Méthodologie et flot semi-automatique d'aide à la conception et à la validation des macro-cellules ASIC dédiées au traitement du signal

L. Tambour

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque
|__/__/__/__/__/__/__/__/__/__|

T H E S E

pour obtenir le grade de

DOCTEUR DE l'INPG

Spécialité : Microélectronique

préparée au laboratoire **TIMA** dans le cadre de
l'Ecole Doctorale Electronique Electrotechnique Automatique Télécommunications Signal

présentée et soutenue publiquement

par

Ludovic TAMBOUR

le 03 décembre 2003

Titre :

Méthodologie et flot semi-automatique d'aide à la conception et à la validation des macro-cellules ASIC dédiées au traitement numérique du signal

Directeur de thèse : Ahmed-Amine JERRAYA

Co-directeur : Nacer-Eddine ZERGAINOH

JURY

M.	Pierre GENTIL	, Président
M.	Eric MARTIN	, Rapporteur
M.	Frédéric PETROT	, Rapporteur
M.	Ahmed-Amine JERRAYA	, Directeur de thèse
M.	Nacer-Eddine ZERGAINOH	, Co-directeur
M.	Henri MICHEL	, Examinateur

Avant-propos

Cette thèse rentre dans le cadre d'une convention CIFRE entre ST Microelectronics (site de Crolles, Central R&D) et le laboratoire TIMA.

Les travaux décrits dans ce manuscrit ont été réalisés dans le groupe SLS du laboratoire TIMA (System Level Synthesis) sous la direction d'Ahmed Jerraya (Directeur de recherche CNRS) et de Nacer-Eddine Zergainoh (Maître de conférence à Polytech'Grenoble).

Au sein de ST Microelectronics, ces travaux ont été suivis par : Frank Ghénassia, responsable de l'équipe SysAr (System and Architecture) ; Henri Michel, responsable de l'équipe SHIVA (Signal processing Hardware Implementation and VAlidation) ; Pascal Urard, ingénieur architecte.

Remerciements

Je remercie ici du fond du cœur toutes les personnes qui par leur aide, leurs conseils, leur soutien, leur confiance mais aussi par la chaleur de leurs amitiés, m'ont permis de réaliser et d'aboutir ce travail de thèse. Je vous en serai éternellement reconnaissant pour cela.

Je tiens tout d'abord à exprimer mes sincères remerciements à Monsieur Ahmed-Amine Jerraya, directeur de recherche au CNRS, pour son accueil au sein de sa formidable équipe, ainsi que pour toutes les opportunités qui m'ont été permises d'évoluer intellectuellement durant ces trois années de thèse.

De même, je remercie Monsieur Nacer-Eddine Zergainoh, maître de conférence à l'université Joseph Fourier de Grenoble pour son encadrement, ses conseils toujours pertinents, sa confiance et sa patience.

J'exprime une très grande gratitude à Monsieur Henri Michel, responsable de l'équipe SHIVA de ST Microelectronics, pour son aide, son soutien et ses encouragements pendant tous les moments aussi bien calmes que difficiles de cette dernière année. Cela a été un grand privilège de travailler avec lui.

Je tiens à remercier Monsieur Pierre Gentil, responsable de l'école doctorale EEATS, pour m'avoir fait l'honneur de présider le jury. Je remercie également Monsieur Eric Martin de l'Université de Bretagne Sud et Monsieur Frédéric Pétrot du LIP6, de m'honorer de leur participation au jury en tant que rapporteurs et pour les conseils apportés à l'amélioration de ce travail.

Un grand remerciement à Monsieur Frank Ghénassia, responsable de l'équipe SysAr de ST Microelectronics, pour sa confiance en m'accueillant au sein de son équipe SysAr et pour m'avoir permis de bénéficier de l'expérience industrielle en matière de conception SOC.

Je souhaite également exprimer ma gratitude envers Monsieur Pascal Urard, ingénieur-architecte de ST Microelectronics, de m'avoir former puis conseiller sur la conception d'ASIC, d'avoir bénéficier de sa longue expérience.

Je remercie également Monsieur Bernard Courtois, directeur au laboratoire TIMA pour m'avoir accueilli dans son laboratoire.

Je remercie mes stagiaires Sébastien, Asra, Riadh, Haithem, Quentin et Abdel pour le travail formidable de développement qu'ils ont fait.

Je suis aussi reconnaissant aux thésards de l'équipe SLS, aux membres de l'équipe SysAr et de l'équipe SHIVA, avec lesquels j'ai eut le plaisir de travailler dans une ambiance fort sympathique et

qui n'ont jamais hésité à me faire profiter de leurs compétences scientifiques et industrielles. Merci également aux membres des autres équipes de TIMA et ST Microelectronics ainsi qu'au personnel administratif (en particulier, notre toujours souriante et chaleureuse secrétaire, Sonja) pour leur aimable collaboration.

Je ne pourrais pas terminer ces remerciements sans exprimer un remerciement venant du plus profond du cœur à tous mes amis qui m'ont toujours soutenu pendant ces trois dernières années et qui ont alors contribué à cette thèse à leur façon : aux amis de Grenoble (Eric, Fred, Mel, Tristan, Steph, Habiba…) ; aux amis et les rencontres du « barbecue du mardi soir » (Cec, Thierry, Florent, Meg, Hélène…) ; aux amis d'Auxerre (Vin's, Gwen, Richard, Yo, Estelle, Céline, Rémi, S'ril, Ben, Raph, Andréa, Grover, François, Marciou, Guigui, Salam, Fred, Etienne, Marine…) ; aux amis du laboratoire (Lobna, Wassim, Aimen, Fred, Amer, Lovic, Damien, Gabriela, Bogdan, Faiza, Dhanistha, Ferid…) ; aux amis de ST (Antoine, Rodrigue, JD, Hervé, Romain, Laurent P, Thierry, Fabrice, Phil, Etienne…) ; à Seb ; à Zoltan ; à Mathias ; à Fred de B ; un spécial remerciement à Imène ; et aux autres amis que j'oublierez par inadvertance dans cette longue liste. Pour tout ce que vous avez fait, je ne pourrez jamais assez vous remerciez.

Un dernier remerciement (mais non des moindres) à mes parents pour leur confiance, leur soutien et d'avoir fait de moi ce que je suis maintenant.

Résumé

Aujourd'hui, les macro-cellules ASIC dédiées au traitement du signal deviennent de plus en plus complexes, coûteuses en temps et efforts de conception. Ces macro-cellules seront ensuite assemblées avec d'autres composants IPs (e.g. processeurs, mémoires, média de communication, etc.) pour être intégrées dans un Système-sur-Puce (SoC, pour System-On-Chip). Les procédés de conception deviennent insuffisants pour conserver la maîtrise de la complexité (d'un point de vue aussi bien algorithmique qu'architectural) des nouvelles applications tout en respectant le temps de mise sur le marché.

Cette thèse est consacrée au problème de conception et de validation des macro-cellules ASIC dédiées au traitement du signal. Nous étudions et nous illustrons les possibilités d'une nouvelle méthodologie comme une alternative à la synthèse de haut niveau. Cette méthodologie se base sur l'assemblage de composants élémentaires (IPs) paramétrables et préconçus. Elle part d'une description fonctionnelle de l'application et produit le modèle RTL de l'architecture finale. Le principal problème d'une méthodologie de conception basée sur l'assemblage de composants IPs préconçus et pré-validés est que le modèle RTL de l'architecture finale peut avoir un comportement défectueux. Cela est dû à des retards induits par des contraintes d'implémentation. Nous présentons la formalisation de ce problème et proposons une méthode automatique de correction (dite correction de retard) pour le résoudre. Nous proposons deux algorithmes originaux qui garantissent des solutions optimales en latence et en surface. La faisabilité de l'approche et l'optimalité des solutions proposées sont démontrées mathématiquement. Des outils ont été développés pour transformer cette méthodologie en un flot semi-automatique. Nous illustrons l'efficacité de l'approche par l'expérimentation sur un exemple industriel : une chaîne de modulation numérique.

Abstract

Today, ASIC macro-cells dedicated to signal digital processing (DSP-ASIC macro-cells in follow) become more and more complex. Their design requires much time and money. These DSP-ASIC macro-cells will be assembled with other components to build the required Multiprocessor System-On-Chip (MP-SoC). The current design processes become insufficient to master the increasing complexity (in term of algorithm and architecture) while meeting the design time requirements.

This Ph-D thesis targets the problem of the design and the validation of DSP-ASIC macro-cells. We study the possibilities of a new methodology that could be used as an alternative to high level synthesis. This methodology is based on the assembly of basic generic predesigned components (IPs). It starts from a functional description of the application and produces an RTL model of the final architecture. The main problem of a component based assembly methodology is that the RTL model can have dysfunctions. This is due to delays introduced by implementation constraints. We present the formalization of this problem and propose an automatic correction method (called delay correction method) to resolve it. The viability of the method and the optimality of the solution are mathematically proved. Tools have been implemented to transform the methodology into a semi-automatic flow. We illustrate the approach efficiency by experiment on an industrial example: a digital modulation chain.

Table des matières

Liste des figures

Liste des tableaux et algorithmes

Chapitre I. Introduction

Sommaire :

I.1. Motivation

Les applications de traitement numérique du signal (*DSP* pour *Digital Signal Processing*) sont utilisées dans un large domaine d'activités : télécommunications, multimédia, réseaux, automobile, aérospatiale ou le grand public. L'incessante augmentation de la complexité des algorithmes DSP ainsi que du débit des données requièrent l'utilisation de macro-cellules ASIC (*Application Specific Integrated Circuit*) dédiées au traitement du signal. Ces macro-cellules seront ensuite assemblées avec d'autres composants (e.g. processeurs, mémoires, etc.) pour être intégrées dans un système monopuce multiprocesseurs complet.

