des entrées d'une architecture synthétisée par GAUT peuvent être présentées plusieurs fois sur les bus de l'UT. Une unité de « duplication » (UD) est présentée pour :

- faire une « copie » si le même coefficient est présent sur plus d'un bus au même instant.
- stocker dans un registre et le présenter sur les bus aux instants convenables si le même coefficient est présent à différents instants.

Nous considérons, comme indiqué dans la figure 41, une architecture synthétisée sans la considération de l'UCOM. Cette unité de communication que l'outil GAUT peut générer est spécifique à un contexte d'interfaçage spécifique [Bom04].

Pour interfacer l'architecture de l'IP RTL générée par GAUT (figure 41), nous avons besoin des informations sur les contraintes temporelles et l'ordonnancement des entrées et des sorties. Par la suite, la phase qui nous intéresse est la phase de synthèse d'une description comportementale de l'IP.

2.2.2. Phases de synthèse avec GAUT

La première phase de la synthèse est la phase de compilation. L'étape de compilation consiste en une vérification syntaxique et lexicale de la description source et une analyse ou une extraction de dépendance et de parallélisation du code. La description est transformée en un graphe de flot de donnée (SDFG : signal data flow graph) qui est le format intermédiaire supporté par l'outil GAUT. Ces transformations ont pour but la parallélisation automatique du code.

En entrée à cette phase, il est exigé d'avoir un code de spécification de l'algorithme au niveau comportemental :

- Le fichier.src contient le code VHDL à synthétiser. Le comportement est décrit par un seul processus (une seule entité/architecture) en VHDLGAUT[4].
- Le fichier.c contient le code C à synthétiser

En sortie de cette phase, il y a une génération du (figure 42) :

- fichier.gc représentant le modèle interne (graphe de flot de données) que GAUT considère pour la génération de l'architecture.
- fichier.op contenant un bilan des opérations existantes dans le code compilé.

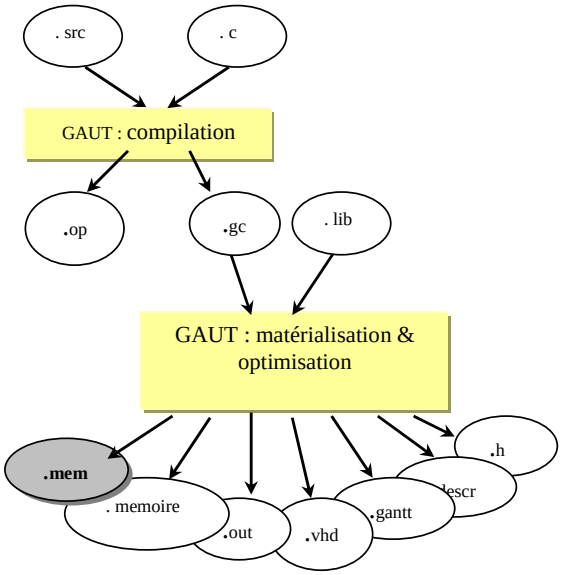

Figure 42: Phase de synthèse de l'outil GAUT

[4] C'est un sous ensemble de VHDL limité aux instructions séquentielles pour l'affectation d'expressions arithmétiques avec des structures conditionnelles ainsi que les boucles *for* et *while* déroulables « nombres de boucles finies lors de la synthèse ».

La deuxième phase consiste à générer une architecture associée à l'unité de traitement. Cette dernière est décrite à l'aide de plusieurs unités fonctionnelles interconnectées selon le modèle générique de GAUT :

En entrée de cette phase, il est exigé :

- d'avoir une caractérisation des opérateurs de la technologie ciblée. Cette caractérisation est regroupée dans un fichier d'extension « .lib ».
- de spécifier les options de synthèse sur les fichiers de sortie de la synthèse :
- pour la génération du fichier.vhd correspondant à la synthèse du fichier.src.
- pour la génération du fichier.h correspondant à la synthèse du fichier.c.
- pour la génération du fichier.mem.
- pour la génération du diagramme de Gantt.
- de spécifier les paramètres de la synthèse :
- le critère de coût : choisir la nature de l'optimisation (sur les opérateurs, sur les registres, sur le bus ou une combinaison d'optimisations différentes) selon les contraintes de l'utilisateur afin de pouvoir réduire le coût sur la surface.
- la latence (celle qui est spécifiée dans la phase de synthèse libellée cadence est prioritaire sur celle fixée dans le code du fichier.src).

En sortie, et si nous supposons que toutes les options existantes dans GAUT sont sollicitées, les fichiers suivants sont générés :

- Fichier d'extension «.vhd » : il contient le code VHDL de niveau RTL de l'algorithme correspondant au fichier.src
- Fichier d'extension «.h » : il contient le code SystemC de niveau RTL de l'algorithme correspondant au fichier.c
- Fichier d'extension «.mem » : il contient les informations sur les contraintes temporelles (l'ordonnancement) des entrées /sorties sur les bus.
- Fichier d'extension «.memoire » : il contient des estimations globales du coût de mémorisation des variables et des constantes.
- Fichier d'extension «.gantt » : il contient le diagramme de Gantt de l'architecture.
- Fichier d'extension « .descr » : c'est une interface entre GAUT et les outils de synthèse logiques commerciaux.

Les caractéristiques à considérer sont :

- Un IP GAUT est généré pour des applications flot de données : l'architecture est synchronisée par les données.

- Les données à entrer ne changent pas au cours du temps (pour le cas d'une seule itération de calcul)
- Le motif répété cycliquement dure plusieurs cycles pour GAUT ; l'IP GAUT est une architecture non régulière (l'architecture GAUT autorise des cycles d'horloge virtuelle sans lecture).
- L'affectation des variables d'entrées et de sorties aux ports de l'architecture générée par GAUT est effectuée de manière non prévisible.
- Les tailles (en bits) des différents ports sont toutes les mêmes pour une architecture générée par GAUT.
- Un même bus peut servir pour faire transiter une entrée ou une sortie : les bus sont bidirectionnels.

2.3. Conclusion

La présentation du contexte d'application de l'approche d'intégration favorise spécifiquement :

- un flot de conception des SoC basé sur des outils SHN.
- la génération d'interface de communication au profit des architectures générées par l'outil SHN GAUT. Ce qui revient à dire que l'IP généré par « GAUT » est l'IP cible à intégrer.

Nous détaillons dans la suite l'outil « Générateur d'Interface de Communication : GIC», favorisant, dans sa première version, l'automatisation du flot d'intégration proposée au profit des IPs générés par l'outil de SHN GAUT.

3. L'Outil de CAO « GIC »

Notre approche traite le problème d'intégration d'IPs à interface décrite au niveau CABA. Elle permet une évaluation haut niveau par la simulation et la synthèse de l'interface d'adaptation dans un contexte de réutilisation d'IPs ciblant les SoCs et le MPSoCs. Nous décrivons dans la suite l'outil générateur d'interface de communication « GIC ».

L'outil GIC rassemble et met en œuvre les concepts liés au flot d'intégration d'IP proposés dans ce mémoire (chapitre III). Il nous a permis d'automatiser les étapes de ce flot et de gagner en productivité dans un flot de conception SoC.

La description de l'environnement se fait d'une manière incrémentale par l'utilisation de plusieurs outils, chacun affecté à un rôle spécifique pour la génération du code de

l'interface de communication appropriée au contexte de réutilisation cible. Les données sont mémorisées dans des structures de fichiers XML permettant ainsi une interopérabilité facile entre ces sous outils. Les commandements qui régissent le développement de l'environnement sont les suivants :

- réduire le temps réservé à l'intégration d'IP, qui constitue un goulot d'étranglement dans le processus de conception des SoCs. Cela implique des méthodes et des outils pour automatiser les tâches principales dans le flot : vérification de la compatibilité et génération de code. Vu la complexité du problème traité et la nouveauté de ce type de recherches, il paraît difficile d'obtenir des solutions complètement automatiques dans un avenir proche. D'où le deuxième commandement :
- offrir une interaction facile entre les concepteurs et les outils. Il faut pouvoir mélanger l'intégration automatique avec des interventions manuelles pour obtenir des solutions efficaces.
- assurer une intégration facile dans les environnements de conception existants. Il s'agit de faciliter la réutilisation de composants existants. Ce point est nécessaire pour avoir des outils flexibles pouvant être utilisés dans un contexte assez large.
- permettre une intégration facile dans les méthodologies de conception existantes. Il s'agit d'être compatible avec les autres outils d'aide à la conception existants (spécification, partitionnement, compilation, simulation, vérification).

3.1. Implémentation du GIC

Afin de produire une application évolutive et modulaire, le choix a été fait sur la modélisation orientée objet. L'utilisation d'une approche objet est cruciale car elle est particulièrement adaptée à la modélisation de notre environnement qui doit être évolutif.

3.1.1. Modèle d'intégrité entre les outils

Les outils qui ont été intégrés dans l'environnement peuvent se classer en deux catégories : les outils développés en interne et adaptés directement à l'approche de l'intégration d'IP, et les outils externes :

- Les outils externes : outil GAUT, outil de spécification de l'architecture SoC (outil externe qui utilise le résultat de l'outil GIC
- Les outils internes sont les sous outils de l'outil GIC (plus de détails dans la section 3.2).

Les outils externes assurent la réalisation des interfaces qui permettent la visualisation, l'édition, la résolution des problèmes d'adaptation. Ils vérifient si les fichiers sont cohérents et surtout s'ils ne sont pas utilisés par d'autres outils, etc. le modèle d'intégration de ces outils se déroulent autour d'un modèle intermédiaire et est assuré à l'aide d'un traducteur (cf. figure 43).

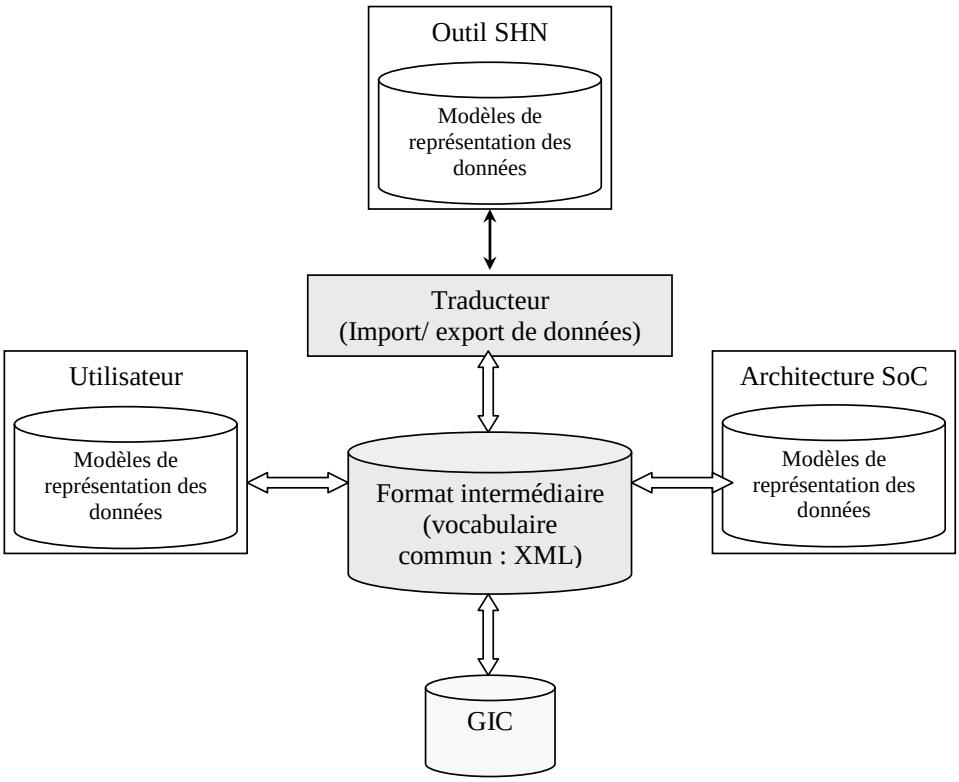

Figure 43. Modèle d'intégration d'outils autour d'un format intermédiaire

Où :

- Le format intermédiaire : Le format intermédiaire utilisé doit être neutre, c'est-à-dire indépendant des outils, des méthodes et des processus de conception. Il doit être aussi suffisamment flexible pour maintenir les sémantiques entre les outils, garantir l'intégrité du système, préserver la structure du système et les décisions fondamentales dont dépend sa performance.
- Le processus de traduction : Généralement, un traducteur spécifique à chaque outil est construit pour achever l'étape d'intégration d'outils dans un environnement de conception donné sans l'intervention des concepteurs.

Le format d'échange (format intermédiaire) inter outils (externes ou internes) choisi est XML (eXtensible Markup Language). XML est un méta-langage standardisé par *W3C*, il a un très grand succès dans le monde de l'informatique grâce à sa souplesse, sa clarté et sa

facilité à être analysé. Il facilite l'intégration des outils et l'échange d'informations entre les différents environnements.

C'est un format d'échange standard et facile à comprendre car humainement lisible. Il peut être édité par un simple éditeur ou dans des environnements XML [XML00]. XML est basé sur le concept de balises : toute information est délimitée par une balise ouvrante et une balise fermante. Les balises XML peuvent avoir des attributs et peuvent être hiérarchiques. Pour obtenir un langage dérivé de XML, il faut définir des balises particulières et préciser les contraintes qui leur sont associées. Ces contraintes concernent leurs attributs ou la hiérarchie. Ces informations sont à donner dans un fichier à part nommé "DTD" (Document Type Definition). Ce langage possède deux caractéristiques intéressantes pour le but fixé : c'est un méta-langage qui, grâce au concept de DTD, n'a besoin que d'un seul analyseur quelque soit le langage dérivé et c'est un standard recommandé pour la représentation de données. XML est cependant une notation plus qu'un langage, en particulier il ne dispose pas de sémantique.

Nous avons implémenté les graphes d'ordonnancement de données proposés pour l'abstraction du transfert des données initialement considérée dans le flot d'intégration sous formats XML pour les raisons suivantes :

- XML est un langage de balisage normalisé qui peut être utilisé pour représenter la structure logique de n'importe quel type de documents contenant du texte libre.
- XML permet l'interopérabilité : il est extensible, portable et, par opposition à HTML (qui est également portable, mais non extensible), définit une norme concernant la structure des documents et non pas leur contenu.
- XML est en quelque sorte un méta-format de document. Il présente un grand nombre d'APIs « Application Programming Interface » disponibles pour l'interpréter.

Par ailleurs, l'outil GIC est implémenté avec le langage JAVA. Le langage JAVA a été choisi pour son caractère à la fois objet et portable (multi-plateformes). Par ailleurs, il permet de manier des fichiers de configuration avec le langage XML. En plus, il existe plusieurs API Java pour XML comme SAX « Simple API for XML » et JDOM « Java Document Object Model » [Vid00] que nous avons utilisés au cours de l'implémentation de l'outil GIC. En effet JDOM :

- permet de construire des documents XML,
- permet de naviguer dans leur structure,

- permet d'ajouter, de manipuler, de modifier et/ou de supprimer leur contenu d'une manière plus simple qu'avec les APIs classiques.
- utilise des collections SAX pour « parser » les fichiers XML. Ce type de parseur (analyseur) utilise des événements pour piloter le traitement d'un fichier XML.

3.1.2. Etapes du Flot d'intégration dans l'outil GIC

Les principales étapes du flot d'intégration (cf. figure 44) dans le GIC sont les suivantes :

- saisie/importation des contraintes d'intégration ;
- modélisation en graphes des contraintes sur le transfert des données ;
- vérification de la compatibilité entre graphes modélisant l'échange des données entre l'IP et le système ;
- sélection des modules de l'architecture de l'interface appropriés au choix du concepteur ;
- génération automatique du code (VHDL ou SystemC) et du pilote logiciel ;
- configuration des paramètres génériques.

La figure 44 montre le flot du GIC. L'expérimentation de l'outil GIC (Générateur d'Interface de Communication) que ce soit pour la simulation ou pour la synthèse exige :

- du côté du système :
  - Le partitionnement et l'ordonnancement des tâches de l'application cible afin :
    - d'identifier le comportement des données du côté du système,
    - et de choisir la conception de l'architecture de l'interface.
  - L'adaptation entre le protocole de l'interconnecte du système et l'interface générique ; ce qui revient à spécifier les adaptateurs « interconnecte-FIFO » et « FIFO-interconnecte ».
  - La prise en compte des contraintes du système (mode de transmission, taille des données etc.)
- du côté de l'IP cible :
  - Une caractérisation spatio-temporelle à l'interface du comportement de transfert des entrées/sorties au cycle près et au bit près.

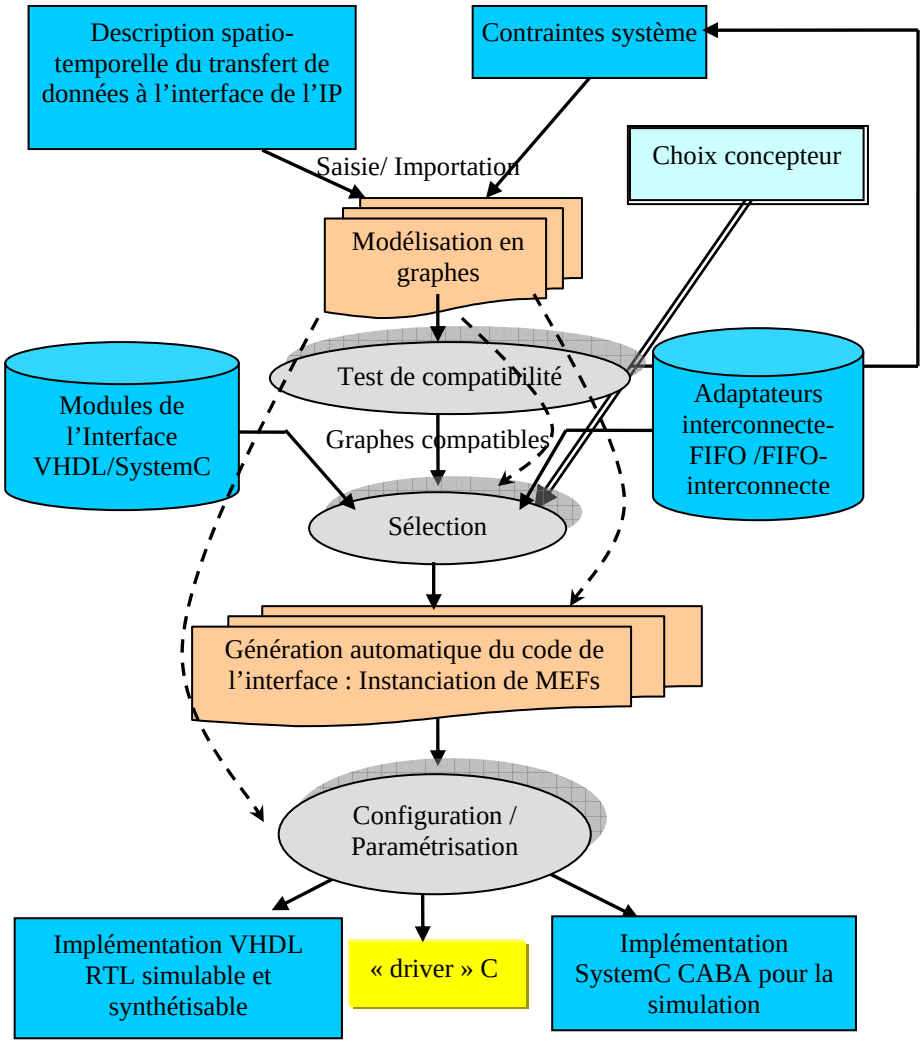

Figure 44. Flot du GIC

A partir de ces fichiers de configuration, l'outil génère les différents graphes (GOESS, GOES, et GOS) au format XML. Si la compatibilité est vérifiée, l'outil permet de générer le code de l'interface de communication :

- En instanciant et en configurant les modules matériels à partir d'une librairie. En effet, l'outil dispose d'une librairie de modules de l'interface (décrits en langage VHDL ou SystemC : l'outil cible un contexte de synthèse/simulation) décrivant la conception en MEF de l'architecture générique de l'interface.
- En générant le pilote matériel « driver C » à partir du graphe d'ordonnancement du système.

L'intégration de l'interface dans un contexte de réutilisation se fait par un simple copier coller de ces fichiers dans le contexte d'intégration.

3.1.3. Entrée de l'outil

L'outil de génération GIC prend en entrée les éléments suivants (cf.figure 45) :

- Une architecture qui représente une description annotée en graphes de tâches de l'architecture matérielle et logicielle cible ; cette description permet de procéder à une interface pour le cas monoprocesseur ou pour le cas multiprocesseurs. Elle identifie également les paramètres liés au système intégrant concernant le mode de transfert, la taille des données, les types de données, les priorités des tâches etc.
- Une description du comportement spatio-temporel à l'interface de l'IP
- Une bibliothèque de sous module d'interface (objets template, classe MEF) pour la génération d'adaptateur matérielle.