Aujourd'hui, ces macro-cellules deviennent de plus en plus coûteuses en temps et efforts de conception. Elles sont souvent trop rigides pour suivre les modifications qui peuvent intervenir lors du cycle de conception du composant. De pair, la dynamique du marché des systèmes électroniques nécessite des temps de développement de plus en plus court. Les procédés actuels de conception deviennent insuffisants pour conserver la maîtrise de la complexité (d'un point de vue aussi bien algorithmique que architectural) des applications de traitement du signal tout en respectant le temps de développement.

Nous ciblons ici quatre problèmes fondamentaux liés à la conception de macro-cellules ASIC de traitement numérique du signal. Les procédés de conception demandent trop de temps et d'effort pour :

- *problème 1 - modéliser et valider la fonctionnalité* : une étude faite par ST Microelectronics montre que les erreurs de fonctionnalité ou de logique représentent plus de 43% des erreurs commises [MAG 03]. Ce pourcentage tend à augmenter avec l'accroissement de la complexité des applications DSP. Une correction au plus tôt de ces erreurs est donc nécessaire pour améliorer la fiabilité des composants produits.

- *problème 2 - explorer conjointement l'algorithme et l'architecture* : le travail de spécification de la fonctionnalité de l'application et celui de conception de l'architecture sont généralement décorrélées et réalisées par des personnes différentes. Or, des opportunités d'optimisation d'architecture sont perdues à cause de cette séparation entre l'exploration algorithmique et l'exploration architecturale [MAR 01].
- *problème 3 - préserver la qualité de l'architecture* : les architectures obtenues doivent être compétitives en terme de surface, performance et puissance de consommation pour pouvoir percer le marché. Par exemple, à ST Microelectronics, il est estimé qu'un outil ou un procédé de conception ne peut être accepté lorsqu'il produit une architecture de surface supérieure à 5% à celle d'une architecture faite manuellement.
- *problème 4 - concevoir des applications dérivées (flexibilité)* : les applications DSP sont en constante évolution. Or, les macro-cellules ASIC sont beaucoup trop rigides pour suivre cette évolution. La conception doit être quasiment reprise à zéro lorsqu'une application dérivée doit être conçue.

I.2. Objectif

L'objectif général de cette thèse est de proposer une méthodologie et un flot de conception pour les macro-cellules ASIC dédiées au traitement numérique du signal. Cette méthodologie et ce flot doivent permettre :

- une modélisation et une validation rapide de la fonctionnalité de l'application traitée.
- une exploration conjointe de l'algorithme et de l'architecture.
- la préservation de la qualité de l'architecture comme si elle avait été manuellement faite par un concepteur matériel.
- une conception rapide d'applications dérivées (flexibilité).

I.3. Sommaire de l'état de l'art des méthodologies et flots de conception pour macro-cellules ASIC

Beaucoup de travaux ont proposé des méthodologies et outils de conception permettant de maîtriser la complexité des applications DSP.

Les outils de synthèse d'architecture (plus étudiés en détails dans le chapitre II) [GAS 94][JER 96][KU 92][LEE 96] synthétisent une architecture RTL à partir d'un modèle comportemental ou fonctionnel. L'exploitation d'un modèle d'entrée au plus haut niveau d'abstraction permet une exploration, modélisation et validation algorithmiques rapides et efficaces. Mais, l'architecture générée reste de médiocre qualité et difficile à prévoir [SUG 00]. Les outils passent par trop d'étapes de raffinement pour pouvoir comprendre comment modifier la description initiale en vue d'améliorer la qualité de l'architecture générée. Souvent l'architecture proposée par l'outil n'est pas toujours compatible avec les règles industrielles de synthèse RTL (trop de niveau de hiérarchie, instanciation de « designware », code RTL illisible et non réutilisable…).

Nous pouvons également citer les flots de conception basés sur le langage C tels que HardwareC [KU 88], Handel-C [STO 98], SpecC [SPE 03] et SystemC [SYS 03]. Les langages d'entrées de ces flots offrent des concepts similaires aux langages de description matérielle, mais faisant abstraction de certains détails d'implémentation. Ceci facilite leur utilisation et simplifie la modélisation. Leur handicap est le niveau d'abstraction qui reste encore trop proche du RTL. L'architecture RTL générée est fortement conditionnée par la description initiale. L'environnement de validation n'est pas suffisamment souple pour permettre une modélisation et une exploration algorithmique rapide.

Finalement, le projet SHAFFT [DAS 01][DAV 02] de Berkeley et DSP builder [DSP 01] de Altera proposent chacun un flot de conception et de synthèse automatique en partant d'une description Simulink [SIM 00]. Certes, ces deux flots exploitent un environnement de haut

niveau, mais qui est utilisé pour une modélisation RTL, puisque le point d'entrée reste un modèle RTL schématique. Par conséquent, la modélisation d'entrée est plus coûteuse en temps de modélisation et de validation qu'un modèle fonctionnel. En plus, l'architecture est figée par le modèle d'entrée et restreint donc les possibilités d'exploration architecturale.

	Modélisation et validation fonctionnelle	Exploration conjointe algorithme / architecture	Qualité de l'architecture résultante	Conception d'applications dérivées (flexibilité)
Synthèse d'architecture	+++	+++	+	+++
Flot basé sur des langages de programmation	++	++	+	++
Flot basé sur une description graphique	++	+	+++	++
Conception manuelle	+	+	+++	+

Tableau 1. Comparaison des outils et méthodologies de conception existante

Le tableau 1 résume les points forts et les points faibles de chaque méthodologie ou flot présentés précédemment vis-à-vis des quatre problèmes ciblés. Ce tableau montre que les travaux précédents proposent des méthodologies de conception permettant de répondre à certains des problèmes de la conception des applications DSP mais, n'ont pas encore recouvert la globalité du problème. En effet, la plupart des approches citées ne peuvent satisfaire un compromis entre modélisation et validation algorithmiques efficaces d'une part, et exploration architecturale rapide et qualité de l'architecture générée d'autre part.

I.4. Contributions

La contribution générale de ce travail est la proposition d'une méthodologie de conception pour macro-cellule ASIC-DSP. Cette méthodologie part d'une description fonctionnelle. Elle est basée sur :

- l'utilisation de blocs de base (DSP-IP) de traitement du signal, préconçus, paramétrables, et décrits à la fois au niveau fonctionnel et RTL.
- l'assemblage automatique des IP-RTL suivant l'assemblage défini par le modèle fonctionnel d'entrée.

Comme nous le verrons dans le chapitre III, cette méthodologie permet de résoudre les quatre problèmes ciblés.

Ce travail peut être décomposé en trois contributions principales : (1) une étude comparative des outils commerciaux actuels de synthèse d'architecture ; (2) la proposition d'une méthodologie de conception et de validation pour macro-cellules ASIC DSP ; (3) la formalisation et la proposition d'une méthode (dite *méthode de correction de retard*) pour la résolution des problèmes de différences de comportement lors du passage d'un assemblage d'IP fonctionnels vers un assemblage d'IP RTL ; (4) l'implémentation de la méthodologie en un flot semi-automatique de conception.

Chaque contribution sera présentée par la suite.

I.4.1. Etude comparative des outils commerciaux actuels de synthèse d'architecture

Principaux travaux concernant la conception des ASIC, une étude a été menée sur les outils commerciaux actuels de synthèse d'architecture (chapitre II). L'objectif est ici de comprendre comment la synthèse d'architecture peut être insérée dans une méthodologie complète de conception et de validation pour macro-cellules ASIC DSP. Cette étude compare trois outils à des deux types de réalisation manuelle. Cette étude est à la fois une étude théorique et pratique par l'expérimentation de ces trois outils sur deux exemples industriels. De cette étude, nous dressons les problèmes communs à la synthèse d'architecture permettant de comprendre pourquoi elle n'a pas encore été acceptée par le monde industriel.

I.4.2. Proposition d'une méthodologie de conception et de validation pour macro-cellules ASIC DSP

Puisque la synthèse d'architecture n'a pas eut le succès escompté, nous proposons alors une méthodologie et un flot semi-automatique de conception et de validation pour macro-cellules ASIC DSP (chapitre III). Cette méthodologie débute par une description fonctionnelle et se fonde sur l'assemblage automatique de blocs de base (IP) préconçus et paramétrables. Cette méthodologie a été proposée en vue de résoudre les quatre problèmes cités en § I.1.1. Ne faisant pas d'hypothèse sur le moyen de conception des IP, ces IP peuvent provenir du résultat de la synthèse d'architecture ou d'une réalisation manuelle.

I.4.3. Méthode de correction de retard lors du passage d'un assemblage d'IP fonctionnels vers un assemblage d'IP RTL

Comme il sera expliqué dans le chapitre III, le principal problème d'un flot d'assemblage est que le passage d'un assemblage d'IP fonctionnels vers un assemblage d'IP RTL peut produire une différence de comportement (i.e. les valeurs numériques de sortie du modèle RTL ne sont plus les mêmes que les valeurs numériques de sortie du modèle fonctionnel). Notre contribution est ici, la formalisation du problème et la proposition d'une méthode automatique de correction (dite *correction de retard*) pour résoudre ce problème. Cette méthode recherche une solution optimale en latence ou en surface. La faisabilité de la méthode ainsi que l'optimalité des solutions trouvées sont démontrées mathématiquement.

I.4.4. Implémentation d'une méthodologie de conception en un flot semi-automatique de conception

Un ensemble d'outils indépendants a été développé pour transformer la méthodologie proposée en un flot semi-automatique de conception et de validation (chapitre V). Ces outils sont regroupés sous une interface graphique pour assurer une cohérence, homogénéité et convivialité dans leurs utilisations.

I.5. Plan

Le **chapitre 2** présente une étude des outils de synthèse d'architecture actuels. Nous présentons un historique et les principes généraux de la synthèse d'architecture. Une étude comparative de trois outils industriels de synthèse d'architecture est réalisée par expérimentation sur des exemples industriels. De cette étude, les problèmes communs aux outils de synthèse d'architecture sont mis en évidence.

A partir de cette étude, le **chapitre 3** présente la méthodologie proposée alternative à la synthèse d'architecture. Après avoir introduit les concepts préliminaires, nous décrivons la méthodologie étape par étape. Cette méthodologie est expérimentée sur un exemple industriel concret. Dans ce chapitre, nous verrons que le passage d'un assemblage d'IP fonctionnels vers un assemblage d'IP RTL peut entraîner une différence de comportement (i.e. production de sorties numériques différentes entre les deux niveaux d'abstraction).

Le **chapitre 4** est dédié à la formalisation et à la résolution du problème de différence de comportement. Il introduit la méthode de correction de retard dans le cas des assemblages sans boucle et détaille les différentes étapes. La correction est réalisée en insérant des registres entre les IP RTL. La faisabilité de la méthode ainsi que l'optimalité des solutions trouvées sont démontrées mathématiquement (démonstrations en annexe A). La méthode de correction est appliquée sur un exemple. Le coût de la correction par insertion de registres est alors comparé au coût d'une correction par modification de la FSM, autre moyen de correction. La problématique des assemblages contenant des boucles est également abordée. Le formalisme utilisé pour les applications sans boucle employé sera adapté au cas des boucles et un nouvel algorithme sera proposé.

Le **chapitre 5** présente les outils développés permettant d'automatiser la méthodologie proposée. Il présente la forme intermédiaire utilisée (COLIF) et décrit le synoptique général

du flot. Chaque outil ainsi que l'interface graphique unifiant tous les outils est également présenté.

Le **chapitre 6** conclut ce mémoire et dresse les perspectives de ce travail.

Chapitre II. Etude comparative des outils commerciaux de synthèse d'architecture

Résumé : malgré vingt ans de recherche intensive, la synthèse d'architecture n'a pas encore été acceptée par le monde industriel. Dans ce chapitre, nous essayons de comprendre les causes de cet échec grâce à une étude comparative de trois de ces outils. Les outils sont évalués par expérimentation sur deux exemples industriels actuels : un filtre interpolateur et un filtre SRC. Les résultats obtenus par synthèse d'architecture sont comparés à deux modèles RTL conçue manuellement. De cette étude, nous dressons les problèmes communs à ces outils faisant que la synthèse d'architecture n'a pu trouvée sa place dans un contexte industriel.

Note importante : la partie II.3 (expérimentations sur des outils de synthèse d'architecture) a été réalisée à ST Microelectronics et est couverte par des accords de confidentialité avec les vendeurs d'outils CAO. Ainsi, elle n'a pu être intégrée dans cette thèse. Les personnes intéressées pourront se procurer cette partie à partir de janvier 2005.

Sommaire :

Note importante : la partie II.3 (expérimentations sur des outils de synthèse d'architecture) a été réalisée à ST Microelectronics et est couverte par des accords de confidentialité avec les vendeurs d'outils CAO. Ainsi, elle n'a pu être intégrée dans cette thèse. Les personnes intéressées pourront se procurer cette partie moyennant un accord de non divulgation.

II.1. Introduction

Un moyen de maîtriser la complexité croissante des applications de traitement du signal est d'élever le niveau d'abstraction du modèle d'entrée jusqu'au niveau fonctionnel ou comportemental. Le modèle fonctionnel ou comportemental [VSI 98][ZER 02] est un modèle exécutable décrivant la fonctionnalité de l'application et faisant abstraction des détails d'implémentation. Leur but est de valider que la fonctionnalité décrite satisfasse les contraintes fonctionnelles en terme de fonctionnalité, de qualité de signal et de quantification.

Des méthodologies et outils ont été développés dans le milieu académique et industriel pour synthétiser un modèle fonctionnel ou comportemental en une réalisation RTL. Ce passage d'un haut niveau d'abstraction vers le RTL est appelé *synthèse d'architecture* ou *synthèse de haut niveau* dans la littérature [JER 96][GAS 94][ELL 99][CES 02]. Initialement, les objectifs visés par la synthèse d'architecture sont : un court cycle de développement, une estimation des performances dès le haut niveau, un meilleur facteur de réutilisabilité, l'indépendance à la technologie, une amélioration du compromis surface / latence / débit / puissance.

Des recherches intensives sur la synthèse d'architecture ont été menées depuis plus de vingt ans. Cinq outils de synthèse d'architecture sont devenus des outils industriels. Bien que les perspectives de la synthèse d'architecture semblaient prometteuses, ces outils n'ont jamais eut le degré d'acceptation que la synthèse RTL a obtenue.

II.1.1. Objectif et contribution

Les objectifs de ce chapitre sont :

- comprendre pourquoi la synthèse d'architecture n'a pas été encore acceptée par le monde industriel.
- savoir comment insérer un outil de synthèse d'architecture dans un flot de conception et de validation pour macro-cellules ASIC dédiées au traitement numérique du signal.

Dans ce chapitre, nos contributions sont l'évaluation et la comparaison des trois principaux outils industriels de synthèse d'architecture par l'expérimentation sur des exemples industriels et de dresser les problèmes communs à la synthèse d'architecture. Les applications que nous visons sont les applications de traitement numérique du signal. Les résultats obtenus par synthèse d'architecture seront comparés à deux réalisations RTL faites manuellement. Les exemples utilisés sont un filtre interpolateur par deux [WAN 92] et un filtre SRC (SRC pour Sample Rate Converter).

II.1.2. Plan

Ce chapitre est organisé comme suit. Dans la section 2, nous présentons un historique et les principes généraux de la synthèse d'architecture. Dans la section 3, une étude comparative de trois outils industriels de synthèse d'architecture est réalisée. Cette étude est basée sur l'application de ces outils sur deux exemples industriels. De cette étude, les problèmes communs de la synthèse d'architecture sont mis en évidence dans la section 4. La section 5 conclut ce chapitre et dresse les perspectives de cette étude.

II.2. Introduction à la synthèse d'architecture

La synthèse d'architecture peut être classée en trois générations d'outils [CES 00b][SUG 00].

Les premiers travaux sur la synthèse d'architecture ont commencé vers 1970. Les outils de synthèse de première génération ont été mis au point principalement par des chercheurs issus de la recherche sur l'architecture des ordinateurs [BAR 73][MAR 79][GRA 85]. L'un des premiers outils de synthèse d'architecture fut CMUDA développé par les chercheurs de l'université de Carnegie-Mellon. Le bénéfice le plus important des travaux sur la synthèse de première génération est la formalisation et la définition des étapes de la synthèse d'architecture : ordonnancement, allocation, assignation d'unités fonctionnelles, partitionnement et génération d'interface. A cette époque, la synthèse RTL n'étant qu'à ces débuts, les outils de synthèse d'architecture réalisaient aussi les étapes de synthèse logique et celles de la réalisation physique [SOU 83].

Vers 1980, la seconde génération d'outils (e.g. CALLAS [BIE 92], HAL [PAU 86]) se caractérise par des domaines d'application restreints et spécialisés ainsi que par la génération d'une architecture composée d'une unité de contrôle (contrôleur ou FSM) et d'un chemin de données. Le flux de synthèse est réduit à un nombre plus limité d'étapes telles que l'ordonnancement, l'allocation et l'assignation des unités fonctionnelles. Plus d'une centaine d'outils de cette génération ont été publiés et visent une large variété de domaines d'applications : DSP, contrôleurs, circuit de communication. L'interdépendance entre les étapes de synthèse étant forte, toutes les possibilités de l'ordre des étapes ont été étudiées. Le lien étant faible entre ces outils et la synthèse RTL devenue de plus en plus puissante, ces outils n'ont pu s'adapter aux nouvelles capacités des outils de synthèse RTL [STO 94].

La troisième génération d'outils de synthèse a commencé en 1994. Cette génération essaie d'exploiter entièrement les capacités de la synthèse RTL dont l'évolution a été accélérée par son utilisation intensive en milieu industriel. Elle se caractérise par des modifications appliquées au flux traditionnel de synthèse d'architecture. En 1994, VOTAN [WHE 94] introduit une nouvelle approche d'ordonnancement considérant la synthèse comme une

problématique de transformation de code au niveau comportemental suivi de la synthèse RTL. Néanmoins, elle a été abandonnée avant même d'avoir été validée par des applications complexes et réelles. Hiasynth [BER 99] propose une approche radicalement différente par rapport aux outils traditionnels en intégrant synthèse d'architecture et synthèse RTL dans le même flux de conception. Au sein de cet outil, la synthèse RTL est utilisée pour effectuer les tâches de la synthèse d'architecture basées sur un nouveau modèle de conception nommé *graphe de réseau comportemental* (*Behavioral Network Graph*). L'outil RapidPath [LAG 90] accepte comme entrée une description comportementale avec l'exactitude du cycle. Le style d'écriture de ces descriptions peut être considéré comme une extension du sous-ensemble pour la synthèse RTL. Une dernière voie en cours d'exploration est d'ajouter des contraintes supplémentaires pour guider l'outil lors des étapes de synthèse. Par exemple, l'outil UGH (*User Guided High Level Synthesis*) [AUG 97] ajoute à la description d'entrée une spécification simplifiée du chemin de données (notamment, pas de taille de signaux, ni de multiplexeurs, ni de constantes). Le but est alors de contraindre l'outil à atteindre le chemin de données souhaité par l'utilisateur.