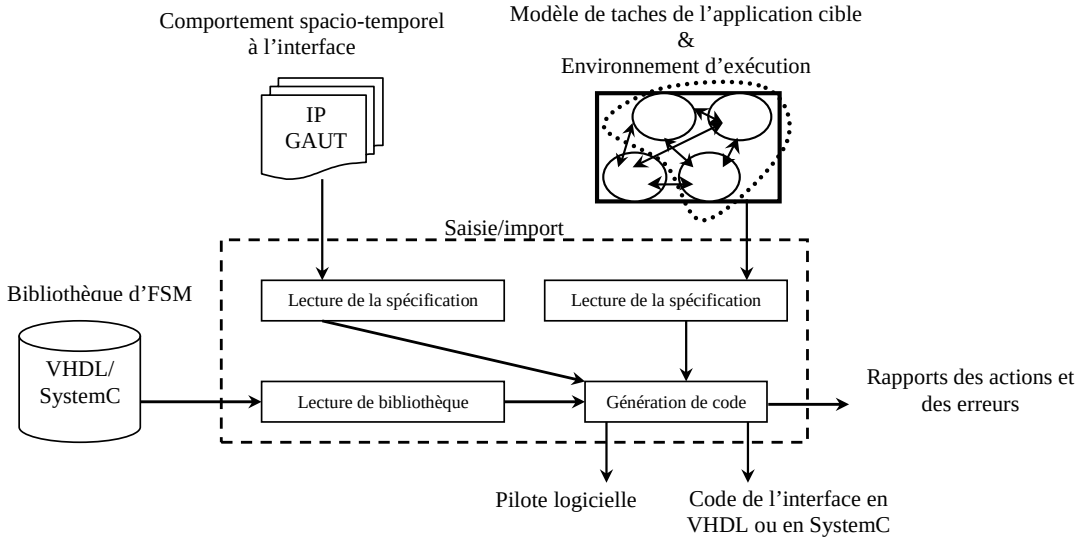

Figure 45 : Entrées/ sorties de l'outil GIC

3.1.4. Sortie de l'outil

La sortie de l'outil (cf. figure 45) de génération des interfaces est le code de l'adaptateur matériel de l'IP cible et le pilote logiciel adéquat. Le code de l'interface de communication généré est en SystemC ou en VHDL pour la partie matérielle ; le code de l'interface logicielle est en C.

Il contient également :

- des entêtes permettant d'adapter le code de l'application au système d'exploitation généré et à l'architecture cible : assemblage.
- des rapports fournissant des informations sur les actions effectuées durant la génération ainsi que les erreurs survenues.

3.1.5. Modélisation en graphes

Les trois types de graphes sont enregistrés dans des fichiers XML considérant la même organisation sémantique et syntaxique (hiérarchies et notations) définissant GOS, GOES et GOESS. Ces fichiers fournissent des informations utiles pour le déroulement des étapes du flot GIC, pour alimenter les outils internes du GIC.

3.2. Outils internes au GIC

L'outil GIC repose sur trois sous outils ou encore sur trois outils internes :

- GIC_*checker* : outil de vérification de compatibilité
- GIC_*interface_generator* : outil de génération de code de l'interface de communication (conception, sélection, configuration, paramétrage, génération VHDL ou SystemC)
- GIC_*driver_generator* : outil de génération de pilote logiciel ou de pilote logicielle pour l'interface matérielle l'outil

3.2.1. GIC_*checker*

L'outil GIC-C*hecker* (cf.figure 46) est un composant logiciel qui permet d'interpréter un ensemble de règles, spécifiées en langage XML, et de les appliquer sur un modèle d'entrée issu d'un processus de traduction automatique.

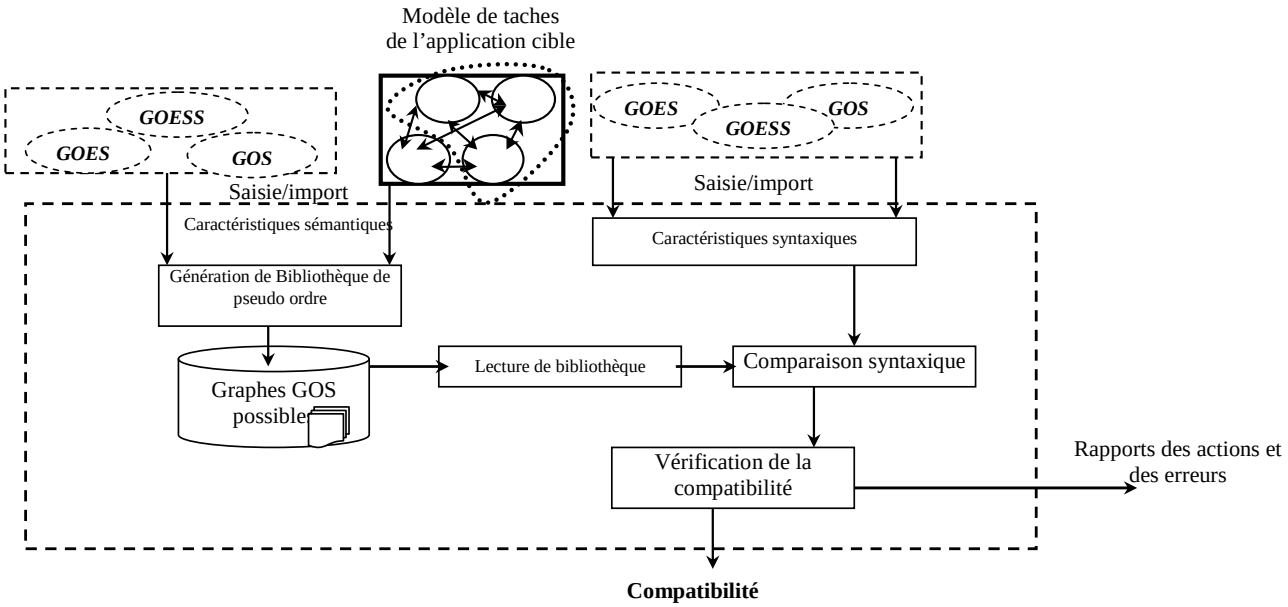

Figure 46 : Architecture du GIC_Checker

L'outil GIC-Checker interagit avec une bibliothèque de fichiers XML qu'il crée lui même. Cet outil permet de vérifier d'une façon automatique la bonne formation du modèle XML. Cette vérification est faite en deux étapes :

- la première étape consiste à vérifier la structure syntaxique (structure XML).
- La deuxième étape consiste à vérifier la sémantique (motif, pseudo ordre)

Les règles syntaxique et sémantique sont stockées dans un fichier XML. L'analyseur XML (*parser*) permet la lecture du fichier XML. Ensuite, il crée une structure en mémoire (arbre) des différents objets permettant la description des différentes règles et retourne un objet de type *Document* dans le *Document Object Model – DOM* Consortium *W3C* pour manipuler les documents XML. L'objet *Document* représente la racine de la structure créée en mémoire et permet un accès aux différents objets de l'arbre. Les différentes opérations (création, suppression, etc.) appliquées sur les règles sont exécutées directement sur la structure en mémoire.

Les entrées de l'outil sont :

- le modèle de graphes GOS considéré en entrée à l'outil GIC sous forme de fichier XML;
- une base de données contenant un ensemble de règles spécifiques au modèle GOS généré pour couvrir l'ensemble des pseudo ordre possible pour le cas cible.

La sortie de l'outil dépend de deux cas :

- si l'ensemble de règles à vérifier par rapport au modèle GOS d'entrée est complètement validé alors le GIC_*Checker* affiche un message indiquant la validation de cet ensemble de règles ;
- dans le cas contraire, c'est-à-dire si l'ensemble de règles syntaxiques et sémantiques à vérifier n'est pas complètement validé, alors le GIC_*Checker* génère un rapport de violation de règles. Ce rapport indique les noms (s'il existent) des objets *Document* présentant les origines des violations. Techniquement, le rapport est un fichier texte simple qui peut être visualisé à l'aide d'un simple éditeur de texte.

Les différentes étapes et composants de l'outil GIC-*Checker* sont (cf.figure 46):

- lecture de la description de l'application : modèle de taches de l'application
- génération de la bibliothèque de pseudo ordre sous format XML
- lecture de la bibliothèque

- comparaison syntaxique GOS et GOS générés : cette étape représente un processus qui commence par la récupération d'une liste de toutes les règles produites par l'étape précédente.
- Vérification de la compatibilité : c'est le résultat de la comparaison qui servirait à continuer le processus de génération d'interface (passer dans le sous outil GIC*_interface _generator)* ou à retourner à l'étape de saisie et d'import (interface utilisateur).

3.2.2. GIC*_interface_generator*

L'étape de génération de code prend en entrée les listes d'éléments provenant de l'étape de sélection de code et les paramètres produits par l'étape d'analyse de la description de l'architecture. Cette étape construit une liste de tous les paramètres associés à chaque élément précédemment sélectionné. Ceci permet de faire appel à un processus de paramétrage et d'optimisation des éléments afin de générer leur code spécialisé respectif. Ensuite, ces éléments sont assemblés pour concrétiser sous forme de code les relations entre chaque sous modules requis et/ou fournis.

Le processus de génération de code se compose de cinq étapes élémentaires :

- ***l'étape d'analyse* :** cette étape permet l'extraction d'informations relatives au système à partir de l'architecture. Par exemple, les informations telles que les types de données et leur taille, les priorités des tâches, etc. Ceci permet de guider le processus d'intégration au cours des étapes suivantes ;
- ***l'étape de sélection* :** cette étape localise, compare, et sélectionne les sous modules qui fournissent les services de l'interface à partir d'une bibliothèque. Ceci est fait selon les informations de conception extraites précédemment (graphes). Cette étape doit être exécutée récursivement jusqu'à ce qu'un ensemble convenable (optimal) de sous modules qui fournissent tous les fonctionnalités requises soit identifiées ;
- ***l'étape de spécialisation* :** cette étape permet la personnalisation des sous modules, sélectionnés au cours de l'étape précédente, pour satisfaire les exigences des services à fournir (par exemple : les types de données, la taille du bus, etc.). Elle permet aussi de vérifier la compatibilité de ces sous modules. Les valeurs des paramètres finaux sont stockées dans des fichiers de configuration utilisés pour la génération automatique au cours des étapes suivantes ;

- ***l'étape de génération* :** cette étape a besoin de deux types de fichiers : (1) les fichiers contenant le macro code des composants configurables et (2) les fichiers contenant les paramètres de configuration. Elle permet de générer des fichiers contenant le code source spécialisé correspondant au comportement de chaque sous modules sélectionné ; en effet au début le code généré correspond à un modèle d'implémentation générique décrit en macro-langage. Une implémentation générique n'est ni simulable ni synthétisable, car elle n'est pas encore configurée. Le type de données est abstrait, le nombre, la taille et la direction des ports sont encore génériques. Pour une application donnée, les paramètres d'allocation (type de données, taille du port, etc.) sont utilisés pour configurer le code générique sélectionné à partir de la bibliothèque. Cette expansion du code génère le code final de l'interface qui est simulable et synthétisable.
- ***l'étape d'assemblage* :** au cours de cette étape les composants produits par les étapes précédentes sont assemblés pour fournir une architecture complète et raffinée. L'interface utilisateur pour la spécification des paramètres de l'interface se fait à l'aide d'un éditeur XML. L'utilisateur peut rentrer ses spécifications applicatives, réaliser les transformations qui vont imposer des décisions selon des paramètres, et enfin sauvegarder les résultats de ces transformations (configuration, architecture, tache, pseudo ordre, etc.).

Une étape est nécessaire pour le déroulement des étapes précédentes**.** C'est une étape de ***lecture de la bibliothèque***. Elle consiste à charger la bibliothèque architecturale. Cette lecture permet de connaître la disponibilité des ressources architecturales nécessaires pour la réalisation de l'interface cible. Cette bibliothèque contient différents sous modules de l'architecture de l'interface. La fonctionnalité est indiquée sous forme d'un chemin vers l'implémentation.

3.2.3. Bibliothèque de l'outil : Structures et relations utilisées

La bibliothèque de sous modules de l'interface de communication est indispensable pour le processus d'intégration. Elle contient des modules configurables (paramétrables) utilisés pour la composition. Ces modules possèdent des granularités différentes : ils peuvent être des modules d'interface de base ou des sous-modules complexes. Chaque composant communique avec l'environnement extérieur (le reste de l'interface) via un ensemble de signaux qui encapsule une liste de services fournis/requis par ce module. Cette bibliothèque est basée sur trois concepts qui donnent chacun une vue différente :

- le sous module : représente une partie de l'interface de sorte que la structure de l'interface complet est modélisée par un ensemble de sous modules liés ;
- le service : représente une fonctionnalité du sous module, de telle sorte que le comportement fonctionnel d'une interface complet est modélisé par un ensemble de services.
- l'implémentation : représente une réalisation particulière d'une partie du comportement précis de l'interface, de telle sorte que le code d'un interface complet est vu comme un assemblage d'implémentations paramétrées. Une implémentation peut être compatible avec une conception de la structure de l'interface, et incompatibles avec d'autre. C'est aux implémentations que le driver logiciel générique de l'interface est associé.

Ces trois concepts ne sont pas indépendants, il y a des relations entre les éléments et les services, les éléments et les implémentations, et les services et les implémentations.

- Les relations entre les sous modules et les services : ce sont des relations de dépendance indirectes entre les sous modules basées sur le concept de services requis et de services fournis. Ces relations sont présentées sous forme d'un graphe orienté (figure 47). Chaque noeud du graphe représente soit un sous module, soit un service. Chaque arc orienté relie soit un noeud sous module (sm) à un noeud service (si) pour modéliser le fait que l'élément requiert le service, soit un noeud service à un noeud sous module pour modéliser le fait que le sous module fournit le service.

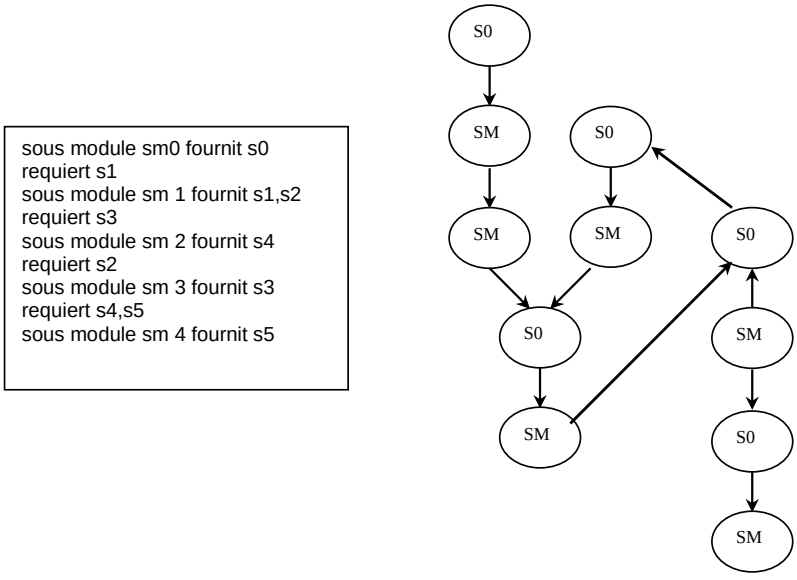

Figure 47 : Relations de dépendance entre sou modules et services

Nous distinguons quatre types de services : le contrôle, mémorisation FIFO, synchronisation, inhibition. Selon les spécifications d'entrée, les types de services sont spécifiés pour exploiter les sous modules de la bibliothèque.

– Les relations entre les sous modules et les implémentations : chaque sous module possède un ou plusieurs implémentations. Les implémentations d'un sous module sont organisées suivant un arbre hiérarchique (figure 48), les fils d'une implémentation étant toujours compatibles avec moins d'architectures que leur père. Pour décrire complètement un sous modules compatible avec une architecture donnée, il faut prendre les sources associées à chaque implémentation traversée et parcourir l'arbre en profondeur avec comme critère de choix la compatibilité avec le composant virtuel. Si aucune feuille n'est atteinte, alors l'implémentation est invalide pour l'architecture visée. Si toutes les implémentations d'un sous modules sont invalides, alors le sous modules est lui aussi invalide. La figure 48 montre un exemple d'arbre d'implémentations d'un sous modules compatible avec la conception 1 (c1) (cas monoprocesseur) SystemC (s). (les implémentations en gras sont celles compatibles avec le multiprocesseur SystemC). Ce type de structure est archivé dans des fichiers XML considérant la même organisation sémantique et syntaxique.

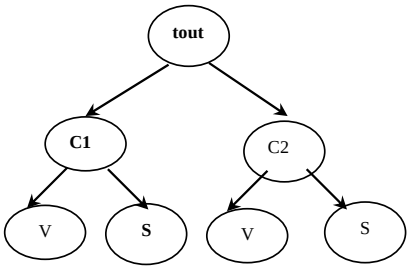

Figure 48 : Arbre d'implémentation d'un sous module SystemC compatible monoprocesseur

– les relations entre les services et les implémentations permettent d'éviter l'inclusion du code d'implémentation d'un service fournis par un sous module mais non requis par d'autres sous modules dans l'interface de communication généré.

3.2.4. GIC_*driver_generator*

Le pilote logiciel associé à l'interface de communication permet de commander la communication des données entre l'IP et son environnement d'intégration. Nous avons considéré une communication avec une mémoire partagé. Ce qui conserver le paradigme de communication « espace d'adressage partagé ». Il s'agit pour chaque transfert d'un mode de communication via mémoire partagée. D'où la nécessité d'un pilote qui orchestre le transfert d'une part entre l'initiateur ou le processeur et l'espace mémoire et entre la mémoire et l'accélérateur d'un autre part. Pour cela, nous avons considéré particulièrement que les données rentrant / sortant successivement sur un port sont rangées successivement en mémoire. Par exigence de base de la méthodologie d'intégration proposée, l'ordonnancement des données (rentrant et sortant sur un port) suit un pseudo ordre pour une itération de calcul. Ce pseudo ordre est exactement le graphe d'ordonnancement système (GOS) spécifié en entrée à l'outil GIC.

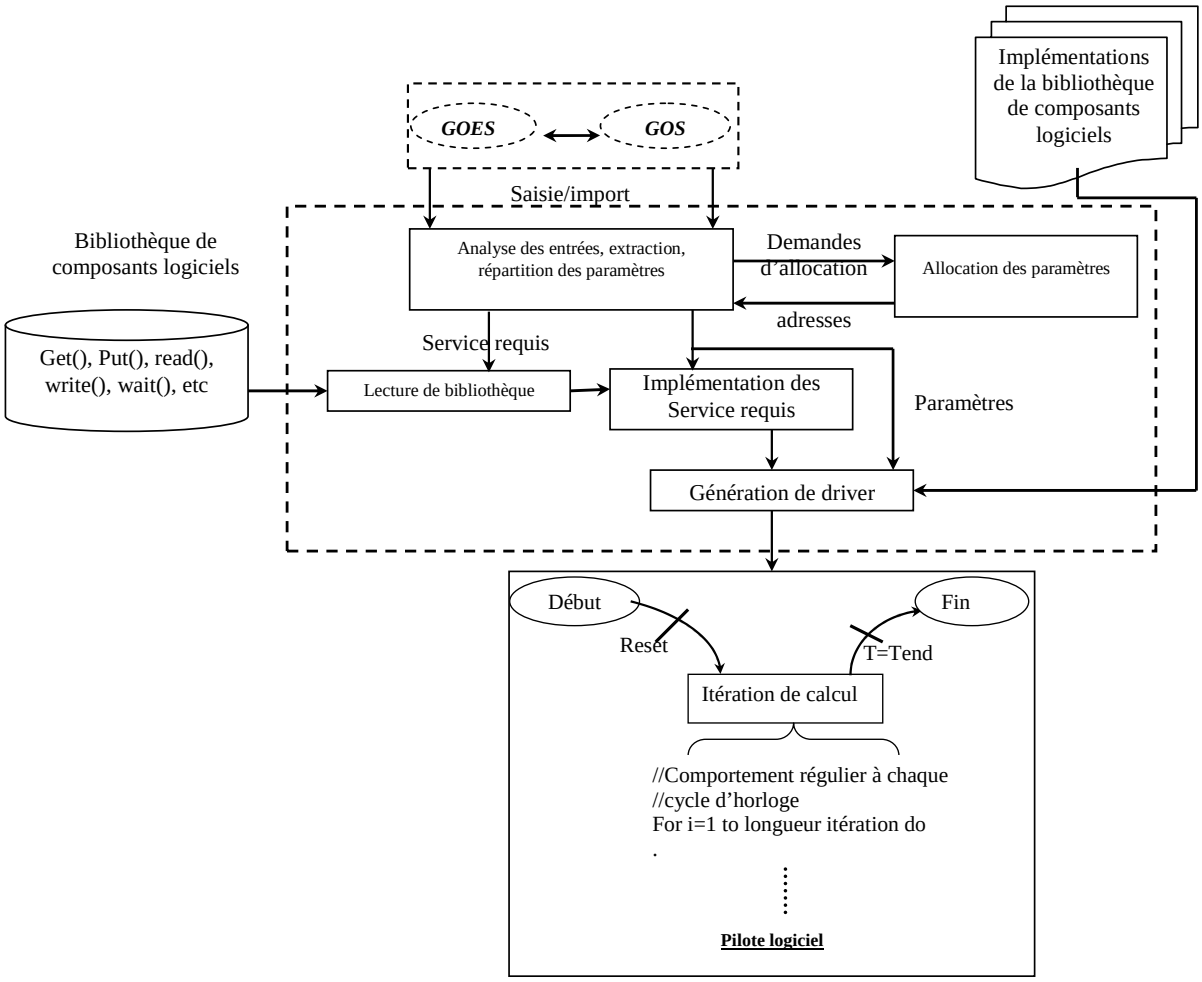

Figure 49 : Architecture du GIC_*driver_generator*

Pour chaque architecture cible, nous considérons une nouvelle définition des instructions de pilotages utilisés dans le développement de la partie logicielle (différentes implémentations de la bibliothèque de composants logiciels)

Considérant l'ordre donné par le graphe GOS et les transitions temporelles données par les graphes GOES donné pour chaque structure de donnée, le GIC_*driver_generator* construit le code du « driver » adéquat (cf.figure 49):

- Il analyse les entrées et les sorties et extrait la répartition spatio-temporelle des paramètres : Les informations et les paramètres contenus dans cette description sont de plusieurs types :
  - les informations topologiques provenant de la hiérarchie des modules de la description : il s'agit du nombre d'initiateurs, du nombre de tâches logicielles, quels processeurs exécutent quelles tâches
  - les informations fournies par les paramètres des modules : il s'agit des caractéristiques des ressources locales, etc ;
  - les informations fournies par les paramètres d'allocation pour les tâches et les divers éléments du système d'exploitation : dans ces paramètres se trouvent les adresses des données en mémoire, la taille des données, les types de données, les priorités des tâches, etc
- à partir de cette description, le GIC_*driver_generator* peut faire l'allocation des paramètres en demandant des adresses de la mémoire
- suivant le service requis (envoi, réception, synchronisation etc.), une étape de lecture de bibliothèque de composants logiciels est lancée pour la sélection des services. Cette bibliothèque contient tous les composants logiciels élémentaires et génériques qui peuvent être spécialisés et assemblés pour obtenir un pilote complet, ainsi que toutes les informations qui définissent ces composants et leur environnement d'utilisation.
- La lecture de bibliothèque permet une implémentation des services requis en langage évolués (avec des routines générales :read(), write(), get(), put() etc)
- La génération de driver permet une implémentation des services en considérant la répartition et l'allocation des paramètres, les paramètres, et l'implémentation des composants logiciels de la bibliothèque en langage C.

3.3. Modélisation du GIC

Nous avons utilisé comme langage de modélisation le standard UML « Unified Modeling Language ». Le diagramme de classes de la figure 50 résume la conception des relations entre les plus importantes classes utilisées dans le GIC. Les classes sont représentées sans leurs attributs ni leurs méthodes pour plus de lisibilité du diagramme.

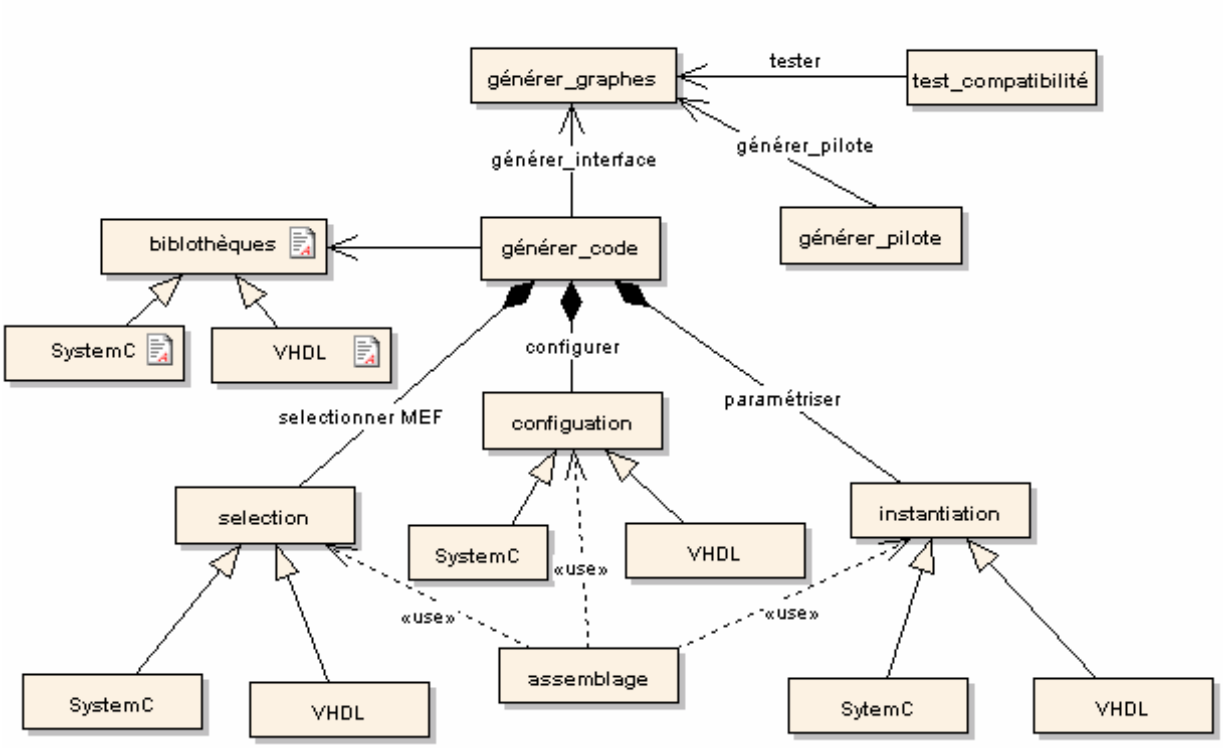

Figure 50. Diagramme de classes du GIC

La classe « générer_graphes » permet l'extraction des données (à partir du fichier de caractérisation de l'IP à intégrer ou selon la saisie de l'utilisateur) et la génération des graphes d'ordonnancements tels que :

- GOES : permet de présenter les données dans un fichier .xml intitulé GOES, ordonnées selon les transitions temporelles.
- GOESS : permet de présenter les données dans un fichier .xml sous forme de deux catégories : les entrées et les sorties.
- GOS : permet de présenter les données dans un fichier .xml « GOS » selon des structures sans les contraintes sur les délais.

La classe « test_compatibilité» permet de vérifier la possibilité d'intégration de l'IP dans l'environnement considéré. Les fichiers représentant les graphes servent pour analyser la compatibilité de l'ordonnancement des données à l'interface de l'IP avec celle exigée par le système intégrant.

La classe « générér_pilote » permet de générer le « driver » logiciel de l'interface de communication.

La classe « générer_code» représente la classe principale qui fait l'appel de toutes les autres classes et donc de générer les fichiers qui constituent l'interface de communication matérielle. Elle est composée de trois classes :

- Classe « sélection » : cette classe permet de sélectionner les sous_modules constituants l'interface de communication matérielle à partir de la bibliothèque adéquate au choix de l'utilisateur.
- La classe « configuration » : cette classe permet de configurer les modules choisis à partir des fichiers XML.
- La classe « instantiation » : cette classe permet d'instancier autant que nécessaire les modules choisis.

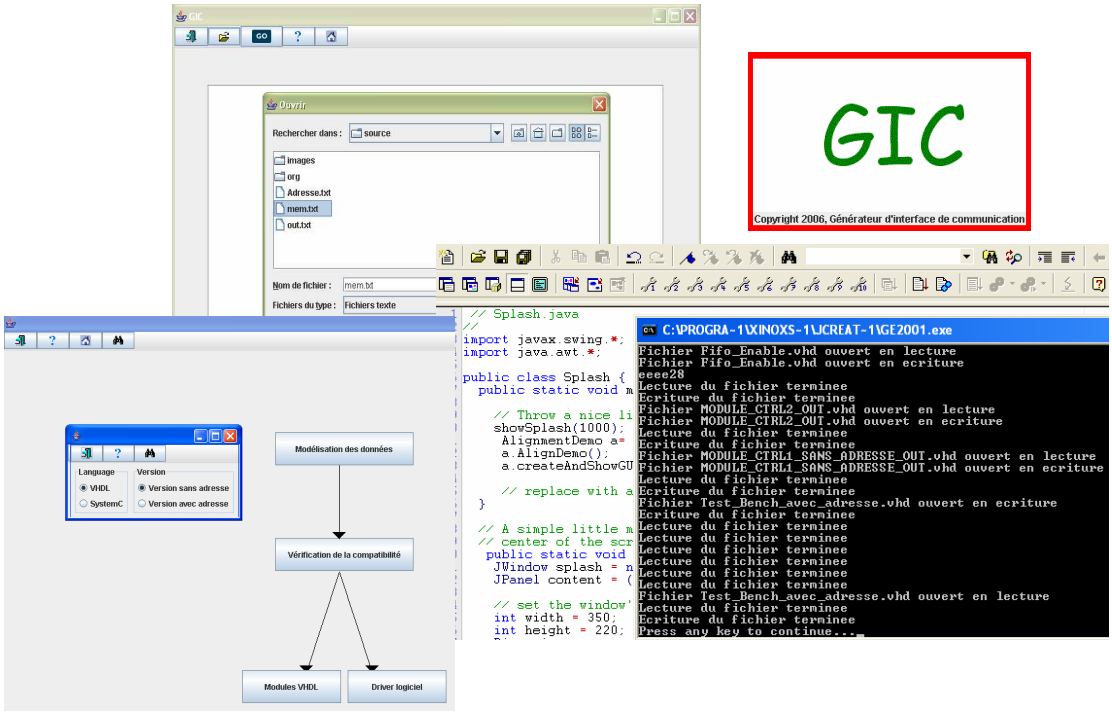

Figure 51. Snapshots de l'outil GIC

La figure 51 montre l'interface de l'outil GIC. Le choix (partie encadrée en traits pointillés) considère une génération de modules d'une interface de communication en VHDL synthétisable en tenant compte de la première conception de l'interface de communication (onglet « version sans adresse » dans la figure 51) pour un IP généré par GAUT (fichier de caractérisation mem.txt).Le flot d'intégration associé à ce choix (figure 51) comporte essentiellement 4 menus dont le premier permet de modéliser les données à l'interface sous

forme de fichiers XML. Ensuite, le générateur permet de vérifier la compatibilité du transfert des données avec celles envoyées du côté système. Le troisième menu permet de générer les modules nécessaires pour assurer la bonne communication de l'IP avec le système entier que ce soit pour la simulation (SystemC) ou pour la synthèse (VHDL) (dans ce cas de figure, l'expérience cible le langage VHDL : cf. fenêtre DOS dans figure 51). L'outil génère parallèlement l'interface logicielle ou le « driver » logiciel qui s'exécute par le processeur de l'architecture cible. En effet, le processeur pilote l'IP via l'interface matérielle.

3.4. Conclusion

A travers ce chapitre, nous avons présenté l'environnement GIC qui intègre l'approche d'intégration proposée dans le chapitre 3. L'environnement développé est de type générateur de code. Il est facilement extensible pour être utilisé avec d'autres outils de conception réalisant d'autres tâches de co-design. Ce qui permet d'augmenter la productivité dans un milieu industriel.

GIC est capable de modéliser une interface de communication pour la paramétrer et de générer son code VHDL synthétisable et son code SystemC simulable. Cet outil est facilement évolutif. Il pourra également être utilisé comme support de génération automatique et d'évaluation des résultats de simulation et des résultats de synthèse

L'expérimentation de l'approche d'intégration automatique à travers l'implémentation et le test de l'outil « GIC » est illustrée d'abord pour la simulation et puis pour la synthèse. L'approche a été testée pour l'accélération d'une application multimédia nommée « pipeline graphique », « synthèse d'images 3D » ou encore « pipeline 3D » qui sera présentée dans chapitre suivant.

CHAPITRE 5. EXPERIMENTATION DE L'APPROCHE D'INTEGRATION : EXEMPLE ET VALIDATION

1. Introduction

La génération de l'interface de communication est testée pour le contexte de simulation en utilisant la plateforme SoCLiB. Pour le contexte de synthèse, l'interface est testée en utilisant le processeur NIOS équipé du bus AVALON. L'application cible est la « synthèse d'images 3D ». Divers tests ont été effectués avec l'outil GIC pour les deux contextes : simulation et synthèse.

Dans la section 2, nous présentons l'application cible « synthèse d'image 3D » et la démarche utilisée pour le choix d'un IP accélérateur : l'IP « produit matriciel ». Dans la section 3, nous développons la démarche de validation de l'approche pour la simulation. Pour cela, nous présentons l'environnement de simulation « SoCLiB », la bibliothèque de modules de l'interface, les fonctionnalités du GIC pour la simulation et enfin les résultats de simulation. Dans la section 4, nous présentons la démarche de validation de l'approche pour la synthèse ; nous présentons les modules VHDL de l'interface et les fonctionnalités principales du GIC pour la synthèse. Une démarche d'optimisation des FIFOs est proposée pou améliorer les résultats de synthèse.

2. Exemple d'application cible : « Pipeline 3D »

La chaîne de production d'une image 3D est appelée pipeline graphique. Elle est formée par l'ensemble des opérations nécessaires pour afficher un objet 3D vu depuis une position et avec une orientation donnée. En effet, l'écran d'un ordinateur est seulement capable de représenter des coordonnées en deux dimensions. Comme les écrans de sortie tridimensionnelle n'existent pas encore, nous sommes amené à transformer les coordonnées 3D en coordonnées 2D. Pour se faire, nous utilisons la projection par perspective, qui permet de représenter correctement la « profondeur » d'un objet en donnant l'impression de volume. Le « pipeline 3D » est l'ensemble des étapes nécessaires pour la création et la visualisation d'une image 3D. Cette chaîne est décomposée en un ensemble d'opérations nécessaires pour afficher un objet 3D observé à partir d'une position et avec une orientation donnée. Ces opérations constituent diverses étapes de l'application illustrées par la figure 52 [Fol95].

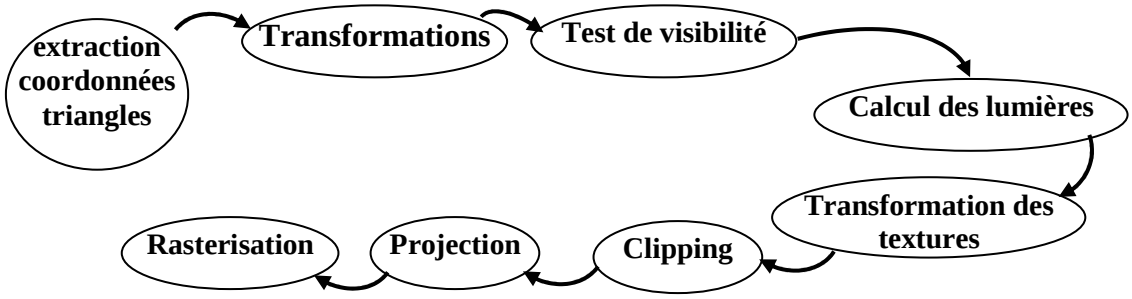

Figure 52. Synthèse d'images 3D

– Extraction des coordonnées des triangles : Les éléments d'entrée de ce pipeline sont des triangles qui sont plus pratiques que les quadrilatères ou autres polygones pour les calculs. En effet, ils ne peuvent être ni vrillés (dont les sommets ne sont pas coplanaires) ni concaves. La plupart des moteurs 3D effectuent une triangulation des différentes faces avant de les envoyer au pipeline graphique. La figure 53 montre deux exemples de cette transformation. Les coordonnées des différents sommets des triangles (dans le repère propre de l'objet) ainsi que la définition des arêtes des triangles sont sauvegardées dans un fichier. Il n'est pas nécessaire que les triangles soient égaux comme le montre la figure 53. L'extraction des coordonnées permet de charger les coordonnées des sommets des triangles à partir de la base de données décrivant les objets pour les appliquer aux étapes suivantes de rendu 3D.

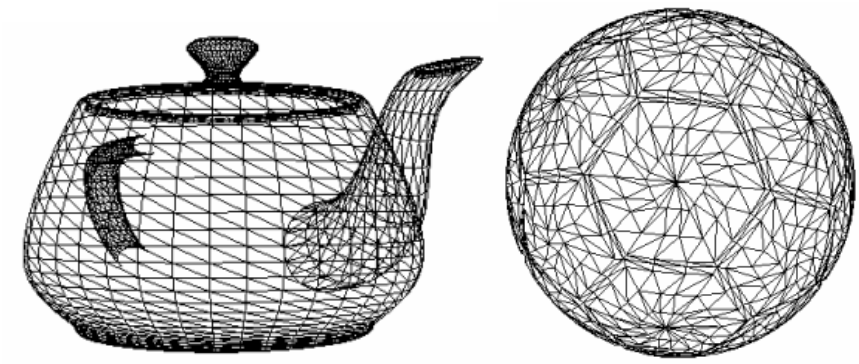

Figure 53. Objets transformés en un ensemble de triangles

– Transformation géométrique : Les transformations permettent de convertir les coordonnées locales de l'objet 3D dans le repère global de la scène (appelé repère du monde) puis dans le repère de la caméra (observateur). Elles permettent également d'animer l'objet 3D.
– Test de visibilité : Il permet de déterminer les faces de l'objet 3D visibles par l'observateur.

- Calcul des lumières : Le calcul des lumières permet de déterminer l'intensité de la couleur en un point.
- Transformations des textures : La texture est une image plaquée sur la surface de l'objet 3D. Cette étape permet de transformer les textures avant qu'elles ne soient appliquées au triangle dans l'étape de « rasterisation ». Si aucune texture ne doit être appliquée au triangle, cette étape est omise.
- Clipping (fenêtrage) : Cette étape consiste à éliminer les triangles qui ne font pas partie du volume de vue (situés hors du champ de vision de l'observateur).
- Projection : C'est la transformation de l'objet 3D représenté dans le système de coordonnées du monde, en coordonnées de l'écran.
- Rasterisation : La « rasterisation » est l'étape transformant les formes géométriques 3D en pixels sur l'écran, tout en donnant un aspect réel à l'objet 3D en question.

Nous considérons le cas de rendu 3D d'objets entièrement visibles sur l'écran sans texture. Cette application est formée donc par 6 étapes (puisque nous éliminons l'étape de transformations des textures et l'étape de clipping). L'application est codée en langage C. Nous analysons ce code pour pouvoir identifier des modules matériels spécifiques à cette application.

2.1. Graphes de tâches de l'application 3D

L'application de traitement d'images 3D est décomposée en 15 tâches réalisant chacune une fonctionnalité simple comme le montre la figure 54 [Lou04].

Une étape de la synthèse 3D (Cf. la figure 52) peut correspondre à une ou plusieurs tâches selon sa complexité. Par exemple l'étape « extraction coordonnées triangles » correspond à la tâche « LoadAS ». Par contre, l'étape « transformation » correspond aux tâches « Changement d'échelle », « Translation », « Rotation » et « Calc_new_coord ». Le tableau 1 résume l'ensemble de ces tâches.

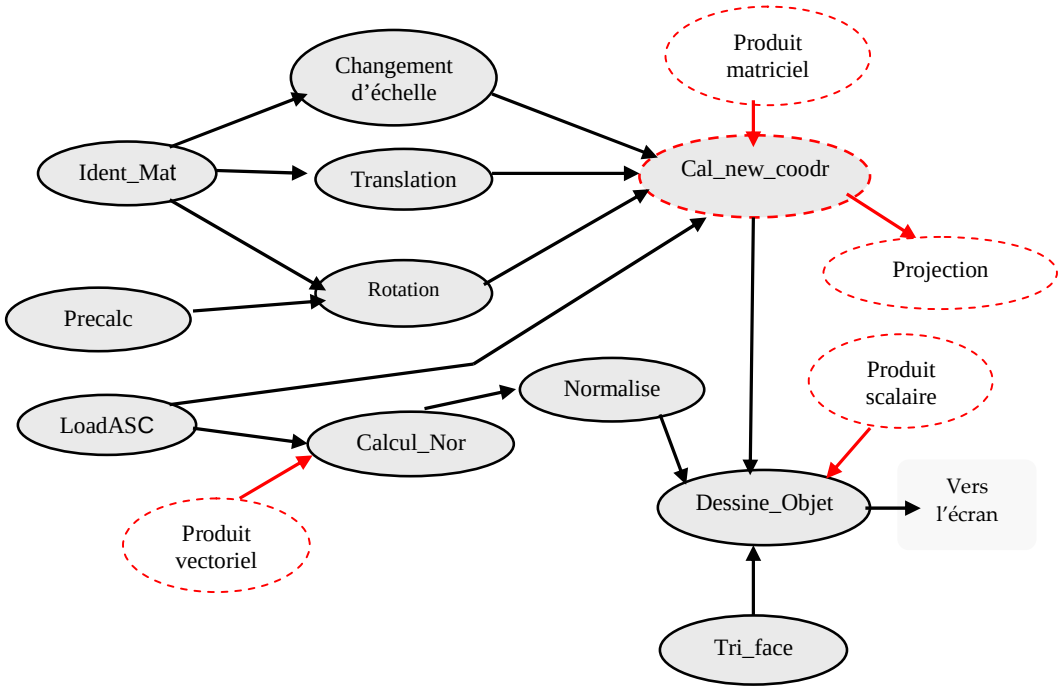

Figure 54. Graphe de tâches de l'application synthèse 3D

Le « profiling » de cette application nous a permis de sélectionner les fonctions qui sont le plus souvent appelées. Il s'agit des fonctions suivantes : « produit scalaire», « produit vectoriel », « produit matriciel », « la projection 2D » et la « Calc_new_coord», (Cf. fonctions en pointillés dans la figure 54).

Tableau 1. Description des tâches de l'application synthèse 3D

Ident_Mat	construit une matrice identité
Precalc	Calcule le tableau de sinus/cosinus
LoadAS	charge les coordonnées des sommets d'un objet 3D
Produit vectoriel	Réalise le produit vectoriel de deux vecteurs
Echelle	Calcule la matrice de changement d'échelle
Translation	Calcule la matrice de translation
Rotation	Calcule la matrice de rotation
Calcul_Nor	calcule les normales de face pour chaque polygone
Normalise	Permet de normaliser un vecteur
Produit matriciel	Calcule le Produit matriciel
Calc_new_ coord	Calcule les nouvelles coordonnées après transformation géométrique
Dessine_Objet	Dessine l'objet sur l'écran.
Projection	Permet la transformation des coordonnées 3D sur l'écran 2D
Produit scalaire	Calcule le Produit scalaire
Tri_face	Permet de trier les faces pour n'afficher que celles visibles

L'accélération de l'application synthèse 3D passe donc par l'accélération de ces fonctions. Nous détaillons dans la suite brièvement ces fonctions.

2.2. Accélérateur matériel

L'objet 3D est décrit en utilisant les coordonnées homogènes et qui sont utilisées afin d'unifier le traitement des transformations géométriques d'une scène et de les exprimer sous forme matricielle dans une seule matrice. Pour cela, une quatrième coordonnée « w » est ajoutée aux coordonnées cartésiennes (x,y,z) d'un point M de l'objet 3D.