Aujourd'hui, les outils industriels de synthèse d'architecture appartiennent à la seconde génération. Bien que plus d'une centaine d'outils académiques furent développés, seulement cinq outils ont été portés vers l'industrie par les vendeurs CAD : ART Designer [ART 01] de Frontier Design, CoCentric SystemC Compiler [COC 00] de Synopsys, PrecisionC [GUO 03] et Mentor Graphics, Protocol Compiler de Synopsys, Visual Architect de Cadence. La plupart de ces nouveaux outils sont une évolution de précédents outils industriels. Par exemple, PrecisionC est issu de Monet alors que CoCentric SystemC Compiler vient de Behavioral Compiler. Durant les cinq dernières années, la principale évolution de ces cinq outils est le changement de la description d'entrée. La description comportementale d'entrée en VHDL ou Verilog a été abandonnée au bénéfice d'une description fonctionnelle en C [ART 01][GUO

03][AUG 97] ou en C++ [GUO 03] ou d'une description comportementale en SystemC 1.0 [SYS 00]. Le bénéfice est alors une description d'entrée plus simple à modéliser et plus rapide à la simulation. Mais, fondamentalement, les algorithmes de synthèse employés n'ont pas significativement évolués pour appartenir à la troisième génération.

II.2.1. Généralité sur la synthèse d'architecture

Les outils actuels industriels de synthèse d'architecture appartiennent à la seconde génération. Pour comprendre pourquoi ces outils ont eut peu d'acceptation dans le monde industriel, il est nécessaire de comprendre les principes généraux de la synthèse d'architecture de seconde génération.

La synthèse d'architecture est une méthodologie de conception partant d'une description réalisée au niveau comportemental ou fonctionnel et permettant d'obtenir une architecture décrite au niveau RTL. L'entrée de la synthèse d'architecture est une description comportementale ou fonctionnelle décrite sous forme algorithmique. Sa principale sortie est une architecture RTL répondant à la spécification initiale et satisfaisant les contraintes d'implémentation imposées par le concepteur. Ses autres sorties sont différents rapports et schématiques d'architecture pour le raffinement de l'architecture générée. L'architecture résultante est décrite au niveau RTL en VHDL ou Verilog. Un ensemble d'algorithmes de synthèse raffine progressivement la description d'entrée en une architecture cible. Typiquement, ces étapes de synthèse sont *l'élaboration* [MAR 86] (appelée aussi *compilation*), *l'ordonnancement* [ZEM 86][GOO 87][ROM 92][GOO 89][PAU 87], *l'allocation de ressource* [PAU 87], *l'assignation de ressource* et *la génération de code* (figure 1). En particulier, l'efficacité d'un outil de synthèse d'architecture dépend du choix de la représentation intermédiaire (étape d'élaboration), de l'algorithme d'ordonnancement et du type d'architecture générée.

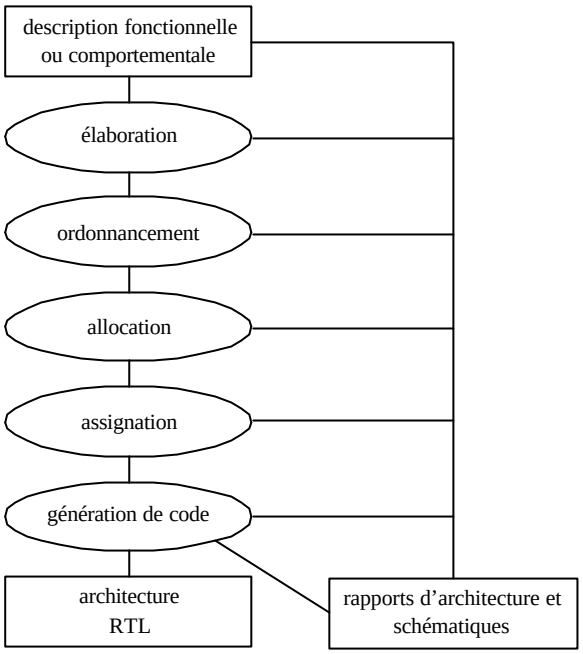


Figure 1. Flot de synthèse d'architecture

II.2.1.1. Description d'entrée : fonctionnelle ou comportementale

La description d'entrée est une description fonctionnelle ou comportementale suivant l'outil utilisé. Elle décrit la fonctionnalité d'un système sans décrire une implémentation spécifique. Une description fonctionnelle modélise la fonctionnalité de l'application traitée sous une forme purement algorithmique et séquentielle. Une description comportementale ajoute la notion de parallélisme et de concurrence. En effet, elle se constitue d'un ensemble de processus concurrents et communicants via des ports et des signaux. Chaque processus est décrit sous forme algorithmique et séquentielle. Des instructions de synchronisation (e.g. fonction *wait* en VHDL) assurent le déterminisme et la cohérence entre les processus concurrents.

Le format des signaux est spécifiée en virgule fixe ou en représentation hybride [KED 98]. Dans une représentation hybride, les variables ayant un rôle pertinent dans la quantification sont représentées en virgule fixe alors que les autres sont laissées en virgule flottante (i.e.

avec un nombre indéterminé de bits). Durant le processus de synthèse d'architecture, le format des variables laissées en virgule flottante sera calculé par une approche dite *interpolative* [KED 98] (ou encore *conservative*).

La description d'entrée est généralement supportée par un langage de programmation (e.g. C [ART 01], C++, Pascal [TRI 88] ou Ada [GIR 84]) ou par un langage de description matériel (*HDL* pour *Hardware Description Langage*) (e.g. VHDL [CAM 89a], SystemC [COC 00], DSL [CAM 89b] ou MIMOLA [ZIM 80]). La plupart des outils de synthèse utilise des langages de type procéduraux. Ces langages décrivent la manipulation de données en termes d'assignation de variables qui conservent leurs valeurs tant qu'elles ne sont pas sur-écrites. Certains outils ont aussi expérimenté différents types de HDL non procéduraux tels que LISP [JOH 84] ou Prolog [AOY 85].

II.2.1.2. L'élaboration de la description d'entrée

L'élaboration est la première étape de la synthèse d'architecture. Elle consiste à transformer la description d'entrée en une représentation interne. La majorité des outils de synthèse d'architecture reposent sur une représentation interne nommée *graphe de flux de donnée et de contrôle* (*CDFG*). Cette approche est utilisée par plusieurs outils de synthèse tels que Behavioral Compiler [KNA 96], HIS [CAM 91], HERCULES [MIC 88], CALLAS [BIE 92], etc. D'autres représentations internes existent et sont plus spécifiques à certains outils de synthèse : MIMOLA [MAR 86] utilise une représentation arborescente similaire aux arbres syntaxiques des compilateurs logiciels ; CATHEDRAL [NOT 91] a conçu une représentation appelée *graphe flux de signaux* dérivée directement du langage Silage [HIL 85].

Pendant la construction de la représentation interne, des pré-optimisations peuvent être effectuées. Ces optimisations, connues d'ailleurs du monde de la compilation des langages de programmation [AHO 86] [STO 94], permettent de s'affranchir des éléments relatifs au

langage d'entrée. Ces optimisations sont l'élimination du code mort, la propagation des constantes, la simplification des expressions, la pré-évaluation des conditions, l'expansion des macro, etc.

II.2.1.3. L'ordonnancement

L'ordonnancement est le partitionnement de la description d'entrée en sous graphes dont l'exécution peut être effectuée en un seul cycle (appelé *cycle de contrôle*) au niveau RTL. Ces cycles de contrôle correspondent aux transitions du contrôleur résultant qui peuvent comprendre plusieurs opérations s'exécutant en parallèle. L'ordonnancement essaye de produire une architecture de latence la plus petite suivant les ressources disponibles.

Les instructions d'attente de la description comportementale spécifient des états de contrôle implicites imposés par le concepteur. Le traitement exécuté entre deux instructions successives d'attente est appelé *pas d'exécution* (*execution thread* en anglais). Pour une description fonctionnelle, ces pas d'exécution se retrouvent au niveau des coupures dans le séquencement des données (i.e. le contenant d'une variable sera utilisé à prochaine exécution de la description). Au niveau de l'algorithme d'ordonnancement, plusieurs stratégies peuvent être envisagées pour le fractionnement des pas d'exécution dans la fréquence d'horloge impartie. La plupart des outils de synthèse fournissent trois modes d'ordonnancement : le *mode en un cycle*, le *mode des macro-états* et le *mode des états comportementaux*.

- *le mode en un cycle (cycle mode* en anglais*)* : dans ce mode, chaque pas d'exécution du modèle d'entrée est ordonné en un seul cycle de contrôle dans le modèle RTL. Ce mode impose des restrictions strictes sur le style d'écriture de la description d'entrée : le pas d'exécution ne peut contenir des boucles infinies ou dépendantes du signal d'entrée ; les opérations multi-cycles ne sont pas permises ; les traitements doivent être suffisamment simples pour pouvoir être réalisables en un cycle d'horloge. Le

comportement du modèle RTL est très proche de celui du modèle initial. Ce mode permet alors une vérification et une insertion aisées du modèle RTL avec d'autres composants. Par contre, ce mode laisse peu de liberté à la synthèse pour une large exploration d'architecture.

- *le mode des macro-états (superstate mode* en anglais*)* : chaque pas d'exécution est décomposé en plusieurs cycles d'horloge dont le nombre est fixé durant la synthèse. La seule restriction sur le style d'écriture du modèle d'entrée est l'interdiction des boucles infinies ou dépendantes du signal d'entrée. Dus aux cycles ajoutés, le comportement du modèle RTL diffère de celui du modèle d'entrée. Cela implique une vérification et une insertion du modèle RTL plus complexe qui doit tenir compte des cycles supplémentaires. Ce mode permet une plus large exploration d'architecture que le mode précédent.
- *le mode des états comportementaux (behavioral mode* en anglais) : chaque pas d'exécution est réalisé par une FSM dont les transitions peuvent être exécutées en un seul cycle d'horloge. Ainsi, le résultat de l'ordonnancement est généralement très complexe, l'exécution des transitions étant souvent conditionnelle. Ce mode laisse une très grande liberté dans le style d'écriture du modèle d'entrée car n'imposant aucune restriction. Les possibilités d'exploration d'architecture deviennent très étendues. En contrepartie, le comportement du modèle RTL devient très éloigné du modèle d'entrée. L'avantage de ce mode se manifeste lorsque l'application contient des traitements complexes tels que des boucles imbriquées ou des traitements conditionnels.

La décomposition des pas d'exécution du modèle d'entrée en plusieurs cycles d'horloge implique généralement des changements du comportement des entrées et des sorties du modèle RTL. Lors de l'ordonnancement, ces transformations nécessitent de fixer les stratégies

pour les ordonnancements manipulant des entrées et sorties. Deux classes de stratégies sont connues :

- *entrées/sorties fixes* : ce mode préserve l'ordre des opérations des entrées/sorties entre elles et vis-à-vis des instructions de synchronisation.
- *entrées/sorties flottantes* : ce mode ne préserve plus l'ordre des opérations des entrées et sorties. Les opérations manipulant les entrées et sorties sont ordonnées de la même manière que les autres opérations : elles sont ordonnées de façon à ce que le résultat de la synthèse satisfasse les contraintes de dépendances de données tout en minimisant le temps de calculs. Bien que ce mode permette un très large espace d'exploration, le comportement du modèle RTL devient extrêmement éloigné de la description d'entrée. La validation et l'intégration du modèle RTL sont complexifiées en conséquence.

Pour calculer le temps de propagation d'un opérateur, deux approches sont employées par les outils : la première approche suppose que le temps de propagation d'une ressource est d'une période d'horloge ; la seconde approche détermine les temps de propagation des différents opérateurs à partir de la technologie cible.

C'est l'efficacité de l'étape d'ordonnancement qui a le plus d'influence sur la qualité de l'architecture générée parce que le nombre de cycles de contrôle et le parallélisme des opérations y sont fixés. Le problème d'ordonnancement sous contrainte de ressources est connu comme un étant un problème NP complexe dans le domaine de l'optimisation combinatoire. Une solution optimale ne peut être envisagée en un temps raisonnable et les algorithmes d'ordonnancement doivent alors trouver une solution proche de la solution optimale. Vu la complexité de cette tâche, des centaines d'algorithmes ont été publiés bien que la plupart d'entre eux puissent être ramenés à quelques algorithmes de base. Une étude approfondie des algorithmes d'ordonnancement se trouve dans [JER 96][RAH 97]. Ces

algorithmes peuvent être classés en quatre familles : les *algorithmes transformationnels* améliorent une solution courante en appliquant des transformations successives ; les *algorithmes PLE* (*Programmation Linéaire en nombre Entier*) reformulent le problème d'ordonnancement comme un problème de programmation linéaire ; les *algorithmes neuronaux* modélisent le problème d'ordonnancement par un réseau de neurones trouvant une solution par auto-réorganisation ; les *algorithmes itératifs* répartissent les opérations dans le temps les unes après les autres suivant un critère de priorité sur chacune d'elle. Cette dernière famille contient plusieurs sous-classes d'algorithmes : *l'ordonnancement au plut tôt* (*ASAP* pour *As Soon As Possible*), *l'ordonnancement au plus tard* (*ALAP* pour *As Late As Possible*), *l'ordonnancement par liste, l'ordonnancement à priorité sur le chemin critique* et *l'ordonnancement basé sur les distributions*.

L'ordonnancement est fortement dépendant de la tâche d'allocation : restreindre le nombre de ressources diminue la surface mais augmente le débit (et réciproquement). Le choix d'amélioration de la surface ou du débit est laissé au jugement du concepteur et peut être imposé à l'outil via des options ou des pragmas. La forte interdépendance entre l'ordonnancement et l'allocation justifie certaines approches de synthèse où ces deux tâches ne sont pas considérées comme des tâches séparées. Par exemple, CATHEDRAL [NOT 91] ou GAUT [MAR 93] exécutent l'allocation avant l'ordonnancement. D'autres, comme Behavioral Compiler [KNA 96], AMICAL [JER 96], CALLAS [BIE 92], EASY [STO 91], HAL [PAU 86], HIS [CAM 91] ont choisi l'ordre inverse.

II.2.1.4. L'allocation

L'allocation fixe le nombre de ressources nécessaires pour la réalisation du modèle RTL décrit par la description d'entrée. Le nombre de ressources est déterminé en fonction des résultats de l'étape d'ordonnancement. Cette étape détermine le nombre et le type des unités fonctionnelles (e.g. additionneur, multiplieur), des unités pour les stockages de données (e.g.

registre, mémoire) et des unités de communication (e.g. multiplexeur, bus). Le nombre de ressources peut être restreint par la structure, la complexité du chemin de données ou par l'utilisateur. Les algorithmes d'allocation comprennent souvent une étape permettant de déterminer les unités fonctionnelles devant exécuter chacune des opérations de la description initiale. Cette étape est appelée *l'association des unités fonctionnelles* (ou *binding* en anglais). Suivant la complexité des unités de la bibliothèque cible, plusieurs solutions peuvent être envisagées. Par exemple, pour l'opération de la multiplication, plusieurs types de multiplieurs peuvent exister avec des caractéristiques différentes. En particulier, les caractéristiques en temps de propagation peuvent être différentes. Ainsi, le choix d'une unité peut influencer l'étape d'ordonnancement si l'allocation est faite avant l'ordonnancement. D'un autre coté, l'étape d'ordonnancement peut restreindre le choix de l'unité fonctionnelle lors de l'allocation si l'ordonnancement est réalisé avant l'allocation.

II.2.1.5. L'assignation

L'assignation est l'affectation d'une ressource du modèle d'entrée à une unité fonctionnelle d'une librairie spécifique. Cette étape peut être parfois réalisée en même temps que l'étape d'allocation.

II.2.1.6. La génération de code et les architectures RTL générées

Durant l'étape de génération de code, une architecture est modélisée au niveau RTL. Le type de modélisation RTL varie suivant l'outil utilisé. Typiquement, dans les outils de seconde génération, la partie contrôle est séparée de la partie flot de données. La partie contrôle impose le bon séquencement des signaux dans le flot de données. Le contrôle est réalisé soit par une machine d'états finis ou soit par un microcontrôleur. Le flot de donnée est représenté soit par une netlist structurelle d'unités fonctionnelles ou soit par des affectations concurrentes. Au niveau du protocole de sortie, deux stratégies sont possibles : dans le

premier cas, le protocole d'entrée est juste un fil physique et celui de sortie est une sortie bufférisée ; dans le second cas, l'outil insert automatiquement un protocole de sortie spécifique. Ce dernier est généralement un *protocole poignée-de-main* (*handshake protocol* en anglais).

L'architecture générée est décrite en VHDL ou Verilog. Le style d'écriture dépend de son utilisation : implémentation ASIC ou prototypage rapide sur FPGA. Il est réalisé pour être compatible avec les règles de codage supportées par les outils actuels de synthèse RTL tels que Design Compiler ou Leonardo Spectrum.

II.2.1.7. Les rapports et les schématiques d'architecture

Pour analyser et raffiner l'architecture générée, des rapports et des schématiques d'architecture sont généralement fournis après synthèse. Classiquement, ils comprennent des rapports d'architecture tels qu'une liste du matériel utilisé, un rapport de timing et un rapport de surface. Les schématiques incluent une visualisation de l'architecture, une visualisation de la machine d'états finis et un tableau de Gant. Le tableau de Gant schématise l'ordonnancement des différents opérateurs arithmétiques pour chaque cycle d'horloge et les dépendances entre ces opérateurs.

II.3. Etude comparative de trois outils de synthèse d'architecture

Note importante : cette partie a été réalisée à ST Microelectronics et est couvert par des accords de confidentialité avec les vendeurs d'outils CAO. Ainsi, elle n'a pu être intégrée dans cette thèse. Les personnes intéressées pourront se procurer cette partie à partir de janvier 2005.

Dans cette section, nous avons réalisé une étude comparative de trois outils de synthèse d'architecture. Ces outils sont évalués par expérimentation sur deux exemples industriels

courants : un filtre interpolateur et un filtre SRC. Les résultats obtenus par synthèse d'architecture sont comparés à deux réalisations RTL conçues manuellement. Cette expérimentation a été faite en appliquant le flot de synthèse complet depuis le modèle d'entrée de ces outils jusqu'au niveau porte logique. Elle comprenait la modélisation et la validation de la description d'entrée, la synthèse d'architecture, la vérification du modèle RTL et la synthèse RTL avec Design Compiler. Les critères de comparaison ont été : la facilité d'écriture du modèle d'entrée, souplesse des outils à atteindre les contraintes d'architecture imposées, le temps de conception pour obtenir une architecture RTL, la qualité de cette architecture et la facilité de vérifier le travail effectué par l'outil.

II.4. Problèmes communs aux outils de synthèse d'architecture

Cette étude est basée sur les résultats d'une étude expérimentale menée à ST Microelectronics. Cette étude a permit de montrer que la synthèse d'architecture permet la génération d'une architecture RTL en un temps de conception plus court qu'une conception manuelle typique. Néanmoins, l'architecture produite est de loin de moins bonne qualité. Ce dernier fait implique que les outils de synthèse actuels ne peuvent encore être insérés dans le flot de conception industriel. Dans cette section, nous dressons les problèmes communs et fondamentaux liés aux outils de synthèse d'architecture de seconde génération. Ces problèmes devraient être résolus par les fournisseurs CAD pour pouvoir admettre la synthèse d'architecture dans un contexte industriel.

II.4.1. Difficulté d'intégration de la synthèse d'architecture dans le flux de synthèse RTL

Le principal problème de la synthèse d'architecture est son incapacité à utiliser pleinement les capacités de la synthèse RTL. Actuellement, la synthèse RTL a fait largement ses preuves en matière de conception matérielle.