La fonction « transformation géométrique » permet d'appliquer à un objet 3D les transformations suivantes :

– *La Translation*

La translation de vecteur $\vec{T}$ (tx, ty, tz) appliquée au point M donne un point M'(x',y',z',w') selon l'équation :

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

– *Le changement d'échelle*

Le changement d'échelle de vecteur $\vec{S}$ (Sx, Sy, Sz) appliquée au point M donne un point M'(x', y',z', w') selon l'équation :

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

– *Rotation*

La rotation d'angle « θx » appliquée au point M donne un point M'(x',y',z',w') selon l'équation :

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta_x) & -\sin(\theta_x) & 0 \\ 0 & \sin(\theta_x) & \cos(\theta_x) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Le produit scalaire peut être considéré comme un produit matriciel d'un vecteur ligne par un vecteur colonne.

La projection est la transformation qui permet de donner la position du point image sur le plan à partir d'un point dans l'espace caractérisé par ses coordonnées (x,y,z).

Nous remarquons que quatre des cinq fonctions critiques peuvent s'exprimer par un produit matriciel sous divers formats (matrice*vecteur, vecteur*vecteur ou matrice*matrice). C'est pourquoi nous avons développé un IP dédié pour le cas général de produit de deux matrices 4*4. Il pourra être facilement modifiable pour traiter les autres types de produit matriciel (matrice*vecteur ou vecteur *vecteur). Chaque fois qu'une fonction a besoin d'un calcul matriciel, elle fait appel à cet IP ce qui permet d'accélérer le rendu d'un objet 3D.

GAUT est utilisé pour générer l'architecture matérielle correspondante de l'IP accélérateur et ainsi de déterminer le comportement aux entrées/sorties des données.

2.3. Synthèse sous GAUT de l'IP « produit matriciel »

Dans cette section, nous présentons plusieurs architectures de l'IP « produit matriciel » obtenues en modifiant les paramètres de synthèse sous GAUT. Ces architectures serviront à tester notre interface. Pour cela, nous avons considéré les contraintes de synthèse suivantes :

- Latence : c'est le temps d'exécution de l'algorithme « produit matriciel » (chemin critique) (voir Tableau 2). Elle correspond à la plage temporelle définie dans le fichier source (ficher.src ou fichier.c) et permet de limiter le temps sur lequel se manifeste l'algorithme architectural. En effet, GAUT n'accepte pas les boucles infinies.
- Cadence : elle correspond à la plage temporelle sur laquelle l'ensemble des données nécessaires à une itération (répétition) de l'algorithme arrive (le taux d'arrivée des ensembles d'entrée des données). Elle doit être un multiple de la période de l'horloge du circuit synthétisé.

Tableau 2. Exemples d'architectures GAUT de l'IP « produit de 2 matrices 4x4 »

	Cas1	Cas 2	Cas 3	Cas 4	Cas 5	Cas6
Latence (ns)	3000	2000	1000	800	500	200
cadence (ns)	3000	2000	1000	800	500	200
Cible technologique (compatible Altera)	EPXA1F4 84C3-16b	EPXA1F4 84C3-16b	EPXA1F4 84C3-16b	EPXA1F4 84C3-16b	EPXA1F4 84C3-16b	EPXA1F4 84C3-16b
Nb_bus*	5	5	14	24	32	46
Nb_bus utiles**	4	4	9	15	28	37

* : nombre de bus total à l'interface de l'unité de traitement de l'IP
** : nombre de bus servant pour le transfert des données aux entrées/sorties de l'IP

La figure 55 présente un exemple d'architecture générée par GAUT. Si la latence est supérieure à la cadence, une architecture de traitement pipeline doit être établie. GAUT génère un « fichier.mem » qui regroupe ces informations ainsi que les contraintes spatio-temporelles sur les données à l'interface de l'architecture de l'IP.

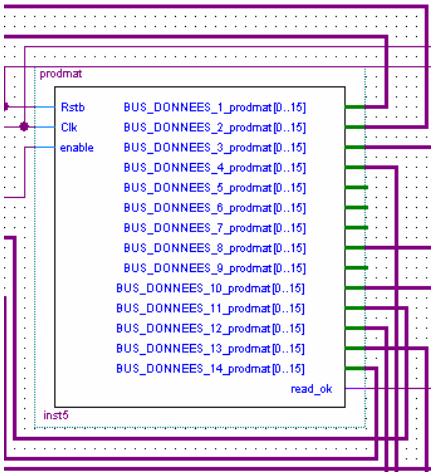

Figure 55. Exemple d'architecture d'IP GAUT (produit matriciel cas 3)

Il suffit pour cela de sélectionner le fichier de caractérisation de l'architecture IP cible pour générer l'interface correspondante à l'aide de l'outil GIC.

Le fichier « mem.txt », image du « fichier.mem », généré à la suite de la synthèse haut niveau avec l'outil GAUT de l'IP accélérateur permet de gagner le temps. En effet, l'importation des contraintes d'ordonnancement à l'interface de l'IP se fait à partir de ce fichier.

Nous présentons dans la section suivante un scénario d'intégration de l'IP « produit matriciel » dans un système sur lequel tourne l'application « rendu 3D ».

2.3. Scénario d'intégration

Nous disposons d'un code C permettant le rendu 3D d'un objet sans texture. Ce code permet de charger un fichier décrivant les sommets et les triangles de l'objet 3D et de le visualiser sur l'écran [Dor05].

L'application « rendu 3D » (représentée par le graphe de tâche de la figure 54) est partitionnée en deux parties :

- Partie logicielle exécutée par le processeur NIOS : elle comprend les différentes fonctions non exécutées par l'IP « produit matriciel ».
- Partie matérielle : formée par l'IP « produit matriciel ». A chaque fois qu'il y a dans la partie logicielle un calcul matriciel, c'est l'IP qui est appelée pour effectuer

cette tâche. Comme nous l'avons détaillé avant, cet IP sera utilisé par les tâches « Translation », « changement d'échelle », « rotation » et « produit scalaire ».

Pour les deux contextes d'intégration (simulation et synthèse), nous avons recours aux étapes suivantes :

– *1°.étape* : Réaliser la boucle suivante :

For (i=1; i<nbre de triangles, i++)

{

Extraire les triangles à partir de fichier.ASC

Mapper dans la mémoire de la plateforme

}

– *2°.étape* : Envoyer les triangles de la sphère avec les transformations correspondantes à l'aide du driver à l'accélérateur :
– envoi des données à FIFO_IN
– *3°. Etape* : Envoyer des données sérialisées à partir de la FIFO_OUT à la mémoire
– mapper la sphère résultante dans la mémoire
– récupérer le résultat à l'aide du driver pour continuer l'application

Par ailleurs, l'IP « produit matriciel » nécessite d'extraire les données :

– ligne par ligne de la matrice correspondant à la matrice sommet
– colonne par colonne à partir de la matrice correspondant à la matrice de transformation.

La sérialisation des données lors de l'envoi se fait selon le modèle suivant :

[ligne1(sommet1), colonne1(transformation1), ligne2(sommet1), colonne2(transformation1), ligne3(sommet1), colonne3(transformatio1), ligne4(sommet1), colonne4(transformation1), ligne1(sommet2), colonne1(transformation2), ligne2(sommet2), colonne2(transformation2), ligne3(sommet2), colonne3(transformation2), ligne4(sommet2), colonne4(transformation2) , ligne1(sommet3), colonne1(transformation3), ligne2(sommet3), colonne2(transformation3), ligne3(sommet3), colonne3(transformation3), ligne4(sommet3), colonne4(transformation3)]

Ce motif est à répéter pour l'envoi de chaque triangle de la sphère. Cette description textuelle décrit le pseudo ordre à vérifier par le système.

En revanche, l'ordre de consommation et de production à l'interface de l'IP est aléatoire. Un exemple de répartition spatiale de cet ordre est décrit par la figure 56 pour le cas 3. Le cas 3 dans le tableau 2 présente une architecture d'IP produit matricielle généré par

GAUT possédant 9 bus à son interface d'entrée sortie. La figure 56 montre le comportement à l'interface de l'IP, des coefficients des matrices d'entrée et de la matrice de sortie sur les bus ainsi que les délais de temps qui séparent ces coefficients les unes des autres.

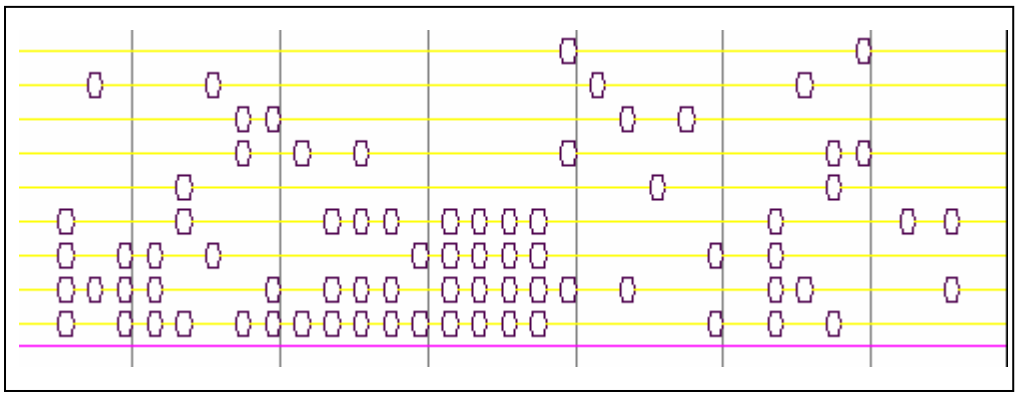

Figure 56. Ordonnancement des données à l'entrée de l'IP (cas 3)

L'application de l'approche d'intégration avec ces considérations est validée tout d'abord pour un contexte de simulation, puis pour un contexte de synthèse.

3. Génération d'interface pour la Simulation

Lors des phases de spécification et d'exploration d'architecture, un certain nombre d'hypothèses sont retenues pour l'assemblage de composants virtuels. Ces derniers sont évalués par simulations puis affinés jusqu'à ce qu'une solution satisfaisante soit retenue. Il est donc important de disposer d'une plateforme de simulation ayant les caractéristiques suivantes :

- qu'elle dispose d'une bibliothèque de modèles de simulation de composants virtuels (IPs) afin d'évaluer un grand nombre de solutions au prix d'efforts de développement réduits et que l'interfaçage des différents composants de la bibliothèque soit simplifié.
- que les modèles de simulation utilisés soient précis pour garantir la fiabilité des mesures de performances. Afin de faciliter le développement du système complet, l'utilisation de plateforme de simulation cycle près bit près devient indispensable.
- que les modèles de simulation soient rapides pour explorer le maximum d'hypothèses différentes sur un grand nombre de scénarios d'utilisation du produit visé.

Pour illustrer la génération de l'interface pour la simulation, nous avons adopté la plateforme SoCLiB [SoCL]. SoCLiB propose une plateforme de simulation pour la

conception des SoCs conformément au protocole VCI. Nous présentons dans la suite cette plateforme de simulation.

3.1. Plateforme de simulation SoCLiB

SoCLiB est une plateforme de prototypage de SoC qui a la particularité de posséder une bibliothèque de composants décrits en SystemC conformément au protocole VCI. Cette particularité permet de simuler des systèmes à différents niveaux d'abstraction, d'implémenter et d'intégrer facilement de nouveaux IPs. Nous présentons dans la suite le protocole VCI adopté dans cette plateforme.

3.1.1. Protocole VCI

La plateforme SoCLiB utilise le protocole d'interface VCI, normalisé par le consortium VSIA, afin de garantir l'interopérabilité entre les différents composants du système. L'objectif de ce protocole est de séparer clairement -au niveau du matériel - la fonction de calcul de la fonction de communication. Il permet de :

- Réutiliser des composants matériels
- Supporter les architectures multi-processeurs
- Conserver le paradigme de communication « espace d'adressage partagé » : un maître désigne sa cible par les bits de poids fort de l'adresse et une case mémoire particulière par les bits de poids faible.
- Réutiliser des composants logiciels
- Fournir à chaque maître l'illusion qu'il dispose d'un canal de communication point à point avec chaque cible.
- Simplifier le protocole d'accès à l'interconnecte (bus, réseaux de communication, NoC, etc).

VCI est construit autour d'un espace adressable partagé. Les initiateurs émettent des requêtes de lecture ou d'écriture. Une requête peut contenir une seule adresse (transaction simple) ou plusieurs adresses (transaction rafale). Pour SoCLiB, le protocole VCI est implémenté dans sa version PVCI (VCI périphérique) et BVCI (VCI Basique). Afin d'utiliser un IP dans cette plateforme, il est nécessaire de l'adapter à ce protocole.

3.1.2. Modélisation des composants sous SoCLiB

Dans la logique SoCLiB, une écriture spécifique des modules d'une architecture est nécessaire. Conceptuellement, un module est décrit comme un unique automate : il comprend

un nombre quelconque de registres, et peut être constitué de plusieurs petits automates communiquant entre eux, à l'intérieur du même module. Chaque module doit être décrit comme un automate d'état synchrone (de type Moore ou Mealy) en utilisant le langage SystemC au niveau CABA. Ce niveau d'abstraction permet de décrire une interface matérielle sans décrire l'architecture matérielle du composant.

Cette spécification permet de simuler un système plus rapidement que le modèle RTL et favorise aussi bien la simulation et la synthèse d'interface. D'ailleurs, elle garantit plus de sûreté dans la conception d'architecture d'interface pour les SoCs. En fait, tous les sous modules de l'interface sont spécifiés au niveau CABA. Ceci peut être assez difficile en utilisant ce modèle de codage. Cependant, les modèles de simulation générés au niveau CABA sont sémantiquement proches de la description habituelle de RTL.

L'architecture d'implantation peut être alors masquée au profit d'une simulation rapide avec des interfaces réelles.

3.2. Expérimentation de l'outil GIC pour la simulation

L'expérimentation de l'outil GIC pour la simulation exige :

- du côté de l'architecture cible :
- le partitionnement et l'ordonnancement des tâches de l'application cible pour identifier le comportement des données du côté du système.
- du côté de l'IP cible :
- Caractérisation à l'interface du comportement spatio-temporel des données aux entrées/sorties au cycle prêt et au bit prêt.
- L'adaptation entre le protocole VCI et l'interface générique revient à spécifier les adaptateurs « VCI-FIFO » et « FIFO-VCI » (adaptateurs VCI↔FIFO dans la figure 57).

La génération de l'interface repose sur une bibliothèque de modules SystemC décrits pour une intégration dans la plateforme SoCLiB.

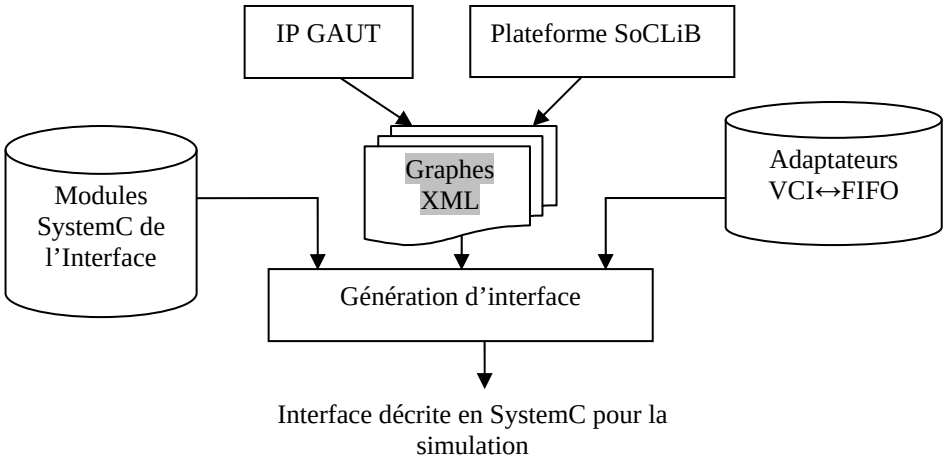

Figure 57. Application de l'approche pour la simulation de l'interface

Nous décrivons dans la suite les différents modules développés pour la génération de l'interface de communication pour la simulation.

3.2.1. Bibliothèque de composants

Les éléments de la bibliothèque ont été implémentés à l'aide de SystemC [Sys00]. Nous allons introduire brièvement quelques concepts de base du langage SystemC. En fait, le système est modélisé comme étant une collection de modules qui contiennent des processus, et qui communiquent en utilisant des ports, des canaux, et des interfaces. Les processus définissent le comportement des modules. Une interface définit un ensemble de méthodes, mais ne les implémente pas. Un canal implémente un ou plusieurs méthodes d'une interface [Sys01]. Un port permet à un module, et désormais ses processus, d'accéder à un canal. Le port est aussi définit en terme d'interface, c'est-à-dire qu'il ne peut être utilisé qu'avec les canaux qui implémentent cette interface. Le concept d'interface permet l'utilisation d'un schéma de communication appelé en anglais (*Interface Method Call – IMC*), qui s'applique à un processus appelant une méthode de l'interface implémentée au niveau d'un canal.

Par ailleurs, l'aspect conception matérielle et l'aspect méta-programmation sont pris en compte avec SystemC [SYS] : une conception haut niveau utilise des modèles « templates » pour la conception générique. En effet, les méthodologies basées sur C++, tels SystemC, gèrent la complexité et la diversité de conception des SoCs en introduisant l'Orienté Objet [Gaj00]. Ce principe permet de séparer l'interface de l'implémentation, et favorise la réutilisation à un niveau d'abstraction élevé.

Nous distinguons ici les adaptateurs au protocole VCI et les modules constituants l'interface de communication générique.

3.2.2. Adaptateurs (VCI-FIFO, FIFO-VCI)

Les modules définis dans SoCLiB communiquent à l'aide du protocole VCI. Afin de relier un composant aux autres composants de la plate-forme, nous avons créé deux adaptateurs de protocoles à savoir un adaptateur pour chaque sens de communication : VCI vers FIFO (VCI_TO_FIFO dans la figure 52), FIFO vers VCI (FIFO_TO_VCI dans la figure 58).

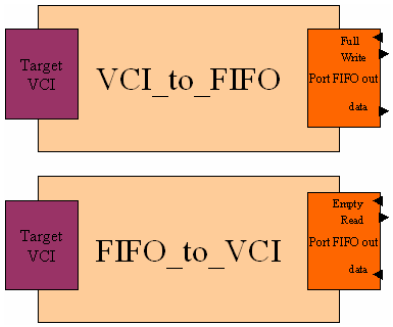

Figure 58. Représentation des adaptateurs VCI-FIFO

Les deux adaptateurs considèrent le composant à intégrer comme un IP cible (Target VCI).

3.2.3. Spécification SystemC des modules de l'interface

Dans le chapitre 3, nous avons décrit deux conceptions de l'interface, une dédiée pour le cas multiprocesseur et l'autre pour le cas monoprocesseur. Pour chacune de ces versions, nous avons implémenté différents modules au niveau CABA SystemC. Chaque module correspond à une fonctionnalité bien déterminée. Les modules seront distingués suivant leur appartenance, soit à la chaîne d'entrée soit à la chaîne de sortie, par leurs paramètres « template ».

La chaîne de communication d'entrée présente les modules suivants :

- FIFO_IN_H
  - template < unsigned int SIZE, unsigned int BITWIDTH_IN, unsigned int BITWIDTH_OUT >
- INPUT_CONTROLER_H
  - template < int BUSWIDTH , int NBVAR_INPUT>
- FIFO_ENABLE_H
  - template < int BITWIDTH_OUT, int BITWIDTH_IN, int NBVAR_INPUT, int NBVAR_OUTPUT >

La chaîne de communication de sortie présente les modules suivants :

- OUTPUT_CONTROLER_H
  - template < int BUSWIDTH , int NBVAR_OUTPUT>
- FIFO_OUT_H
  - template < unsigned int SIZE, unsigned int BITWIDTH_IN, unsigned int BITWIDTH_OUT >

Nous définissons :

- BUSWIDTH=32 bits : taille des données véhiculées par l'interconnecte VCI dans SoCLiB.
- NBVAR implémente le nb_Bus_utiles pour les IPs synthétisés avec GAUT.

Des signaux de synchronisation sont considérés pour assurer le bon fonctionnement de l'ensemble des sous modules constituants l'interface de communication.

3.3. Fonctionnalités principales du GIC pour la simulation

Nous distinguons deux principales fonctionnalités dans le GIC pour l'automatisation de la génération d'interface de communication pour la simulation sous SoCLiB :

- La génération du driver.
- La génération du fichier d'interconnexion qui assure :
  - La paramètrisation des variables génériques (Template) à partir des informations extraites des graphes.
  - L'instanciation des modules à partir de la bibliothèque

L'interconnexion des modules de l'interface est assurée par exemple selon la figure 59.

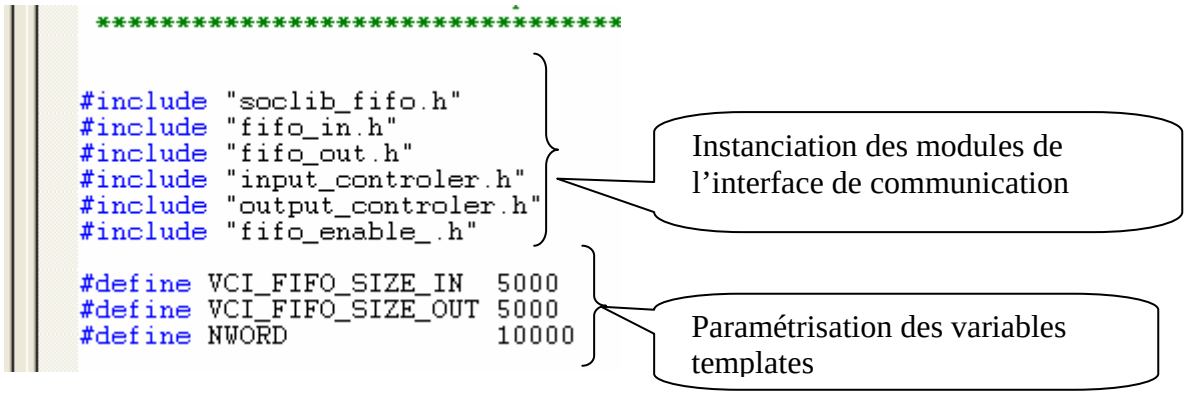

Figure 59. Instanciation des modules et configuration des paramètres génériques

3.4. Simulation et Résultats

Afin de tester l'approche d'intégration de l'IP accélérateur dans un contexte de simulation, l'architecture de la plateforme SoCLiB considérée (Cf. figure 60 pour le cas du test de la conception 1) est composée de :

- 1 MIPS R3000 et 1 cache associé (architecture RISC 32bits) : à ce processeur est associé le compilateur GCC-MIPS, pour le test de la conception 1.
- une mémoire cache associée au processeur : Cette mémoire est paramétrable de 512 octets à 8ko, elle est associative par ensemble.
- une mémoire ROM : contenant le programme à exécuter et les constantes
- une mémoire RAM : contenant les données,
- un système d'interconnexions VCI (crossbar : generic network qui est une sorte de routeur) : Ce n'est pas un bus au sens bus de données, bus d'adresse, mais un réseau qui permet la connexion de tous les composants du système qui répondent à la norme VCI,
- une table de segmentation : La table de segmentation est le cœur de l'architecture. C'est une table de correspondance entre les adresses des différents composants de l'architecture ainsi que les zones ou les espaces mémoires physiques réservés aux données ou aux instructions. Il permet donc de spécifier le « mapping » mémoire des différents composants du système.

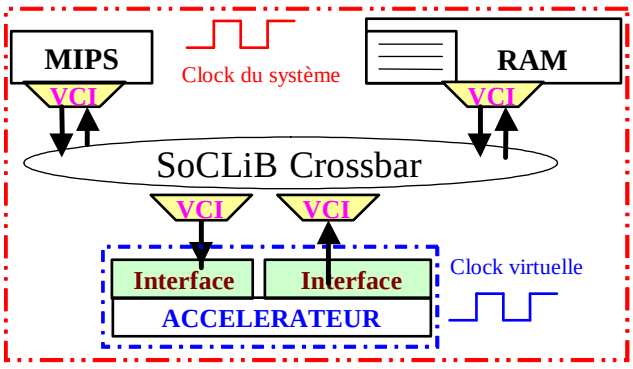

Figure 60. Représentation de l'architecture de l'expérience

- Pour le cas du test de la conception 2, 2 MIPS R3000 sont considérés et un « timer » est utilisé pour la synchronisation des deux processeurs. Le premier processeur génère la matrice de la transformation, le deuxième génère la matrice de la sphère et récupère la sphère transformée.

Les résultats de simulation obtenus pour l'accélération de l'étape de transformation géométrique sont donnés dans ce qui suit. Les simulations ont été effectuées et validées avec un Pentium M Centrino (1.5 GHz, 512Mo de RAM) sous l'environnement Linux comme système d'exploitation, le simulateur SystemC-2.0.1 et le compilateur GCC 3.3.1. L'architecture SoCLiB est au niveau macro architecture et l'interface de l'IP est traitée au niveau microarchitecture CABA. La partie logicielle « driver » et la partie matérielle « wrapper » ou adaptateur de l'interface de communication sont générées à partir de la même description du comportement du transfert des données. Le système entier est co-simulé.

Comme nous l'avons déjà dit, nous faisons varier le nombre de triangles constituant la sphère. Le nombre de triangles influe sur la qualité de la sphère visualisée en 3D sur un écran 2D. L'expérience consiste à effectuer une séquence d'animation de 100 rotations d'une sphère autour des trois axes.

Pour comparer l'influence des conceptions proposées de l'interface, nous considérons les temps de simulation pour différentes sphères (objet d'entrée du pipeline 3D). La taille d'un objet « sphère » est exprimée en nombre de triangles. Un triangle possède trois sommets. Chaque sommet est caractérisé par 4 coordonnées.

Tableau 3. Performances de la simulation complète du SoC

Taille de la sphère (Nb triangles)	Temps de simulation (première conception) (secondes)	Temps de simulation (deuxième conception) (secondes)
62	2,108	2,2
146	4,21	5,11
191	6	6,53
266	10,16	10,94
366	13,4	14,5

Nous rappelons que la première conception est dédiée au cas monoprocesseur. Dans cette conception, le transfert des données entre l'IP et le reste du système suit un ordre imposé par l'IP et déduit à partir de son fichier de caractérisation issu de GAUT. La deuxième conception est dédiée au cas multiprocesseur. Les données peuvent provenir de deux ou plusieurs processeurs, elles sont véhiculées avec l'adresse du processeur expéditeur (initiateur).

Le tableau 3 donne les temps de simulation pour les deux conceptions. Nous remarquons que les temps de simulation relatifs à la deuxième conception sont plus élevés que ceux de la première conception. En effet, dans le cas de la deuxième conception, un décodage d'adresse (afin de reconnaître l'initiateur de chaque donnée) et un contrôle plus complexe au niveau des modules de contrôle en entrée et en sortie de l'interface de communication et au niveau des modules FIFO_IN et FIFO_OUT sont necessaires. C'est ce

qui explique pourquoi dans le cas de la deuxième conception, les temps de simulation sont plus lents. La première conception réduit le temps de simulation. Cependant, elle ne peut pas viser le contexte MPSoC sans un arbitre d'ordonnancement [Zit01].

Dans cette approche, la communication a été opérée directement par le processeur (architecture monoprocesseur) (figure 61). C'est la méthode la plus simple, mais elle génère des redondances de communication (de la mémoire au processeur et du processeur à l'IP). Cette redondance inclut des surcoûts temporels dus à l'accès mémoire et à la latence de l'interconnecte pour chaque mot transféré par le processeur vers l'accélérateur ou de l'accélérateur retourné à la mémoire.

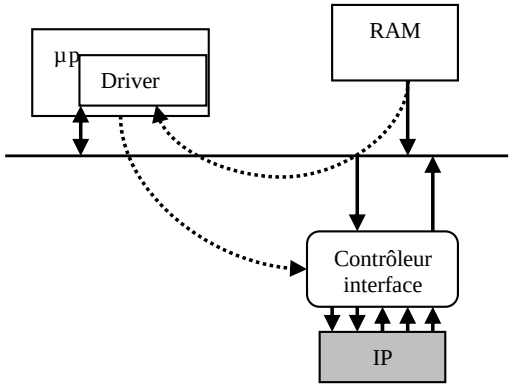

Figure 61. Mode d'utilisation de l'interface dans SoCLiB

Pour utiliser plus efficacement le bus, un DMA peut être utilisé favorisant des transferts en mode continu entre mémoire et accélérateur matériel. Ce DMA doit pouvoir gérer plusieurs transactions successives où des communications en « burst » peuvent être exécutées.

3.5. Conclusion

Cette partie a servi pour la simulation et la vérification des fonctionnalités de l'architecture générique de l'interface. La génération de l'interface et la vérification de son bon fonctionnement a été testée en considérant plusieurs architectures d'IPs : le même IP accélérateur peut être synthétisé en considérant différentes contraintes pour la génération de son architecture RTL. En effet le propre d'un outil SHN, plus particulièrement l'outil GAUT, est la génération de plusieurs architectures de la même description d'entrée en considérant des contraintes d'optimisation convenables pour le contexte d'application ou d'intégration (cadence, taille des données, parallélisme). L'interfaçage automatique obtenu grâce à notre outil permet donc le prototypage rapide de systèmes.

La spécification au niveau CABA tend à simuler plus rapidement que le modèle RTL et favorise la synthèse systématique de l'interface. De plus, elle garantit plus de sûreté dans la

génération de l'architecture de l'interface dans un contexte de SoC/MPSoC. En fait, les sous modules constituants l'interface sont générés avec la représentation CABA. Ce modèle d'exécution génère des modules de simulation sémantiquement comparables à une exécution RTL**.**

4. Génération d'interface pour la Synthèse

L'interface possède la même architecture pour la simulation ou pour la synthèse. La seule différence est le langage de programmation. Dans le premier cas, c'est le langage SystemC qui est retenu. Dans ce second cas, le langage VHDL est retenu.

Afin d'illustrer la génération de l'interface pour la synthèse, nous avons adopté la plateforme Altera [ALT].

L'expérimentation de l'outil GIC pour la synthèse exige :

- du côté de l'architecture cible :
- Le partitionnement et l'ordonnancement des tâches de l'application cible pour identifier le comportement des données du côté du système.
- du côté de l'IP cible :
- La caractérisation à l'interface du comportement des entrées/sorties au cycle près et au bit près.

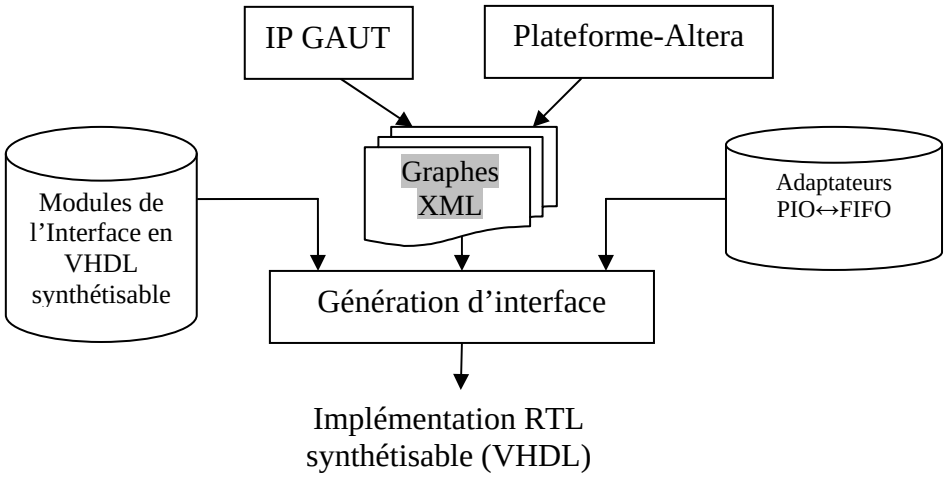

Figure 62. Application de l'approche d'intégration pour la synthèse d'interface

La figure 62 montre l'application de l'intégration pour la synthèse en utilisant l'environnement Altera. L'interface est décrite à l'aide d'un ensemble de sous modules qui modélisent son fonctionnement pour le chemin d'entrée ainsi que pour le chemin de sortie. Une librairie de classes paramétrables est à l'origine de la construction de la structure de

l'interface. Dans ce cas, l'instanciation des modules utiles pour l'intégration d'un IP dans un SoC à partir de la librairie est faite à travers leur configuration. Pour modéliser les données à l'interface, un fichier de configuration de l'IP est généré à partir de l'outil GAUT. A partir de ce fichier, l'outil GIC produit des fichiers de configuration sous format XML enregistrés dans un dossier appelé « Graphes ». Ces fichiers sont utilisés par la suite pour instancier les modules VHDL nécessaires spécifiques à l'interfaçage de l'IP considéré (Cf. figure 62).

Vu le caractère non objet (langage procédurale, expliquer) du langage VHDL d'implémentation, la construction générique des modules constituants l'interface (en VHDL synthétisable) repose sur une première manipulation ciblant un exemple d'étude où les paramètres et les modules sont fixés.

La bibliothèque adoptée est constituée par des sous modules implémentant l'architecture de l'interface générique en VHDL synthétisable en considérant comme technologie cible les FPGA de la société Altera.

4.1. Environnement de validation : Plateforme Altera

L'environnement de validation choisi est la plateforme EXCALIBUR d'ALTERA. Elle comporte la carte de développement avec le processeur embarqué NIOS [Alt04] et les logiciels de développement associés QUARTUS, SOPC Builder [Alt05] et le compilateur C Gnu de Cygnus. Cet environnement est caractérisé :

- par sa convivialité et sa relative facilité d'utilisation.
- par l'architecture personnalisable (customizable) de son processeur associé NIOS : il peut être facilement adapté aux exigences de l'application grâce à la possibilité d'ajout d'instructions spécifiques (jusqu'à 256 instructions pour le NIOS). Il offre également les moyens nécessaires pour faciliter l'ajout d'accélérateurs via son bus d'extension.

Nous décrivons dans la suite les différents modules de l'interface. Nous montrons ensuite comment notre interface est adaptée au bus Avalon.

4.2. Modules VHDL de l'interface

Selon les deux conceptions définissant l'architecture générique de l'interface de communication, nous avons implémenté différents modules. Chaque module correspond à une fonctionnalité bien déterminée. Tous les modules, sont implémentés suivant les paramètres du « produit de deux matrices 4x4 ». Les modules seront distingués suivant leurs appartenances,

soit à la chaîne d'entrée ou à la chaîne de sortie. Ces modules génériques sont par la suite instanciés et configurés par le GIC.

La chaîne de communication d'entrée contient les modules suivants :

- FIFO_SANS_ADRESSE_IN (première conception)
- FIFO_AVEC_ADRESSE_IN (deuxième conception)
- CTRL_SANS_ADRESSE_IN (première conception)
- CTRL_AVEC_ADRESSE_IN (deuxième conception)
- FIFO_SANS_ADRESSE_FE (première et deuxième conception)
- FIFO_ENABLE/ENABLE (première et deuxième conception)

La chaîne de communication de sortie contient les modules suivants :

- FIFO_SANS_ADRESSE_FE (première et deuxième conception)
- CTRL_SANS_ADRESSE_OUT (première conception)
- CTRL_AVEC_ADRESSE_OUT (deuxième conception)
- FIFO_SANS_ADRESSE_OUT (première conception)
- FIFO_AVEC_ADRESSE_OUT (deuxième conception)
- FIFO_ENABLE/ ENABLE (première et deuxième conceptions)

Le fait d'avoir ces modules séparés ne garantie pas une interconnexion automatique et fiable. Nous avons donc pris en compte les retards dans la propagation des données qui sont dus aux temps de traitements de chaque module et qui varient de l'un à l'autre. Pour résoudre ce « problème de synchronisation », nous avons recours à l'assemblage des modules pour garantir une interconnexion automatique des modules de l'interface. L'interconnexion est générée avec l'outil GIC.

Le problème de synchronisation est dû à la différence du temps de traitement entre les différents modules. La solution est l'ajout de signaux de contrôle. Les principaux signaux de contrôle sont :

- Read_E : ce signal de type std_logic, indique au module suivant que les données sont prêtes et qu'elles peuvent être lues. Par exemple, dans les FIFOs, le signal Read_E indique que les FIFOs contiennent des données.
- Write_E : c'est un signal de type std_logic. Il indique que le module peut recevoir les données. Par exemple, dans le cas d'un module FIFO, le signal Write_E indique que la FIFO n'est pas pleine.
- Write_OK : ce type de signal est utilisé pour synchroniser l'IP et les FIFO_SANS_ADRESSE_FE. Le rôle de ce signal est d'activer les FIFO_SANS_ADRESSE_FE selon les paramètres de l'IP.

– Enable : ce signal active l'IP après l'activation des FIFO_SANS_ADRESSE_FE pour que l'IP reçoive les données.
– ChipSelect : ce signal est utilisé pour activer le module FIFO_ENABLE.

L'assemblage est fait en considérant la première conception de l'architecture de l'interface pour la chaîne d'entrée et pour la chaîne de sortie dans la figure 63. L'architecture IP synthétisée (cas 3) avec GAUT présente 9 bus utiles, deux ports d'entrées et un port de sortie.

Cette validation permet d'assurer la synchronisation au niveau de la simulation entre la chaîne de communication d'entrée et celle de sortie de l'interface de communication. Une telle implémentation est ciblée et sa modification est en général manuelle. Le GIC permet d'automatiser la génération du VHDL synthétisable.

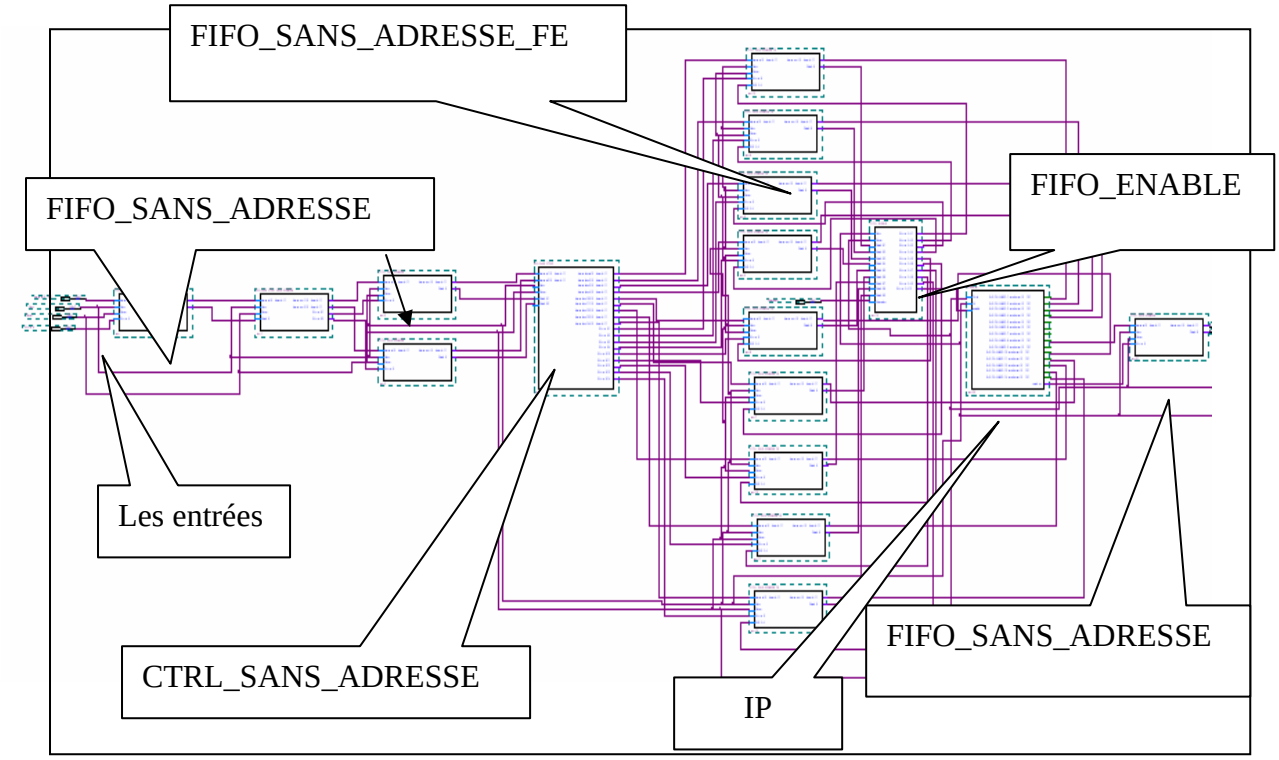

Figure 63. Interface « produit de deux matrices 4x4 cas 3 »

4.3. Adaptateurs (Avalon-FIFO, FIFO-Avalon)

L'ensemble {interface générique, IP accélérateur} est adapté au bus Avalon à travers un PIO (Peripheral Input Output) avec le protocole FIFO. Un PIO est une interface fournie dans l'environnement QUARTUS d'ALTERA. Il permet la communication logicielle/matérielle entre le code (partie logicielle) tournant sur le processeur NIOS et les

modules matériels définis par l'utilisateur. Ces modules peuvent être situés sur le même chip que le processeur ou à l'extérieur de celui-ci [Alt05].

La procédure de l'interconnexion à travers les PIO se résume par l'ajout d'un nombre nécessaire de PIO pour relier les composants au processeur tout en définissant la taille du bus de données et les types d'interruptions matérielles pour les entrées afin de pouvoir récupérer le résultat.

4.4. Fonctionnalités principales du GIC pour la synthèse

Pour généraliser l'application et la synthèse de l'interface pour des contextes différents, l'outil se base sur un ensemble de fichiers de configuration constitués de modules en VHDL. Ces modules sont simulables (le logiciel utilisé pour la simulation est ModelSim) et synthétisables. A partir de cette base, l'outil GIC génère les fichiers de configurations synthétisables correspondant à l'IP généré par l'outil de synthèse GAUT.

Nous distinguons l'automatisation de cinq étapes pour la génération de l'interface de communication pour n'importe quel IP synthétisé par GAUT :

1. Nous assurons la généricité des paramètres en changeant leurs valeurs selon l'application et l'IP cible.
2. A partir de ces paramètres, le code des fichiers implémentant les modules constituants l'interface en question est changée.
3. Les modules de l'interface sont instanciés selon le besoin de l'IP considéré.
4. Un « testbench » est généré pour l'assemblage des sous modules constituant l'interface.
5. Génération du pilote logiciel

Nous détaillons ces différentes manipulations dans la suite.

4.4.1. Modification des paramètres génériques des modules VHDL

Les paramètres configurables pour l'utilisation générique de la bibliothèque sont implémentés en tant que paramètres génériques dans les fichiers VHDL (Cf. figure 64).

Le générateur procède à modifier les paramètres génériques de chaque « entity » module constituant l'interface d'un nouveau IP. Par exemple, les paramètres « depth », « width » et « Nb_port » de l'« entity » module CTRL_IN ont changé de valeurs pour deux cas d'IP.