La synthèse RTL vise à générer un modèle au niveau porte logique à partir d'une description au niveau RTL. Le modèle RTL est un réseau combinatoire pouvant être réalisé en un cycle d'horloge. Il est entrecoupé de registres permettant la mémorisation des données intermédiaires. La synthèse RTL consiste dans un premier temps à identifier ces deux structures. Puis, dans un deuxième temps, la synthèse RTL réalise des optimisations logiques guidées par des contraintes de synthèse et les caractéristiques de la librairie cible. L'étape finale est d'allouer des composants de la librairie et de les assigner des composants aux opérateurs logiques.

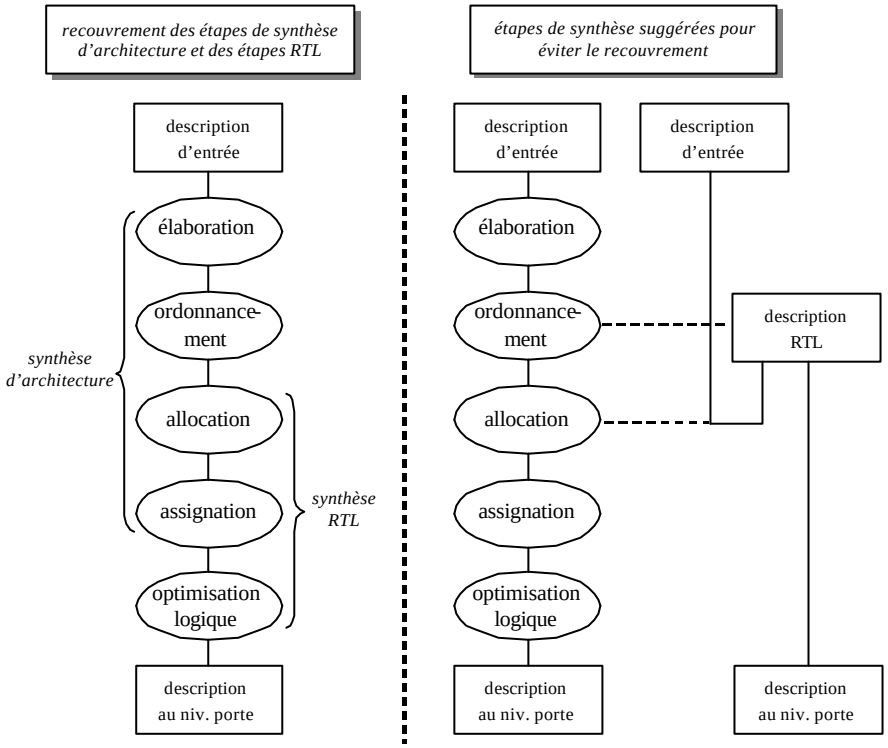


Figure 2. Recouvrement RTL des étapes de synthèse (gauche) et étapes de synthèse d'architecture suggérées pour éviter ce recouvrement (droite)

Les étapes d'allocation, d'assignation et d'optimisation logique font partie des tâches réalisées par la synthèse RTL. Or, la synthèse d'architecture réalise les tâches d'ordonnancement, d'allocation et d'assignation. Donc, seule la tâche d'ordonnancement est

spécifique à la synthèse d'architecture. La synthèse d'architecture recouvre la synthèse RTL pour les étapes d'allocation et d'assignation (figure 2-gauche). Ce recouvrement a pour principale conséquence de restreindre les capacités de la synthèse RTL et de ne pas pouvoir exploiter le maximum des capacités de cette synthèse.

Le type d'architecture RTL produite par la synthèse d'architecture restreint aussi les possibilités de la synthèse RTL. Par exemple, ces choix sont l'assignation d'un opérateur arithmétique avec un composant issue d'une librairie cible. Ce cas empêche toute optimisation logique par la synthèse RTL puisque l'opérateur est déjà assigné. Ce cas supprime aussi la portabilité du RTL généré. D'autres outils génèrent une netlist structurelle de composants sans assignation à une technologie particulière. Le fractionnement du chemin de données en des instances de composants réduit les possibilités de la synthèse RTL. En effet, les optimisations de la synthèse RTL sont locales et ne peuvent dépasser les frontières de la hiérarchie de composants. C'est pourquoi une sur-décomposition de la hiérarchie restreint sensiblement les capacités du synthétiseur RTL. A l'inverse, privilégier une mise à plat de toute la hiérarchie du système complique le travail de placement-routage et l'identification des zones critiques du système.

Un moyen de rendre compatible les outils de synthèse d'architecture avec le flux de conception RTL serait de limiter l'outil à la tâche d'ordonnancement. La tâche d'ordonnancement est la seule réelle tâche qui soit spécifique à la synthèse d'architecture. Malheureusement, la tâche d'ordonnancement est fortement dépendante de la technologie cible et de l'allocation. Donc, une solution serait d'utiliser les étapes de raffinements jusqu'à l'allocation, mais en produisant un modèle RTL compatible avec les règles bien-établies du codage du RTL synthétisable. La façon d'intégrer la synthèse d'architecture dans le flot de synthèse RTL est illustrée par la figure 2-droite.

II.4.2. Difficulté de comprendre et de prédire les résultats de la synthèse d'architecture

La synthèse d'architecture produit des architectures qui sont généralement difficiles à comprendre et à prédire : le lien entre la description d'entrée et le modèle RTL est difficile à établir. La conséquence de cela est l'obtention d'une architecture imprévisible et des pertes d'optimisation. Pour raffiner l'architecture résultante, les outils proposent des rapports d'architecture et autres outils d'analyse. Toutefois, ces outils d'analyse deviennent rapidement difficiles à utiliser lorsque le modèle se complexifie : pour une simple FFT 256 points radix-4 (algorithme de 40 lignes environ), les résultats des rapports deviennent déjà fastidieux à utiliser.

Ce problème provient de deux raisons. Tout d'abord, le modèle d'entrée subit beaucoup trop de raffinements pour établir une correspondance un à un entre l'algorithme et l'architecture. Ensuite, le résultat de la synthèse d'architecture est fortement dépendant du style d'écriture d'entrée. Cette dépendance au style d'écriture se voit aussi bien dans le choix de l'algorithme que dans la façon dont il est écrit (utilisation de variables intermédiaires, de parenthèses, de pointeurs, etc.). Or, la capacité des expressions et des structures des langages d'entrée dépasse largement les limites des algorithmes de synthèse.

La première cause peut être résolue en restreignant les capacités des outils d'architecture. En particulier, comme vu dans le §II.4.1, la correspondance entre le modèle d'entrée et le modèle RTL peut être plus appréhendable si la synthèse d'architecture est restreinte à l'étape d'ordonnancement.

La deuxième cause (la dépendance au style d'écriture) demande de restreindre la richesse des modèles d'entrée à un sous-ensemble maîtrisable et compréhensible. En synthèse RTL, le style strict du modèle RTL assure l'efficacité de l'optimisation logique. Il en devrait être aussi de même pour la synthèse d'architecture. Ce sous-ensemble doit être définit de telle façon que le concepteur doit pouvoir comprendre et anticiper le résultat de la synthèse d'architecture.

II.4.3. Code RTL illisible

La synthèse d'architecture produit un code RTL illisible et pouvant être difficilement reprit manuellement. A titre d'exemple, un simple filtre SRC décrit manuellement en 200 lignes peut être synthétisé en 58 pages de code. Ce manque de lisibilité implique une compréhension difficile de l'architecture générée.

Une meilleure lisibilité faciliterait alors le travail de raffinement et d'optimisation de l'architecture produite.

Une architecture plus claire permettrait de réaliser des optimisations manuelles. En synthèse RTL, il est parfois nécessaire de réaliser des optimisations manuelles pour les parties critiques. De même, avec la synthèse d'architecture, il devrait être possible de reprendre l'architecture produite. Le manque de lisibilité réduit alors la capacité d'optimisation manuelle.

II.4.4. Difficulté d'insertion d'un composant dans son environnement

La synthèse d'architecture produit des architectures difficiles à insérer dans leur contexte. Ce contexte peut être aussi bien d'autres composants numériques (conçus par d'autres moyens ou méthodes), des composants autres que des ASIC (composants analogiques, micro-processeurs, mémoires, etc.) ou l'environnement de validation.

La difficulté de l'insertion dépend du mode d'ordonnancement choisi (mode en un cycle, mode des macro-états ou mode des états comportementaux, entrées/sorties fixes ou entrées/sorties flottantes). L'ordonnancement modifie le comportement des entrées/sorties ainsi que le nombre de cycles de latence entre ces entrées/sorties vis-à-vis du modèle d'entrée. Cette différence de comportement doit alors être tenue en compte lors de l'insertion du composant. La difficulté d'insertion du composant est alors accrue par la perte de la maîtrise du comportement des entrées/sorties lors du processus de synthèse.

Pour palier cette lacune, les fournisseurs d'outils de synthèse d'architecture suggère d'éviter le mode des états comportementaux et le mode des entrées/sorties flottantes. Mais, cela limite les capacités des algorithmes d'ordonnancement et impose des restrictions sur le style d'écriture de la description d'entrée (cf. §II.2.2.3.).

Un autre moyen suggéré est l'utilisation systématique d'un protocole sur chaque IP généré par la synthèse d'architecture. Or, le problème de différence de comportement lors de l'insertion est un problème local : il intervient qu'au niveau du point de convergence de deux branches de traitements concurrentes, au niveau des boucles de contre-réaction et au niveau des IP dépendant du temps. Il n'est donc pas nécessaire d'envisager un moyen de correction sur tous les IP alors qu'une correction locale est suffisante pour obtenir un surcoût en surface moindre.