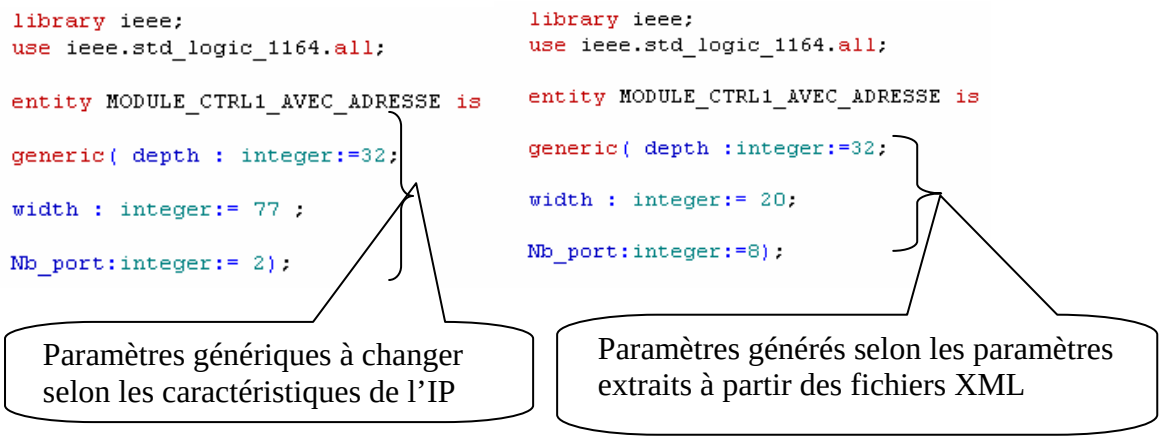

Figure 64. Paramètres génériques

Nous détaillons dans la section suivante le mécanisme à travers lequel le GIC peut modifier les paramètres génériques des modules de l'interface.

4.4.2. Généricité du code dans un module

Une autre fonctionnalité du GIC est la modification des modules de contrôle de l'interface selon le besoin de l'IP. Nous détaillons cette fonctionnalité à travers un exemple.

La figure 65 montre un extrait de code VHDL pris à partir du fichier correspondant au module CTRL_AVEC_ADRESSE_OUT permettant de générer la sortie du module vers le système. Il correspond à un IP possédant un seul port de sortie.

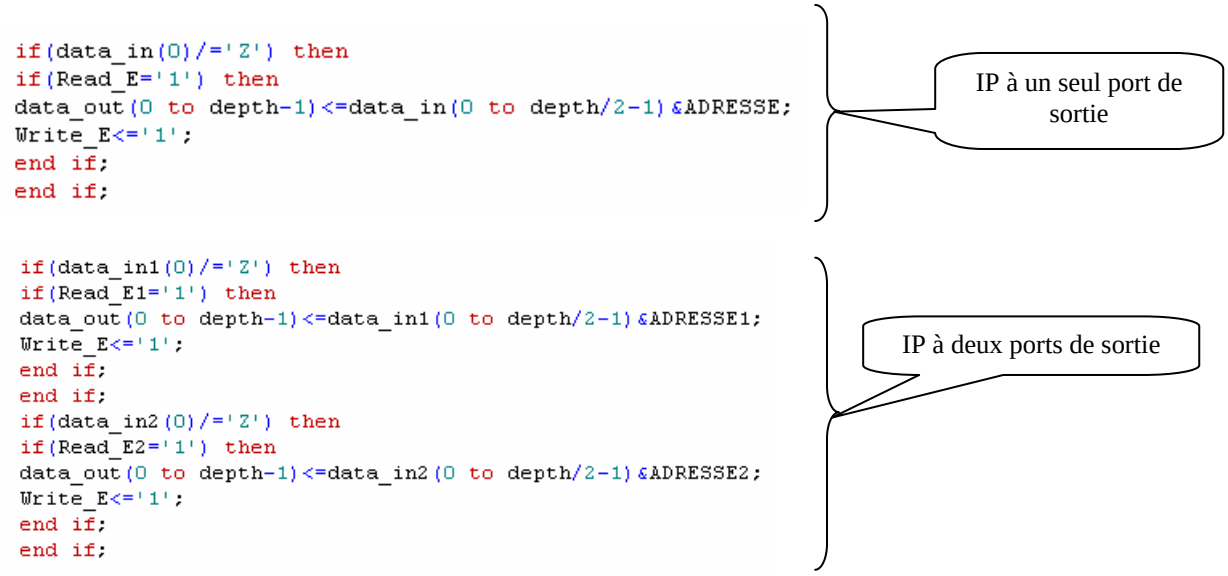

Figure 65. Généricité du code

Si l'IP dispose de deux sorties, le module CTRL_AVEC_ADRESSE_OUT doit multiplexer ces deux sorties en un seul port de sortie de l'interface. Dans ce cas, le GIC procède par générer deux routines dans le fichier résultant du générateur. Le second extrait de code (Cf. figure 65) correspond alors à un IP possédant deux ports de sortie.

Nous donnons dans la figure 66 le code JAVA du GIC qui régénère le code VHDL selon ces nouveaux paramètres dans un module de l'interface.

```
if(ligneLue.trim().equals("if(data_in(0)/='Z') then"))
  {
   for(int i=1;i<(me.n-mc2.nb_Bus)+1;i++)
   {
    writer.println("if(data_in"+i+"(0)/='Z') then");
    writer.println("if(Read_E"+i+"='1') then");
    writer.println("data_out(0 to depth-1)<=data_in"+i+"(0 to depth/2-1)&ADRESSE"+i+";")
    writer.println("Write_E<='1';");
    writer.println("end if;");
    writer.println("end if;");
   }
   for(int i=0;i<5;i++)
    ligneLue=reader.readLine();
  }
```

Figure 66. Extrait de code JAVA

Nous présentons dans la section suivante une autre fonctionnalité du GIC qui est l'instanciation des modules VHDL.

4.4.3. Instanciation des modules constituants l'interface

La généricité réside encore dans l'instanciation des modules constituant l'interface. En effet, selon le nombre de ports et que ce soit des ports de sorties ou des ports d'entrées, nous associons des mémoires FIFOs pour le tamponnage instantané. Les modules FIFOs sont instanciés selon le besoin. Ci-dessous des lignes de code d'instanciation de plusieurs FIFOs « FIFO_SANS_ADRESSE_FE» intercalées entre les modules CTRL_IN et FIFO_Enable. En effet, le nombre de FIFOs dépend du nombre de bus précisés par les paramètres de caractérisation de l'IP généré par GAUT (Cf. extrait de code de la figure 67).

```
b2v_inst10 : fifo_sans_adresse_FE
PORT MAP(Clock => Clock,
     Reset => Reset,
     Write_E => SYNTHESIZED_WIRE_2,
     data_in => SYNTHESIZED_WIRE_3,
     Read_E => SYNTHESIZED_WIRE_22,
     data_out => SYNTHESIZED_WIRE_27,
     WE_OK =>SYNTHESIZED_WIRE_54);

b2v_inst11 : fifo_sans_adresse_FE
PORT MAP(Clock => Clock,
     Reset => Reset,
     Write_E => SYNTHESIZED_WIRE_4,
     data_in => SYNTHESIZED_WIRE_5,
     Read_E => SYNTHESIZED_WIRE_16,
     data_out => SYNTHESIZED_WIRE_29,
     WE_OK =>SYNTHESIZED_WIRE_48);

b2v_inst12 : fifo_sans_adresse_FE
PORT MAP(Clock => Clock,
     Reset => Reset,
     Write_E => SYNTHESIZED_WIRE_6,
     data_in => SYNTHESIZED_WIRE_7,
     Read_E => SYNTHESIZED_WIRE_15,
     data_out => SYNTHESIZED_WIRE_28,
     WE_OK =>SYNTHESIZED_WIRE_47);
```

Figure 67. Instanciation des FIFOs

Les modules sont instanciés selon le besoin : si nous disposons d'un seul port de sortie pour l'IP, nous n'instancions pas le module CTRL_SANS_ADRESSE_OUT. Les données sont alors à multiplexer pour un seul port de sortie. Nous nous contentons donc d'une chaîne de communication de sortie disposant d'un seul élément FIFO de sortie.

```
component module_ctrl1_sans_adresse
   PORT(Clock : IN STD_LOGIC;
        Reset : IN STD_LOGIC;
        Read_E : IN STD_LOGIC;
        data_in : IN STD_LOGIC_VECTOR(0 to depth-1);
        Write_E1 : OUT STD_LOGIC;
        Write_E2 : OUT STD_LOGIC;
        data_out1 : OUT STD_LOGIC_VECTOR(0 to depth-1);
        data_out2 : OUT STD_LOGIC_VECTOR(0 to depth-1));
end component;

component fifo_sans_adresse
   PORT(Clock : IN STD_LOGIC;
        Reset : IN STD_LOGIC;
        Write_E : IN STD_LOGIC;
        data_in : IN STD_LOGIC_VECTOR(0 to depth-1);
        Read_E : OUT STD_LOGIC;
        data_out : OUT STD_LOGIC_VECTOR(0 to depth-1));
end component;
```

Figure 68. Instanciation des sous modules

L'interface est générée par instanciation des modules qui la constituent (Cf. figure 68).

Le générateur permet en outre de générer un fichier de test « testbench » associé à l'interface pour sa simulation sous « ModelSim ». Ce « testbench » permet l'assemblage des modules constituant l'interface, de gérer leur synchronisation à travers les signaux de contrôle des composants et les signaux de synchronisation.

4.4.4. Assemblage des modules

Après la génération de tous les fichiers constituant l'interface, le « testbench » permet d'assembler tous les composants nécessaires pour assurer la communication entre l'IP et le système.

La figure 69 montre un exemple d'instanciation du module CTRL_SANS_ADRESSE_IN et deux FIFOs, qui correspondent au nombre de ports utilisés dans l'interface, avec les signaux nécessaires pour leur assemblage.

```
b2v_inst1 : module_ctrl1_sans_adresse

PORT MAP(Clock => Clock,
Reset => Reset,
Read_E => SYNTHESIZED_WIRE_0,
data_in => SYNTHESIZED_WIRE_1,
Write_E1 => SYNTHESIZED_WIRE_32,
Write_E2 => SYNTHESIZED_WIRE_34,
data_out1 => SYNTHESIZED_WIRE_33,
data_out2 => SYNTHESIZED_WIRE_35);

b2v_inst2 : fifo_sans_adresse

PORT MAP(Clock => Clock,
Reset => Reset,
Write_E => SYNTHESIZED_WIRE_32,
data_in => SYNTHESIZED_WIRE_33,
Read_E => SYNTHESIZED_WIRE_10,
data_out => SYNTHESIZED_WIRE_12);

b2v_inst3 : fifo_sans_adresse

PORT MAP(Clock => Clock,
Reset => Reset,
Write_E => SYNTHESIZED_WIRE_34,
data_in => SYNTHESIZED_WIRE_35,
Read_E => SYNTHESIZED_WIRE_11,
data_out => SYNTHESIZED_WIRE_13);
```

Figure 69. « Testbench » ou fichier d'assemblage

4.4.4. Génération du « Driver »

Le GIC permet la génération de l'interface pour une simulation au niveau comportemental et synthétisable de l'interface. La génération de l'interface pour la synthèse cible la technologie Altera. Dans la section suivante, nous présentons quelques résultats de

synthèse et de valorisation de la génération automatique d'interfaces de communication avec l'outil GIC.

4.5. Synthèse et Résultats

L'application utilisée pour l'expérimentation avec l'approche proposée est « la synthèse d'image 3D ». Le code original permet d'afficher un objet sur l'écran, les coordonnées de cet objet sont sauvegardées dans un fichier « .asc ». Nous avons modifié le code de cette application pour l'adapter au compilateur du processeur NIOS 32 bits. En effet, puisque le compilateur « C » du NIOS ne gère pas les fichiers, le fichier d'extension .ASC a été remplacé par un fichier d'entête d'extension .h appelé par le programme principal du rendu 3D. Nous avons pour cela créé une procédure de conversion d'un fichier .ASC vers un fichier d'entête .h (.ASC=>.h). Nous avons également éliminé la partie du code responsable de l'affichage sur écran.

A l'aide du logiciel « 3Ds max » [3DS] qui est un logiciel de production d'images de synthèse tridimensionnelles, nous élaborons une sphère ayant un nombre varié de triangles. 3Dmax génère des fichiers d'extension .3ds qui sont convertis au format .ASC. À l'aide de l'utilitaire wc2pov27 [Wc297], ces derniers sont à leur tour convertis à l'aide de la procédure de conversion .ASC=>.h. En modifiant le fichier d'entête .h, nous pouvons tester le programme de rendu d'un objet (exemple d'une sphère) pour différents nombres de triangles (Cf. figure 70). Chaque point de la sphère a une couleur codée sur 16 bits.

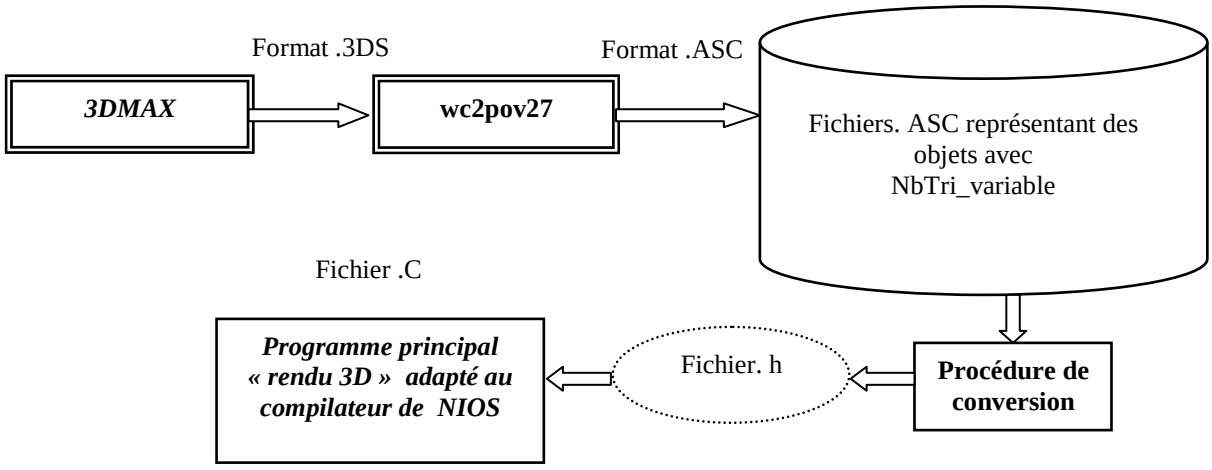

Figure 70. Procédure pour l'étude de l'effet de la modification du nombre de triangles

L'IP cible est un IP synthétisé par l'outil de synthèse GAUT. Le reste de l'application de synthèse 3D est exécuté en logiciel par le processeur NIOS équipé du bus Avalon

disponible sur la plateforme Excalibur d'ALTERA (figure 71). Le processeur NIOS est cadencé à la fréquence de 50 Mhz.

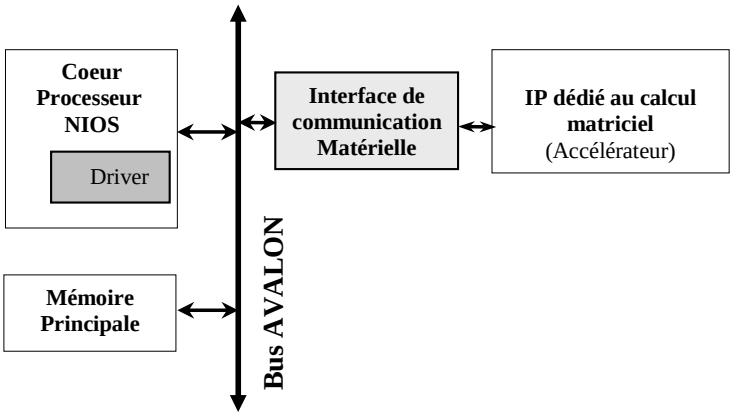

Figure 71. Structure de l'Architecture cible

4.5.1. Influence des paramètres de l'interface sur le temps d'exécution

Diverses techniques peuvent être utilisées pour la détermination du temps d'exécution (noté dans la suite Texec) d'une application sur une cible architecturale donnée. Elles sont divisées en deux catégories : la mesure et l'estimation. Nous avons utilisé la méthode développée dans [Fak04]. Elle consiste à utiliser un compteur qui est activé au début de l'exécution du programme puis arrêté à sa fin. La différence entre la valeur initiale et la valeur finale du compteur représente le temps d'exécution en nombre de cycles comme le montre la figure 72. Nous avons choisi cette méthode vu la facilité d'utilisation du compteur d'une part et vu la précision des résultats de mesure d'autre part.

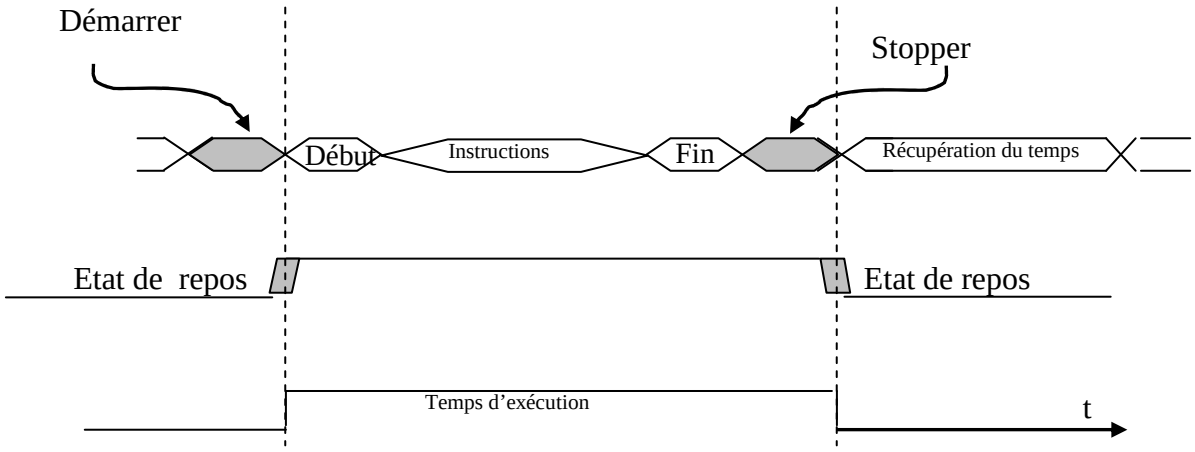

Figure 72. Détermination du temps d'exécution (Texec)

Nous avons calculé le Texec de l'application « rendu 3D » d'une sphère effectuant une séquence de rotations autour des trois axes d'animation 100 fois avec trois versions :

– une version logicielle pure : toute l'application est exécutée par le processeur NIOS.

- une version mixte {logicielle+ interface+ IP accélérateur} en utilisant la première conception de l'interface.
- Afin d'apercevoir l'effet de l'interface sur l'application, nous avons également déterminé le temps d'exécution de l'application en utilisant une communication entre l'IP et le processeur NIOS à l'aide de composants PIO. Le PIO est un composant disponible dans l'environnement Quartus qui facilite l'interconnexion de blocs matériels au processeur NIOS. Dans ce cas, l'envoi des données se fait selon l'ordre demandé par l'IP « produit matriciel ».

Les résultats sont rassemblés dans le tableau 4. Nous faisons varier le nombre de triangles constituant la sphère. Le nombre de triangles influe sur la qualité de la sphère visualisée en 3D sur un écran 2D.

Tableau 4. Mesures de Texec

Nb triangles	Texec (ms) (version logicielle)	Texec (ms) (version mixte sans interface)	Texec (ms) (version mixte avec interface conception1)
62	13839	4152	4637
146	42025	12608	13205
191	44724	13417	13966
266	64364	19309	19745
366	75211	22563	23937

Nous remarquons que l'ajout de l'IP (cas 3) « produit matriciel » a permis un gain moyen en Texec par rapport à la version logicielle de 70%. L'« overhead » dû à l'interface est d'une moyenne de 500ms pour le déroulement de toute l'application.

4.5.2. Comparaison de la surface de l'IP par rapport à son interface

Nous considérons les résultats présentés par le tableau 5. Nous présentons dans la suite les résultats de synthèse de l'IP « produit matriciel » et de l'interface pour une cible FPGA de la famille STRATIX d'Altéra.

Tableau 5. Influence du Nb_bus utiles sur l'occupation FPGA

	Cas1/Cas 2	Cas 4	Cas 5
Nb_bus utiles	4	15	28
Nb_Lut IP sur FPGA (Famille Stratix EPS1S10F484C3)	677	1198	1769
Nb_Lut Interface (conception1) (Famille Stratix EPS1S10F484C3)	156	540	979
Nb_Lut Interface (conception2) (Famille Stratix EPS1S10F484C3)	223	607	1046

Schématiquement, ce tableau ce traduit par la figure 73.

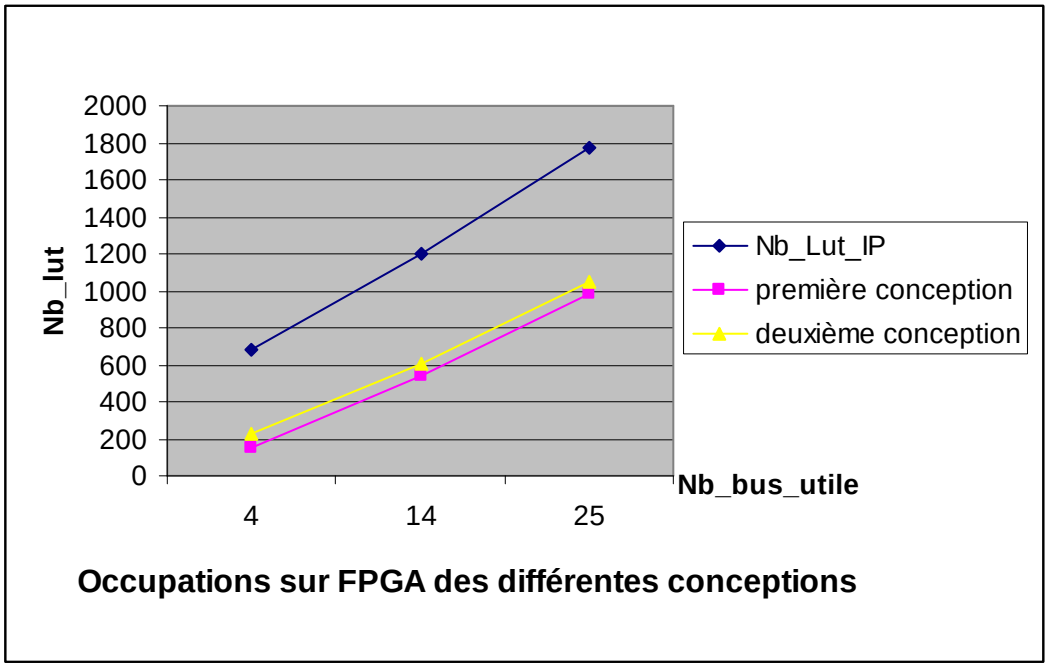

Figure 73. Occupations sur FPGA

La première conception occupe moins de surface que la deuxième conception vu qu'elle ne procède pas à un décodage d'adresse sur les données. En plus, dans la deuxième conception, le module FIFO_IN possède des cellules plus larges puisqu'elles contiennent la donnée et son adresse.

En conclusion, le coût de l'interface est principalement dominé par la taille des FIFOs, si bien que leur minimisation doit être une priorité.

4.6. Optimisation de la taille des FIFOs

Les FIFOs dans une interface ont deux rôles. Le premier est un rôle de découplage entre le système et l'IP connecté en terme de synchronisation. Le second est un rôle de mémorisation intermédiaire de la succession (ordonnancement spatial) des données.

En revanche, le coût des FIFOs en terme de Luts sur FPGA est important. Il est particulièrement important pour une intégration d'un IP GAUT vu :

- le nombre de bus d'interconnexion à l'interface de l'IP : à chaque bus correspond une FIFO.
- la considération du sens de transmission (in, out) : à chaque bus d'entrée/sortie correspond deux FIFOs.

Pour une simulation, la taille des FIFOs n'influe pas sur le temps de simulation. Mais pour la synthèse, la profondeur de la FIFO coûte en terme de cellules de mémorisation.

La classification des FIFOs dans l'interface de communication est considérée selon deux niveaux (cf. figure 74) : niveau 1 (N1) et niveau 2 (N2). Le niveau 1 contient des FIFOs de type FIFO_IN et FIFO_OUT qui peuvent subir une gestion de données (Cf. section 4.3 du chapitre 3). Le niveau 2 contient des FIFOs simples (sans gestion) dont la taille des cellules est égale à la taille des données réelles à consommer ou à produire par l'IP.

La taille d'une cellule FIFO est initialisée par le GIC à la largeur de l'interconnecte définie aussi par la taille de la donnée système : tds. Le nombre des cellules FIFOs est initialisé par une profondeur maximale. La profondeur maximale définie :

- le nombre maximum de données dans le motif considéré pour une itération de calcul pour les FIFOs N1.
- Le nombre maximum de données considéré pour une itération de calcul pour chaque FIFOs N2.

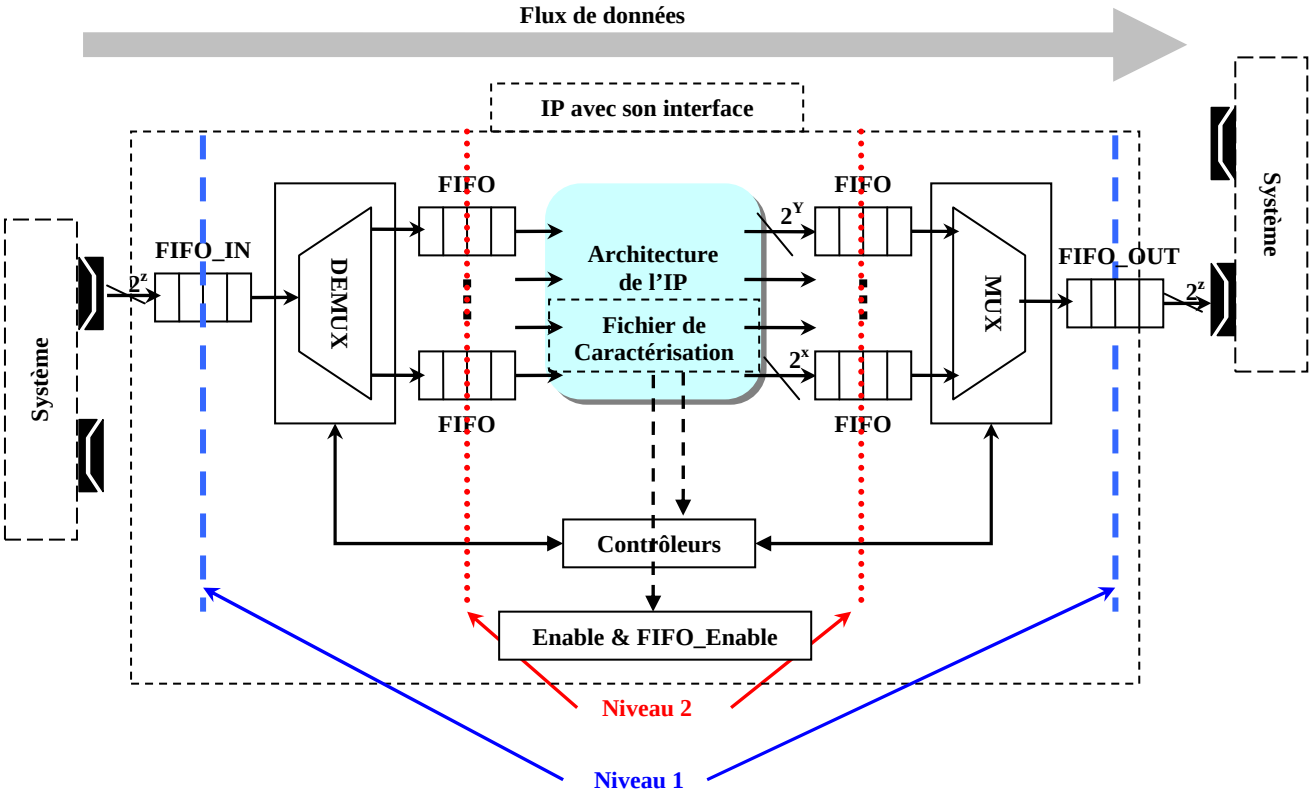

Figure 74. Classification des FIFOs dans l'interface

La gestion des cellules FIFOs (décrite à la section 4.3 du chapitre III) a permis dans des cas d'optimiser le nombre de cellules de la FIFO_IN (resp. FIFO_OUT) de l'interface de communication. En effet, si tds est supérieure à la taille de la donnée de l'IP accélérateur (tda), un motif occupe un nombre de cellules FIFOs inférieur à la profondeur maximale en utilisant le mécanisme de gestion de l'empilement des données au sein d'une cellule FIFO.

Mais cette technique de gestion des FIFOs reste insuffisante puisqu'elle :

- ne traite que les FIFOs d'interconnexion au système (FIFO niveau 1 dans la figure 74 définie par FIFO_IN et FIFO_OUT).
- ne permet pas toujours une optimisation : la gestion signifie une optimisation dans le cas où tds> tda pour les deux conceptions de l'interface.

Ce qui nous a poussé à définir une méthode pour calculer la profondeur minimale des FIFOs nécessaire pour le bon fonctionnement de l'interface.

Dans les expériences qui ont été faites que ce soit pour le contexte de simulation ou de synthèse, les profondeurs des FIFOs de niveau 1 et de niveau 2 sont initialisées par défaut par le GIC à la profondeur maximale pour la communication définie pour une itération de calcul. En effet, l'architecture de l'interface de communication est définie selon la caractérisation d'une itération de calcul, le nombre total d'itérations de calcul n'influe pas sur la taille des FIFOs.

Pour optimiser la profondeur des FIFOs, nous proposons l'algorithme de la figure 75.

```
Début
Profondeur :=profondeur maximale ; //initialisation
        Faire
                Profondeur=profondeur-1 ; //Diminuer d'une cellule la profondeur de chaque FIFO
                dans un niveau de classification des FIFOs;
                //Tester le fonctionnement de l'interface ;
        Jusqu'à (//interface ne fonctionne pas ou interface produit des résultats inexacts)
Profondeur=profondeur+1 ;
Fin
```

Figure 75. Algorithme de calcul de la profondeur FIFO

Cet algorithme itératif est proposé pour l'optimisation de la profondeur des FIFOs. En revanche, il est coûteux en terme de nombre de test. Afin de limiter ce nombre, nous pouvons procéder pour une recherche dichotomique (cf. figure 76) qui considère :

- une recherche récursive dans une moitié du tableau.
- que les éléments du tableau sont deux à deux distincts ;
- que les éléments du tableau sont rangés dans l'ordre croissant ;

Les éléments du tableau sont les numéros des cellules de la FIFO. Le raffinement considère :

- une condition d'arrêt : taille du tableau =1,

- taille = 0 non trouve,
- taille = 1 tester ;
- les limites du tableau (deb et fin) sont des arguments de la fonction.

```
Début
   int rechDichRec(int valRech, int FIFO[], int deb, int fin)
   {
        int mil = (deb+fin)/2;
        if(deb > fin) return (-1);
        if(valRech == FIFO[mil]) return mil;
        if (valRech < FIFO[mil])
                return rechDichRec(valRech, FIFO, deb, mil-1);
        return rechDichRec(valRech, FIFO, mil+1, fin);
   }
Fin
```

Figure 76. Algorithme de la recherche dichotomique

D'où, l'algorithme de calcul de la profondeur de FIFO devient l'algorithme donné par la figure 77 :

```
Début
fin :=profondeur maximale ; //initialisation
deb :=1 ; //initialisation
        int rechDichRec(bool  test, int FIFO[], int deb, int fin)
        {
             int mil = (deb+fin)/2;
             if(deb > fin)
                return fin;
             //Tester le fonctionnement de l'interface avec la nouvelle profondeur = fin ;
             if (test ==true)
                return rechDichRec(test, FIFO, 1, mil-1);
             sinon
                return rechDichRec(test, FIFO, mil+1, fin);
        }
Fin
```

Figure 77. Algorithme de calcul de la profondeur FIFO considérant une recherche dichotomique

À chaque étape :

- 1 addition, 1 division, entre 1 et 3 tests et un appel récursif.
- division par 2 de la taille du tableau FIFO.

L'ordre de complexité à la recherche de la profondeur minimale est donc $O(log_2(n))$ qui est inférieur à la complexité exponentielle de l'algorithme de calcul de la profondeur de FIFO. Ce qui permet de réduire le nombre de test significativement pour des flux de données de tailles importantes.

Nous avons utilisé l'outil GIC pour générer l'interface de communication. La simulation de la structure synthétisable de l'interface de communication avec son « testbench » sous Modelsim nous a permis de trouver la profondeur minimale pour le fonctionnement de l'interface de communication selon l'algorithme proposé.

Le tableau 6 prouve que le nombre d'itérations n'influe pas sur la profondeur des FIFOs, autrement dit, le nombre de données échangées en totalité n'influe pas sur la profondeur des FIFOs. C'est le nombre de données échangées lors d'une itération de calcul qui :

- S'il augmente, la profondeur au minimum des FIFOs dans l'interface augmente.
- S'il diminue, la profondeur au minimum des FIFOs diminue.

Tableau 6. Influence du nombre de données sur la profondeur minimale des FIFOs

Nom de l'exemple (architecture GAUT ***sans unité de duplication***)	Nombre de données par itération de calcul	Taille minimale des FIFO_IN de l'interface générée (une seule FIFO_IN)
Prodmat 4x4	77	31
Dct 4x4	45	10
FFT 4 points	4	4
Fir 16	1	1

Pour notre exemple « produit de deux matrices 4x4», nous considérons le même IP avec des architectures différentes. Le nombre de bus utiles pour le transfert des données diffère d'une architecture à une autre. Le tableau 7 présente les résultats sur la profondeur des FIFOs suite à l'application de l'algorithme présenté.

Tableau 7. Application de l'algorithme de re-dimentionnement de FIFO

	Cas1/Cas 2			Cas 4			Cas 5		
Nb_bus utiles	4			15			28		
Nombre de FIFOs	4 FIFOs N2	2 FIFOs N1		15 FIFOs N2	2 FIFOs N1		28 FIFOs N2	2 FIFOs N1	
Profondeur maximale par FIFO	16	32	16	16	32	16	16	32	16
profondeur minimale par FIFO	4	13	7	4	13	7	4	13	7
Total des cellules FIFOs (profondeur ***maximale***)	16x4 + (32+16) =***112***			16x15 + (32+16)=***288***			16x28 + (32+16)=***496***		
Total des cellules FIFOs (profondeur ***minimale***)	4x4 + (13+7)=***36***			4x15 + (13+7)=***80***			4x28 + (13+7)=***132***		
Gain relatif	71%			72%			73%		

L'application de l'algorithme permet un gain de plus de 70% en termes de cellules FIFOs. Le tableau 8 illustre l'influence de la profondeur des FIFOs sur l'occupation FPGA. Nous remarquons comme prévu, que le fait de réduire la taille des FIFOs implique un gain en terme de coût en surface ou dans notre cas en terme d'occupation sur FPGA.

Tableau 8. Influence de la profondeur des FIFOs sur l'occupation FPGA

	Cas1/Cas 2	Cas 4	Cas 5
Nb_bus utiles	4	15	28
Nb_Lut Interface (conception1) (Famille Stratix EPS1S10F484C5)	156	540	979
Nb_Lut Interface (conception2) (Famille Stratix EPS1S10F484C5)	223	607	1046
Nb_Lut Interface (conception1) (Famille Stratix EPS1S10F484C5) avec optimisation	129	487	928
Nb_Lut Interface (conception2) (Famille Stratix EPS1S10F484C5) avec optimisation	197	564	986
Gain relatif (conception 1)	17%	10%	5%
Gain relatif (conception 2)	11%	7%	6%

Le dimensionnent de la FIFO doit prendre l'aspect de distribution sur les cellules en considération. L'IP GAUT requière des FIFOs plus petites mais plus nombreuses. Cela implique plus de surface mais aussi plus de contrôle.

En revanche, la recherche dichotomique suppose que la répartition des données qui transitent dans les FIFO en amont des bus ou encore dans les FIFOs de niveau 2 est équitable. En réalité, la répartition est imprévisible par l'outil GAUT. A l'interface de l'architecture générée par GAUT, la répartition des données est inéquitables et imprévisibles dans les bus. Le nombre des données transitant en entrée ou en sortie sur les bus est distribué d'une manière non égale. Donc, chaque FIFO de niveau N1 ou de niveau N2 doit être considérée à part. D'où l'apparition d'un problème de distribution des profondeurs des FIFOs. Un algorithme « Branch and Bound » [Cla99] peut être considéré pour résoudre le problème d'optimisation sur le nombre de test pour l'optimisation de la profondeur des FIFOs.

Bref, le problème de redimensionnement de la taille des FIFOs doit être considéré à la fin du flot d'intégration d'IP et que cette fonctionnalité soit intégrée dans l'outil GIC juste avant la génération du code de l'interface de communication.

5. Conclusions

Dans ce chapitre, nous avons présenté l'outil GIC qui automatise les étapes du flot proposé pour l'intégration d'IP. Le GIC permet de générer l'interface de communication entre un IP (généré par l'outil GAUT) et le reste du système. Cette interface peut être générée pour

la simulation, dans ce cas elle est décrite en SystemC. Elle peut être générée pour un contexte de synthèse, dans ce cas elle est décrite en langage VHDL. Dans les deux cas, le principe est le même. Les modules constituant l'interface sont décrits sous forme générique. Ils sont regroupés dans une bibliothèque. A partir de cette bibliothèque, le GIC modifie leurs paramètres génériques, et instancie les modules nécessaires selon la caractérisation du transfert des données sous forme des graphes d'ordonnancement des données. Il génère également le fichier de test associé. Dans les deux cas (simulation et synthèse), nous avons utilisé l'IP « produit matriciel » généré par l'outil GAUT.

La génération de l'interface de communication est testée pour le contexte de simulation en utilisant la plateforme SoCLiB. Pour le contexte de synthèse, l'interface est testée en utilisant le processeur NIOS équipé du bus AVALON. L'application cible est la « synthèse d'images 3D ».

Nous avons élaboré également une démarche pour l'optimisation de la taille des FIFOs qui équipent l'interface afin de réduire son occupation sur FPGA.

Divers tests ont été effectués avec le GIC pour les deux contextes : simulation et synthèse. Nous avons étudié l'impact de la variation des paramètres de l'IP sur la taille de l'interface. A travers ces tests, nous avons montré que l'architecture de l'interface de communication est bien indépendante de l'interconnecte du système intégrant. Sa structure modulaire lui permet une adaptation facile (avec des modifications mineures) avec d'autres systèmes.

Nous avons montré que l'architecture générique du module d'interface permet de réaliser automatiquement l'intégration d'IP dans un système en instanciant des schémas de communication prédéfinis sous contraintes.

De plus, cet outil est un environnement ouvert pour l'exploration de nouvelles pistes. Il peut être équipé d'outils de décision, par exemple, qui automatise l'optimisation de la taille des FIFOs ou qui évalue des résultats de l'intégration.

CHAPITRE 6. CONCLUSIONS & PERSPECTIVES

Ce chapitre conclut la thèse en donnant un bilan du travail effectué et en présentant les perspectives envisageables au terme de cette recherche. Pour cela, nous allons tout d'abord analyser les aspects de l'approche d'intégration d'IP et l'environnement d'application que ce soit pour la simulation ou pour la synthèse. Nous résumons donc les caractéristiques de ces aspects ainsi que notre contribution. Enfin, nous exposons les perspectives de notre travail.

1. Synthèse des travaux de thèse

Les nouvelles technologies s'orientent vers l'intégration sur une même puce de plusieurs cœurs de processeurs tels que les DSPs et les processeurs RISC, ainsi que des blocs matériels dédiés à certaines fonctions, totalisant plusieurs centaines de millions de portes sur une seule puce de silicium. Nous parlons ainsi de systèmes multiprocesseurs mono puce. En fait, les systèmes multiprocesseurs sont une des solutions pour répondre à la complexité croissante des systèmes intégrés utilisés dans les domaines multimédia et traitement de signal. En plus de la complexité des applications supportées, les SoCs sont assujettis à des fortes contraintes de conception telles que les contraintes temps réel, la limitation des ressources de traitement et d'énergie et aussi les contraintes de coût et temps de mise sur le marché. Pour satisfaire ces contraintes antagonistes, l'intégration des blocs existants dits IPs est devenue une exigence. Ceci nécessite des outils de CAO et des langages de haut niveau unifiés pour la conception dite système.

Dans ce cadre se situent les travaux de cette thèse. Elle présente une approche de génération d'interface de communication dans le cadre de la conception d'un SoC/MPSoC pour l'intégration des IPs. Cette approche permet au concepteur d'automatiser la phase d'intégration d'IPs. Elle est considérée dans un cadre de simulation et de synthèse. La génération est basée sur une structure générique d'interface de communication. Cette interface est élaborée au profit des IPs orientés flot de données. Le modèle d'interface est suffisamment générique afin de pouvoir l'adapter aux différents protocoles de communication.

Dans le chapitre 2, nous avons rappelé le flot de conception dans le cadre du « co-design» et de la conception des SoCs. Dans ce cadre, un état de l'art sur la conception automatisée des interfaces de communications a été présenté : les approches, des méthodologies permettant la réutilisation assistée ou automatisée des IPs dans la conception des systèmes monopuce mono/multiprocesseur. Il a été montré que les approches concernées

diffèrent par la spécification de l'application en entrée, les domaines d'application et l'architecture cible.

Le chapitre 3 a présenté l'approche d'intégration que nous proposons. Dans ce chapitre, nous avons traité les hypothèses et les contraintes pour l'application de l'approche. Cette approche part d'un système partitionné en tâches logicielles et matérielles, et aboutit à un système contenant des IPs interconnectés. L'application de l'approche prend en considération la nature de l'IP à intégrer, l'architecture cible d'intégration en se basant sur la compatibilité de l'ordonnancement du transfert des données entre l'IP et le système. Nous avons présenté l'implémentation structurelle du modèle générique de l'interface de communication. Ceci permet la construction d'une librairie de modules assurant l'automatisation de la génération de l'interface sans avoir recours à spécifier les détails relatifs aux protocoles de communication dans une architecture cible. La bibliothèque de communication interagit avec un modèle de graphe permettant une spécification générique de l'ordonnancement des données du côté du composant virtuel et du côté du système intégrant.

Nous avons détaillé dans le chapitre 4 les étapes de conception de réalisation d'un environnement « générateur d'interfaces de communication» automatisant les étapes de l'approche. L'environnement développé est de type générateur de code. Il est facilement extensible pour être utilisé avec d'autres outils de conception réalisant d'autres tâches de co-design. Ce qui permet d'augmenter la productivité dans un milieu industriel.

Dans le chapitre 5, nous avons présenté les résultats d'expérimentation de notre approche pour un contexte de simulation et de synthèse. Deux environnements ont été utilisés : « SoCLiB » pour la simulation et « Quartus » (dédié pour les circuits FPGA Altéra) pour la synthèse. L'application cible utilisée est le « Pipeline graphique 3D ». C'est une application complexe qui doit faire appel à des IPs accélérateurs pour l'amélioration de son temps de calcul. L'IP choisi est « le produit matriciel ». Nous nous sommes placés dans le cadre d'un flot de conception SoC basé sur des outils SHN. L'implémentation des modules pour la réalisation de la bibliothèque de composants de l'interface est élaborée pour des IPs synthétisés avec l'outil GAUT. L'interface de communication est générée pour la simulation puis pour la synthèse ce qui nous a permis de mesurer ses performances en termes de délai et de complexité.

En résumé, l'approche d'intégration d'IP proposée est basée sur une architecture générique modulaire et flexible d'interface dédiée aux applications orientés flot de données. Elle permet la génération des structures des schémas de communication du système incluant l'IP. En effet, la communication du système peut être validée en utilisant une simulation

rapide. Le modèle d'architecture associé permet une implémentation matérielle facile de l'interface.