Des moyens locaux de correction de la différence de comportement lors de l'insertion sont de modifiant la FSM globale pilotant chaque IP ou d'insérer des registres de correction sur le chemin de données. La difficulté reste alors de savoir comment modifier la FSM ou de savoir combien de registres doivent être insérés. Ces modifications ne peuvent être entrepris qu'en maîtrisant le comportement des entrées/sorties du composant issu de la synthèse d'architecture.

II.4.5. Spécificité d'un outil à un domaine particulier

Comme montré dans [SUG 00][BER 97], les outils de synthèse d'architecture sont restreints à des domaines d'applications spécifiques. Les outils actuels industriels sont, en théorie, adaptés à des applications orientées flot de données, flot de contrôle ou mixtes. L'expérience montre que les outils sont plus efficaces soit pour des applications à dominante flot de données, soit à dominante flot de contrôle. Généralement, il donne de mauvais résultats pour les applications mixtes.

Les applications complexes sont généralement mixtes. Cette restriction à des domaines spécifiques demande aux concepteurs de partitionner l'application complète en sous-applications à dominante flot de données ou à dominante flot de contrôle. Ce partitionnement permet de tirer parti des forces de chaque outil.

Néanmoins, la difficulté d'insertion des architectures générées dans leur contexte tend à rendre difficile l'assemblage des sous-applications conçues séparément.

II.4.6. Capacité limitée d'utiliser un IP dans le processus de synthèse d'architecture

Les applications de traitement du signal pouvant être décomposées en sous-fonctions typiques et bien-connues. Ces sous-fonctions peuvent être issues de travaux précédents et être fortement optimisées. Donc, le concepteur peut souhaiter utiliser un modèle RTL pré-conçu d'une sous-fonction alors que le reste de l'application est synthétisée par synthèse d'architecture.

Or, les outils de synthèse d'architecture sont limités dans leur capacité d'insertion d'un modèle RTL déjà conçu dans le processus de synthèse. Par exemple, des outils sont restreints à l'insertion de sous-fonctions purement combinatoires. D'autres permettent l'insertion de composants plus complexes pouvant être séquentiels. Mais, ils sont restreints à des composants dont le nombre de cycles entre l'entrée et la sortie est constant.

II.4.7. Difficulté de vérifier la fonctionnalité de l'architecture produite

Dans les précédents outils de synthèse d'architecture, le modèle d'entrée était décrit en VHDL comportemental. L'avantage du VHDL comportemental est qu'il permet d'utiliser le même langage pour la vérification au niveau comportemental et au niveau RTL. Les outils actuels ont abandonnés le VHDL pour une description en C, C++ ou SystemC. Bien que ces langages permettent une modélisation et validation plus rapide, le même environnement de validation ne peut plus être utilisé pour la validation au niveau comportemental et au niveau

RTL. Il devient alors nécessaire de réécrire un nouvel environnement de validation au niveau RTL. Cette solution reste consommatrice en temps de vérification et source d'erreurs lors de la retranscription. Un autre moyen est de réaliser une cosimulation entre l'environnement de validation au niveau comportemental et le composant RTL synthétisé. Si cette solution permet d'utiliser le même environnement au deux niveaux d'abstraction, l'inconvénient est que le procédé de cosimulation doit tenir compte de la modification de comportement des entrées/sorties due à la synthèse d'architecture.

II.5. Conclusion

Dans ce chapitre, nous avons réalisé une étude comparative de trois outils de synthèse d'architecture. Les principaux résultats de cette étude sont les suivants : la synthèse d'architecture supporte une description d'entrée bien plus efficace à modéliser qu'une description RTL conventionnelle. Dans le cas de nos exemples, le gain est d'un facteur deux en temps de modélisation et de validation. Ce gain augmente sensiblement pour des applications plus complexes. La synthèse d'architecture permet une génération très rapide d'une architecture RTL (quelques minutes avec un outil de synthèse d'architecture comparée à quelques heures pour une conception manuelle dans le cadre de nos exemples). Néanmoins, la qualité de l'architecture produite est bien inférieure à la qualité d'une architecture produite manuellement par un concepteur. Dans le meilleur des cas, l'architecture générée par un outil avait une surface deux fois plus importante qu'une architecture conçue manuellement. Cette faible qualité rend la synthèse d'architecture impossible à insérer dans le flux de conception industriel actuel sinon pour un prototypage rapide.

Ce problème vient du fait que la synthèse d'architecture actuelle a du mal à tirer partie de la puissance de la synthèse RTL. En effet, la synthèse d'architecture recouvre par ses étapes, les étapes de synthèse RTL. Seule l'étape d'ordonnancement est spécifique à la synthèse

d'architecture. De plus, le nombre important des étapes de synthèse implique qu'il est difficile de prévoir le résultat produit par la synthèse d'architecture : une correspondance un à un entre la description d'entrée et l'architecture produite est difficile à établir. Ce constat implique que les boucles de conception sont coûteuses pour comprendre comment modifier la description d'entrée en vue d'obtenir une architecture optimale. La qualité de l'architecture est fortement dépendante du style d'écriture de l'entrée. Or, les langages d'entrée sont généralement des langages de programmation offrant une large variété d'écriture. Il devient alors nécessaire de restreindre le langage d'entrée à un sous-ensemble maîtrisable pour faciliter la compréhension de l'architecture résultante.

L'étape d'ordonnancement de la synthèse d'architecture modifie le comportement des entrées/sorties vis-à-vis de la description d'entrée. Ce constat complexifie à la fois la tâche de vérification de l'architecture RTL et l'intégration de cette architecture avec d'autres composants. Des moyens doivent alors être mis en œuvre pour adapter le comportement des entrées/sorties de l'architecture à son environnement. Mais, le problème majeur pour réaliser ces adaptations reste la perte de la maîtrise de l'architecture générée lors du processus de synthèse.

Chapitre III. Méthodologie pour la conception et la validation de macro-cellules ASIC dédiées au traitement du signal

Résumé : dans ce chapitre, nous présentons une méthodologie pour la conception et la validation de macro-cellules ASIC dédiées au traitement du signal. Cette méthodologie débute par une spécification de haut niveau et se base sur l'assemblage automatique d'IP préconçus et paramétrables. Cela permet une modélisation et une validation fonctionnelle rapide, une exploration conjointe algorithme/architecture, la préservation de la qualité de l'architecture comme si fait main, et une conception rapide d'applications dérivées. Nous illustrons l'efficacité de l'approche par l'expérimentation sur un exemple industriel : une chaîne de modulation numérique.

Note importante : ce chapitre est issu de la publication [TAM 03] (annexe B). Les sections décrivant cette méthodologie et son expérimentation sont résumées ici.

Sommaire :

Note importante : ce chapitre est issu de la publication [TAM 03] (annexe B). Les sections décrivant cette méthodologie et son expérimentation sont résumées ici.

III.1. Introduction

Résumé : la conception de macro-cellules ASIC dédiées au traitement numérique du signal devient trop coûteuse en temps et en effort de conception pour maîtriser la complexité grandissante de ces applications. Nous soulignons alors la nécessité d'avoir un flot et une méthodologie permettant la maîtrise de cette complexité tout en maintenant un temps de mise sur le marché raisonnable.

C.f. annexe B.1.1.

III.1.1. Etat de l'art

Résumé : beaucoup de travaux ont proposé des méthodologies et outils de conception pour améliorer la conception et la validation de ces macro-cellules : synthèse d'architecture, flot basé sur un langage dérivé du C, flot basé sur Simulink, etc. Les travaux précédents proposent des méthodologies de conception permettant de répondre à certains des problèmes de la conception des applications DSP mais, n'ont pas encore recouvert la globalité du problème. En effet, la plupart des approches citées ne peuvent satisfaire un compromis entre modélisation et validation algorithmiques efficaces d'une part, et exploration architecturale rapide et qualité de l'architecture générée d'autre part.

C.f. annexe B.1.2.

III.1.2. Contribution

Résumé : notre contribution est de proposer une méthodologie de conception et de validation pour macro-cellules ASIC de traitement du signal. Cette méthodologie part d'une description fonctionnelle et est basée sur :

- l'utilisation de blocs de base (DSP-IP par la suite ; par exemple, un filtre ou une FFT) de traitement du signal, préconçus, paramétrables, et décrits à la fois au niveau fonctionnel et RTL.
- l'assemblage automatique des IP RTL suivant l'assemblage défini par le modèle fonctionnel d'entrée.

C.f. annexe B.1.3.

III.1.3. Plan

Ce chapitre se compose de 5 sections. La section 2 introduit les concepts préliminaires de notre méthodologie. La section 3 décrit la méthodologie proposée. La section 4 illustre l'expérimentation de notre méthodologie sur un exemple industriel concret. La section 5 conclut ce chapitre.

III.2. Concepts préliminaires

Résumé : dans cette section, nous présentons les concepts utilisés par la suite : modèle fonctionnel, modèle RTL, DSP-IP fonctionnel, DSP-IP RTL et interface.

C.f. annexe B.2.

III.3. Méthodologie de conception et de validation

Résumé : dans cette section, nous présentons notre méthodologie de conception et de validation ainsi que les différentes étapes réalisées (modélisation fonctionnelle, génération du chemin de données du modèle RTL, génération et fusion de sa FSM, correction de retard et vérification). Chacune de ces étapes sont détaillées dans les sections qui suivent.

C.f. annexe B.3.

III.3.1. Modélisation fonctionnelle

Résumé : dans cette section, nous détaillons l'étape de modélisation et de validation fonctionnelle. Ce modèle est obtenu par assemblage de DSP-IP fonctionnels préconçus et paramétrables. Les paramètres des DSP-IP sont obtenus par simulation pour satisfaire un compromis entre qualité du signal et qualité de l'implémentation finale. Le choix de la topologie d'assemblage, des IP et de la valeur des paramètres est guidé par l'expérience du concepteur.

C.f. annexe B.3.1.