Par ailleurs, l'approche cible les méthodologies de conception des circuits intégrés à base de plateforme (platform based design) et à base de composant (component based design).

Notre contribution dans le cadre de cette thèse intervient à deux niveaux : méthodologique et applicatif. Le niveau méthodologique concerne l'élaboration d'une approche de génération d'interfaces de communication pour l'intégration des IPs accélérateurs dans un système sur puce. Le niveau applicatif concerne la validation de cette approche à travers :

- son automatisation dans un environnement de type générateur de code implémentant les étapes du flot d'intégration qui lui est associée.
- son expérimentation pour l'application « synthèse 3D ».

2. Perspectives

Plusieurs perspectives se dégagent à la suite de ces travaux de recherche. D'une part, l'application de l'outil pour la simulation ou pour la synthèse à partir de l'architecture de l'interface nécessite plus de raffinement. D'autre part, ce raffinement permettra d'intégrer l'approche de vérification formelle au flot d'intégration afin de garantir des spécifications exemptes d'erreurs. Cette approche de vérification formelle peut être appliquée surtout pour l'étape de génération de graphes et de test de compatibilité.

Les perspectives envisageables en prolongement direct de cette thèse concernent deux points :

- Le premier point est l'extension de l'approche pour permettre l'intégration d'IPs autres que les accélérateurs matériels.
- Le second point correspond à l'extension de l'environnement pour être un support de génération automatique et d'évaluation des résultats de simulation et des résultats de synthèse.

2.1. Extension de l'approche

L'approche d'intégration d'IP proposée se fait après le partitionnement. Le seul moyen qui permet d'éviter le problème de révisions tardives du partitionnement, si les contraintes de performances ne sont pas satisfaites, réside actuellement dans la modularité et la flexibilité de l'architecture cible. Comme perspective, nous pouvons étendre les primitives de l'approche

pour permettre la génération d'interfaces de communication d'une manière interactive avec l'étape de partitionnement comme celle proposée dans [Jer05]. Dans ce cas, l'approche peut être étendue pour les IPs comportementaux où les contraintes aux données sont plus souples qu'une spécification CABA. Ainsi l'application de l'approche peut s'orienter de plus en plus vers des niveaux d'abstraction plus élevés.

Nous envisageons à moyen terme, d'ajouter de l'intelligence au contrôleur de l'interface. Ainsi, ce travail peut être étendu pour assurer la communication avec des IPs de sources différentes (synthétisés par exemple avec d'autres outils SHN), ou des IPs mémoires ou des IPs processeurs. L'objectif est d'adapter l'approche d'intégration et la conception de l'interface de communication pour généraliser leur application.

2.2. Extension de l'environnement

Les travaux liés à l'intégration d'IPs correspondent à un besoin réel dans la conception des systèmes mono puce mono/multiprocesseurs. Le travail présenté dans cette thèse peut être considéré comme une contribution à la mise en œuvre d'un ensemble de règles applicatives pour la génération automatique des structures de communications (interface, contrôleurs) au niveau comportemental et RTL, facilement synthétisables par les outils de synthèse actuels. Les travaux à venir doivent permettre :

- l'extension de l'environnement pour permettre l'édition graphique du modèle d'entrée, l'édition des configurations et la visualisation des résultats.
- l'évaluation des résultats de l'intégration par l'environnement en tant que support de programmation complet pouvant posséder des métriques pour la mesure des performances.

RÉFÉRENCES BIBLIOGRAPHIQUES

[Abb06a] F. Abbes, M. Abid, E. Casseau, «Interface Architecture Generation for IP Integration in SoC Design», Computer Engineering & Systems (ICCES'06), Egypt, Novembre 5-7, 2006.

[Abi98] M. Abid, «Hardware/Software Co-design methodology for design of embedded system», Integrated Computer Aided Engineering Journal, Vol. 5, pp69-83, 1998.

[Abi00] M. Abid, «Contribution à la conception des systèmes électroniques mixtes logiciels/matériels», thèse de doctorat d'état, Ecole nationale d'ingénieurs de Tunis, Mai 2000.

[ALT] http://www.altera.com.

[Alt04] Nios custom instructions User guide, Altera Corporation, 2004.

[Alt05] SOPC and DSP Builder, Altera Corporation, 2005.

[Aou04] Y. Aoudni, N. BenAmor, G. Gogniat, J.L. Philippe, M.Abid, «Platform and Architecture Adequacy in SoC environment: a case study», International Conference on Microelectronics ICM, Tunis, Tunisia, Décembre 2004.

[ARM99] ARM. AMBA specification Rev 2.0, 1999.

[Bar99]C. Barna, W. Rosenstiel. «Object-Oriented Reuse Methodology for VHDL», proceeding DATE, pp. 689 – 693, Munich, Allemagne, 1999.

[Ben95] T. Ben Ismail, A. A.Jerraya, «Synthesis Steps and Design Model for Co-design», IEEE Computer, Vol. 28, No. 2, pp. 44-52, Février 1995.

[Ben97]T. Benner, R.Enst, «An Approach to Mixed System Co-Synthesis», Workshop on Hardware/Software Co-design, Braunschweig, Allemagne, Mars 1997.

[Ben02]L. Benini, G. De Micheli, « Networks on Chip : A New SoC Paradigm», IEEE Computer, Volume 35, Issue 1, pp. 70 – 78, Janvier 2002.

[Ber00] R. A.Bergamaschi, W. R.Lee. «Designing Systems-on-Chip Using Cores», International Design Automation Conference (DAC), Los Angeles, California, Etats Unis, 2000.

[Bok00] C.Böke. « Combining Two Customization Approaches: Extending the Customization Tool TEReCS for Software Synthesis of Real-Time Execution Platforms», Workshop on Architectures of Embedded Systems (AES2000), Janvier 2000.

[Bom04] P. Bomel, E. Casseau, « Projet ALIPTA : Spécification du protocole et de la structure de l'interface des circuits synthétisé par GAUT avec Plug-IP », Version 4.0, Janvier 2004.

[Bou05] B. Bouallegue, « Contribution à la conception des architectures réseau/application aux systèmes embarqués sur une seule puce », thèse, Université de Bretagne Sud, 2005.

[Bru00] J-Y Brunel, W. Kruijtzer, H. Kenter, F. Petrot, L. Pasquier, E.A. De Kock, W. Smits, «COSY Communication IP's», International Design Automation Conference (DAC), Los Angeles, Californie, Etats-Unis, 2000.

[Chi96] M. Chiodo, P. Giusto, A. Jurecska, «Synthesis of mixed software-hardware implémentations CFSM specifications», Workshop on hardware-software codesign. Cambridge, Massachussets, Octobre 1996.

[Cho95] P. H.Chou, R. B.Ortega, G. Boriello, «The Chinook Hardware/Software Co-Synthesis System», 8 ème International Symposium on System Synthesis, Septembre 1995.

[Cho99] P. Chou, R. Ortega, K. Hines, K. Patridge, G. Borriello, «IPChinook: An Integrated IP-Based Design Framework for Distributed Embedded Systems», International Design Automation Conference (DAC), New Orleans, Louisiana, Etats Unis, 1999.

[Cla99] J. Clausen, «Branch and Bound Algorithms -Principles and Examples», department of Computer Science, université de Copenhagen, Danemark. 1999.

[Cou03a] P. Coussy, « Synthèse d'Interface de Communication pour les Composants Virtuels », thèse, université de Bretagne sud, 2003.

[Cou03b] P. Coussy, A. Baganne, E. Martin, «Communication and Timing Constraints Analysis for IP Design and Integration», International Conference on Very Large Scale Integration (IFIP, VLSI-SoC), Darmstadt, Allemagne, Décembre 2003.

[Cou06] P. Coussy, E.Casseau, P.Bomel, A. Baganne, E. Martin , «Formal Method for Hardware IP Design and Integration», ACM Transactions on Embedded Computing Systems, Vol. 5, No. 1, Février 2006.

[CXA] http://www.celoxica.com/

[Cyr04] G. Cyr, G. Bois, M. Aboulhamid, «Generation of processor interface for SoC using VSIA recommendations», Computers and Digital Techniques, vol. 151, no. 05, pp. 367-376, Septembre 2004.

[Deu92] G.De Micheli, D.ku, «High level synthesis of ASICs under Timing and Synchronization Constraints», Editions Kluwer Academic Publishers Norwell, Etats Unis, 1992.

[Deu02] G. De Micheli. « Network on Chip: a new Paradigm for System on Chip Design», Design Automation and Test in Europe (DATE), Paris, 2002.

[DKDS05] DK Design Suite user guide, version 4, 2005, disponible à : http://www.celoxica.com/support/

[Don04] F. Donnet, «Synthèse de haut niveau contrôlée par l'utilisateur », thèse, université de Paris VI, 2004.

[Dor05] S. Dore, http://pages.infinit.net/shaun/

[Ern93] R. Ernst, J. Henkel, Th. Benner, «Hardware software cosynthesis for microcontrollers», IEEE Design and Test of computers, ISSN: 0740-7475, Vol. 10, pp.64-75, 1993.

[Fak04] S. Fakhfakh, «Evaluation du temps d'exécution d'une application sur une architecture hétérogène », Master, Université de sud, Sfax (ENIS), juin 2004.

[Fol95] J. D.Foley, A. VanDam, S. K.Feiner, J.F.Hughes Philips, « Introduction à l'infographie », Editions Addison-Weslet, 1995.

[Gaj00] D. D.Gajski, J. Zhu, R. Dömer, A. Gerstlauer, S. Zhao., « SpecC: Specification Language and Methodology », Hardcover, ISBN: 978-0-7923-7822-8, p. 336, 2000.

[GAUT] GAUT - HLS Tool for DSP, http://lester.univ-ubs.fr: 8080/

[Gup95] R.Gupta, «A Co-Syntesis of Hardware and Software for Digital Embedded System», Kluwer Academic, 1995.

[Gup96] R. Gupta et G. De Micheli, « A co-synthesis approach to embedded system design automation» Design Automation for Embedded Systems, 1(1-2), 69–120, Juin 1996.

[Har96] J. Hartmanis, R. E. Stearns, «Algebraic structure theory of sequential machines», ISBN-10: 0130222771, édition Prentice-Hall, 1996.

[Her02] T. Hervé, J-Ph. Diguet, J-L. Philippe, «Estimations et métriques au niveau système pour la conception conjointe logiciel/Matériel (Co-design) », 17ème Colloque GRESTSI, Vannes, Juin 1999.

[Hom01] D. Hommais, « Une méthode d'évaluation et de synthèse des communications dans les systèmes intégrés matériel–logiciel », Thèse, Université Pierre et Marie Curie (LIP6), Septembre 2001.

[Ian02] C. Z. Ianculescu Alexandru, T. B. Kienhuis, E. Deprettere, «Solving Out of Order communication using CAM memory an implementation», Workshop on Circuits, Systems and Signal Processing (ProRISC), Utrecht, Netherlands, 2002.

[IBM02] IBM, The CoreConnect Bus Architecture, 2002, disponible à: http://www-3.ibm.com/chips/products/coreconnect.

[Ism94] T. Ben Ismail, M. Abid, K.O'Brien, A. A. Jerraya, «An approach for Hardware/Software Co-design», IEEE Int. Workshop on Rapid System Prototyoing, Grenoble France, 1994.

[Jeg99] C. Jego, E. Casseau, E. Martin. «Architectural Synthesis with Interconnection Cost Control», IFIP Tenth International Conference on Very Large Scale Integration, Netherlands, 1999.

[Jeg01] C. Jego, E. Casseau, E. Martin. «Architectural synthesis of digital signal processing applications dedicated to submicron technologies», ICECS, Malta, Italie, 2001.

[Jer05] A. Jerraya. «Long Term Trends for Embedded System Design», CEPA Workshop, Bruxels, Belgique, 2005.

[Kea03] M. Keating, P. Bricaud. «Reuse Methodology Manual for System-on-a-Chip Design», Kluwer Academic Publishers, 2003 (3ème édition).

[Keu00] K. Keutzer, S. Malik, A. Richard Newton, J. Rabaey, A. Sangiovanni-Vincentelli. «System-Level Design: Orthogonalization of Concerns and Platform-Based Design», IEEE transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 19, No. 12, Décembre 2000.

[Laj03] M. Lajolo, « IP-Based SOC Design in a C-based design methodology», IP Based SoC Design 2003, Grenoble, France, Octobre 2003.

[Lou04] Loukil K., Ben Cheikha H., « Conception d'accélérateurs pour la synthèse d'image 3d », projet de fin d'études, juin 2004, université du sud, Faculté des Sciences de Sfax, tunisie.

[Mic02] G. De Micheli «Network on Chip: a new Paradigm for System on Chip Design », Design Automation and Test in Europe (DATE) 2002, Paris, 2002.

[MOT] http://www.motorola.com/

[Nic02] G. Nicolescu, «Spécification et validation des systèmes hétérogènes embarqués», thèse d'habilitation, TIMA, Novembre 2002.

[Nil99] M. O'Nils, «Specification, Synthesis, and Validation of Hardware/Software Interfaces», thèse d'habilitation, département d'électroniques, institut royal de technologies, Sweden, 1999.

[OCP] Sonics Incorporation, «Open Core Protocol Specification 1.0», 2000, disponible à: http://www.sonicsinc.com

[Oua04] S. Oudjaout, D. Houzet, «Rapid integration of reusable functional IPs with SystemC VCI Adapters », Conférence Internationale en Microélectroniques (ICM 2004), Tunis, Tunisie, Décembre 2004.

[Pas98] R. Passerone, J. A.Rowson, A. Sangiovanni-Vincentelli, « Automatic Synthesis Interfaces between Incompatible Protocols», Design Automation Conference, San Francico, Californie, Etat Unis, 1998.

[PARA] The Paradise Design Environment, disponible à : http://www.clab.de/home/en/f-e-projects/paradise/

[Pet03] F. Pétrot, « intégration des systèmes matériel/logiciel», thèse d'habilitation, Université Pierre et Marie Curie, juin 2003.

[PIBus] Open Microprocessor System Initiative, «PI Bus VHDL Toolkit», Version 3.1.

[PixS06] PixelStreams, product brief, «Streaming Video Systems Design», 2006, http://www.celoxica.com/products/

[Pog99] F. Pogodalla, R. Hersemeule, P. Coulomb, «FastPrototyping: a system design flow for fast design, prototyping and efficient IP reuse», IEEE International Workshop on Hardware/Software Co-Design (CODES), Rome, Italie, pp. 69 -73, 1999.

[PRO] OCP Adoption Adds Value to Prosilog, disponible à, www.prosilog.com/Documents/ OCP_Newsletter_04-2003.pdf.

[Raw97] J. A. Rawson, A. Sangiovanni-Vincentelli, «Interface Based Design», Design Automation Conference, pp.178-183, 1997.

[Reg04] F. Regazzoni, M. Lajolo, «Interface Synthesis in Multiprocessing Systems-On-Chips», IP Based SoC Design, Grenoble, France, 2004.

[Rei00] R. A. Bergamachi, W. R.Lee, «Designing System-on-Chip Cores», Design Automation Conference, LosAngeles, Californie, Etat Unis, 2000.

[Rom96] K.V. Rompayey, D. Verkest, I. bolsens, H. De Man, «CoWare, A Design Evironnement for Heterogeneous Hardware/Software Systems». European Design Automation Conference, Genève, Switzerland, 1996.

[Sar87] R. Saracco and P. A. J. Tilanus, «CCITT SDL: Overview of the language and its applications», Computer Network and ISDN Systems, Vol.13 No.2, 1987.

[Sen95] O. Sentieys, J.Ph. Diguet, J.-L. Philippe, D. Chillet, «Gaut: A high level synthesis tool, dedicated to real time signal processing application». EURO- Design Automation Conference, Brighton, 1995.

[She04] A. Scherrer, T. Risset, A. Fraboulet, «Hardware Wrapper Classification and Requirements for On-Chip Interconnects», Signaux, Circuits et Systèmes, Monastir, Tunisie, Mars 2004.

[Smi98] J. Smith, G. deMicheli, «Automated Composition of Hardware Components», Design Automation Conf., Californie, Etat Unis, Juin 1998.

[SOC] SOCworks, Sonics. Disponible à: http://www.socworks.com

[SoCL] http://soclib.lip6.fr/

[SPIRIT06] SPIRIT_User_Guide_V1.1, 2006, disponible à www.spiritconsortium.org

[SYS] Open SystemC initiative (OSCI), www.systemc.org.

[Sys02] Open SystemC initiative (OSCI), «SystemC Version 2.0 User's Guide», disponible à: www.systemc.org.

[SYSV] An overview of SystemVerilog 3.1, disponible à: http://www.systemverilog.org

[Tur03] J. Turley, «Embedded Processors of Tomorrow», Embedded Systems Programming, ISBN 1578201209, 2003.

[VCI] On chip bus development working group, VSI Alliance, «Virtual Component Interface Standard (OCB 2.2.1) », 2001.

[Ver96] S. Vercauteren, B. Lin, H.De Man, « Constructing Application-specific Heterogeneous Embedded Architectures from Custom HW/SW Application», Design Automation Conference, New York, USA, 1996.

[Vid00] C. Vidal, « Tutoriel DOM et JDOM », disponible à : cyril@planetexml.com

[VSIA] Virtual Socket Interface Alliance, disponible à : http://www.vsi.org

[Wc297] http://fresh.t-systems-sfr.com/pc/src/misc/old/.warix/wc2pov27.zip.html

[Wol02] W. Wolf. N. Martinez, «Platform Based Design and Virtual Component Reuse», Design Automation and Test in Europe (DATE), Paris, France 2002.

[XML00] «XML 1.0 Specification », 2eme édition, W3C Recommandations, Octobre 2000, disponible à : http://www.w3c.org/XML.

[Yan04] Y. Paviot, «Partitionnement des services de communication en vue de la génération automatique des interfaces logicielles/matérielles », TIMA, Juillet 2004.

[Yve05] Y. Vanderperren, W. Dehaene, «UML 2 And Sysml: An Approach To Deal With Complexity In Soc/Noc Design», Design, Automation and Test in Europe, Mars 2005.

[Zit01] A. Zitouni, «Contribution à la synthèse de communication pour les systèmes distribués dans le cadre du co-design», Thèse; Université du centre, Faculté des sciences de Monastir, Juillet 2001.

[3DS] http://www.autodesk.fr/3dsmax

PUBLICATIONS PERSONNELLES

[Abb03a] F. Abbes, M. Abid, E. Casseau, «Spécification et conception de systèmes sur puce : SystemC et approches actuelles », Troisième Journée scientifiques des jeunes chercheurs en Génie Electrique et Informatique (GEI'03), Mahdia, Tunisie 16-19 Mars 2003.

[Abb03b] F. Abbes, E. Casseau, M. Abid, « Spécification et conception de systèmes sur puce avec SystemC, Etude de cas : le turbo-codage », Symposium en Architecture et Adéquation Algorithme Architecture (Sympaa03), La Colle Sur Loup 14-17 Octobre 2003, Nice France.

[Abb03c] F. Abbes, E. Casseau, M. Abid, « SoC Design Case Study Using SystemC Specifications», 15ème Conférence Internationale en Microélectroniques (ICM 2003), El Giza 9-11 Décembre 2003, Caire, Egypte.

[Abb04] F. Abbes, E. Casseau, M. Abid, P. Coussy, J.B.Legoff, « IP integration methodology for SoC design», 16ème Conférence Internationale en Microélectroniques (ICM 2004), 7-9 Décembre 2004, Tunis, Tunisie.

[Abb06a] F. Abbes, M. Abid, E. Casseau, «Interface Architecture Generation for IP Integration in SoC Design», Computer Engineering & Systems (ICCES'06), Egypt, Novembre 5-7, 2006.

[Abb06b] F. Abbes, E. Casseau, M. Abid, «IP Encapsulation: Approach and case study», IP Based SoC Design 2006, Grenoble, France – December 6-7, 2006.

ACRONYMES & ABRÉVIATIONS

API Application Programming Interface
ASIC Application Specific Integrated Circuit
CABA Cycle Accurate Bit Accurate
CAO Conception Assistée par Ordinateur
DCT Transformée en Cosinus Discrète
DMA Data Memory Access
DSP Digital Signal Processing
FFT Transformée de Fourrier Rapide
FIFO First In First Out
FPGA Field Programmable Gate Array
GAUT Générateur Automatique d'Unité de Traitement
GIC Générateur d'Interfaces de Communication
GOES Graphe d'Ordonnancement des Entrées/Sorties
GOESS Graphe d'Ordonnancement des Entrées/Sorties par Structure
GOS Graphe d'Ordonnancement aux entrées/sorties du Système
HDL High Description Language
IPERM IP Execution Requirement Model
IP Intellectual Property
IOCG Input Output Constraint Graph
MPSoC SoC multiprocesseur
MEF Machine à Etats Finis
NoC Network on Chip
OCB On Chip Bus
OCP Open Core Protocol
OS Operating System
PIO Peripheral Input Output
PiBus Peripheral Interconnect Bus
RAM Random Access Memory
ROM Read-Only Memory
RTL Register Transfer Level
RTOS Real time Operating System
SoC System on Chip
SHN Synthèse de Haut Niveau
SDL System Description Language
TLM Transaction level modelling
TDSI Traitement du Signal et de l'Image
VC Virtual component
VCI Virtual Component Interface
VHDL Very High Speed Integrated Circuit Hardware Description Language
VSIA Virtual Socket Interface Alliance
XML eXtensible Markup Language
UAL Unité Arithmétique et Logique
UT Unité de Traitement
UM Unité de Mémorisation
UCOM Unité de Communication
UD Unité de Duplication
SDFG Signal Data Flow Graph
UML Unified Modeling Language