III.3.2. Génération du chemin de données et de la FSM

Résumé : à partir du modèle fonctionnel, le modèle RTL est obtenu en trois étapes : l'extraction de la netlist d'assemblage décrite dans le modèle fonctionnel, son raffinement en une netlist au niveau RTL et la génération de la machine d'états finis.

C.f. annexe B.3.3.

III.3.3. Méthode de correction de retard (i.e. approche de retiming)

Résumé : bien qu'individuellement chaque DSP-IP fonctionnel et RTL produisent les mêmes sorties numériques, après assemblage, le modèle RTL ne produit plus les mêmes valeurs numériques (i.e. perte du comportement) que le modèle fonctionnel. Une méthode dite *méthode de correction de retard* a été développée pour automatiquement corriger ce problème de différence de comportement.

Principale difficulté de notre méthodologie, le chapitre IV est dédié à l'analyse du problème de différence de comportement, à sa formalisation et à sa résolution.

C.f. chapitre IV et annexe B.3.3.

III.3.4. Plateforme de vérification

Résumé : dans cette section, nous proposons une plateforme ré-exploitant l'environnement de validation fonctionnelle pour la vérification du modèle RTL.

C.f. annexe B.3.4.

III.4. Exemple de conception industrielle

Résumé : dans cette section, nous présentons les résultats obtenus en appliquant notre méthodologie sur un exemple industriel : une chaîne de modulation numérique.

C.f. annexe B.4.

III.4.1. Chaîne de modulation numérique

Résumé : dans cette section, nous présentons la chaîne de modulation numérique, l'architecture système[1] adoptée ainsi que les contraintes fonctionnelles et d'implémentation à atteindre.

C.f. annexe B.4.1.

III.4.2. Conception et validation de la chaîne de modulation numérique

Résumé : cette section montre, étape par étape, comment la méthodologie est appliquée à un exemple industriel : une chaîne de modulation numérique. Elle présente l'application des étapes de modélisation fonctionnelle, de détermination des valeurs des paramètres, de génération du chemin de données, de génération de la FSM et de correction de retard.

C.f. annexe B.4.2.

III.4.3. Analyse des résultats

Résumé : cette section est dédiée à l'analyse des résultats obtenus.

C.f. annexe B.4.3.

[1] L'*architecture système* est la définition des fonctions élémentaires utilisées et de leurs interconnections

III.5. Conclusion

Dans ce chapitre, nous avons présenté une méthodologie pour la conception et la validation de macro-cellules ASIC dédiées au traitement du signal. Cette méthodologie débute par une spécification de haut niveau et se base sur l'assemblage automatique d'IP préconçus et paramétrables. Cela permet une modélisation et une validation fonctionnelle rapide, une exploration conjointe algorithme/architecture, la préservation de la qualité de l'architecture comme si fait main, et une conception rapide d'applications dérivées. Nous avons illustré l'efficacité de l'approche par l'expérimentation sur un exemple industriel : une chaîne de modulation numérique.

Nous avons souligné que la principale difficulté de cette méthodologie est que le passage d'un assemblage d'IP fonctionnels vers un assemblage d'IP RTL peut entraîner une différence de comportement : le modèle RTL ne produit plus les mêmes valeurs numériques que le modèle fonctionnel. La formalisation de ce problème ainsi que sa résolution est le sujet du chapitre qui suit.

Chapitre IV. Correction de retard pour le passage d'un assemblage de DSP-IP fonctionnels vers un assemblage de DSP-IP RTL : formalisation et approche

Résumé : le comportement d'une application de traitement du signal peut être perdu lors du passage d'un assemblage d'IP fonctionnels vers un assemblage d'IP RTL. Ce chapitre traite d'une approche (appelée *méthode de correction de retard*) permettant de corriger automatiquement ce problème. Dans le cas des assemblages sans boucle, les corrections sont faites par insertion de registres dans le chemin de données du modèle RTL. La localisation et la détermination du nombre de registres sont réalisées pour optimiser la latence et la surface du modèle RTL final. La validité de l'approche proposée ainsi que l'optimalité de la solution sont démontrées mathématiquement. Ce moyen de correction sera expérimentalement comparé à une correction par modification de la FSM du modèle RTL. Dans le cas des assemblages avec boucle, nous montrons que l'insertion de registres ne peut pas résoudre le problème de différence de comportement et discutons les alternatives d'implémentation. Nous montrons comment adapter la formalisation utilisée dans le cas des assemblages contenant des boucles. Un algorithme de correction est alors proposé.

Sommaire :

IV.1. Introduction

IV.1.1. Problématique, cause et moyens de correction

Une idée largement répandue et appliquée pour concevoir une application DSP est d'adopter une approche modulaire : la complexité globale de l'application est alors décomposée en sous-fonctions élémentaires, bien connues et de complexité plus abordables [MAR 02][SCH 01] (e.g. filtre, FFT). Par la suite, ces sous-fonctions seront appelées *DSP-IP fonctionnels* ou *DSP-IP RTL* suivant leur niveau d'abstraction. Dans une approche modulaire, un premier modèle (appelé *modèle fonctionnel*) est réalisé par assemblage de DSP-IP fonctionnels. Le but de ce modèle est de valider la fonctionnalité de l'application. Le modèle RTL est obtenu en assemblant de la même façon des DSP-IP RTL. Une phase de vérification permet de vérifier que le processus de conception a conservé le comportement du modèle RTL vis-à-vis du modèle fonctionnel.

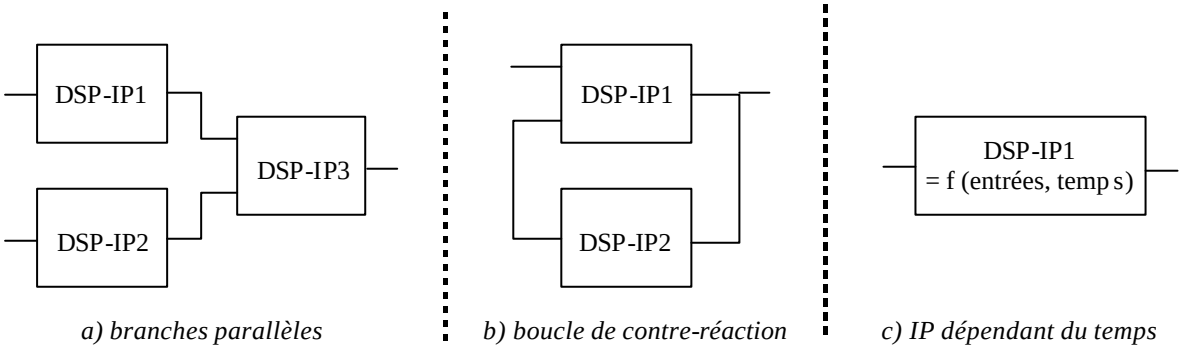


Figure 3. Cas pouvant produire une différence de comportement

Bien que chaque DSP-IP fonctionnel et RTL produisent les mêmes sorties numériques, le problème est ici que le passage de l'assemblage au niveau fonctionnel vers l'assemblage au niveau RTL peut entraîner une *différence de comportement* de l'application complète [TAM 02]. Nous entendons par *différence de comportement*, le fait que le modèle RTL ne produise plus exactement les mêmes sorties numériques que le modèle fonctionnel : la fonctionnalité

globale de l'application a alors été perdue pendant l'assemblage. Ce problème intervient (figure 3) lorsque l'application contient des branches parallèles de traitement convergeant sur un IP, des boucles de contre-réaction ou des IP dépendant du temps (i.e. lorsque le signal d'entrée est retardé, l'IP ne produit pas une version retardée du signal de sortie). Hormis les applications composées d'IP en série, la grande majorité des applications de traitement du signal (e.g. modulation/démodulation, codeur de voix, adaptateur de fréquences) contiennent ces trois éléments.

Cette différence de comportement est due à des retards présents dans le modèle RTL et absents du modèle fonctionnel. En effet, la modélisation fonctionnelle demande de modéliser uniquement les retards liés à la fonctionnalité propre de l'application (e.g. la ligne de retards d'un filtre). Or, la conception du modèle RTL entraîne l'ajout de retards supplémentaires permettant de satisfaire les contraintes d'implémentation (e.g. registres de pipeline, registre de sortie).

Une correction doit être apportée sur le modèle RTL en vue d'obtenir le même comportement que le modèle fonctionnel. Il existe trois techniques possibles de correction :

- *Protocole de synchronisation* : la première technique consiste à ajouter un protocole *poignée de main* (*handshake*) sur chaque DSP-IP RTL. Ce protocole spécifie au DSP-IP à quel instant les données d'entrée et de sortie sont valides. L'avantage de cette solution est que les problèmes de retard sont résolues dés l'assemblage des DSP-IP. L'inconvénient est le surcoût en surface sur chaque IP car un protocole est ajouté sur tous les IP alors que le problème de différence de comportement est seulement localisé au niveau des cas de la figure 1.
- *Insertion de registres* : la deuxième technique consiste à insérer des registres entre les DSP-IP du modèle RTL. Cette solution permet de compenser les retards additifs et

présente l'avantage d'être non intrusive (i.e. pas de nécessité de modifier les IP). Mais, effectuer manuellement les corrections est un processus ardu et complexe à réaliser sur des applications complexes (source d'erreurs, accroît le temps de conception). C'est-à-dire, localiser les endroits où la différence de comportement intervient, déterminer comment apporter les corrections, et implémenter ces corrections. En plus, le processus de correction inclut la recherche d'une solution optimale en latence et en surface. En effet, la résolution du problème de différence de comportement n'admet pas qu'une solution, mais plusieurs solutions.

- *Modification de la FSM (Finite State Machine) globale* : la troisième technique consiste à modifier la FSM initiale en générant des signaux de contrôle supplémentaires. Ces signaux sont des versions décalées dans le temps des signaux initiaux. Ils permettent de retarder ou d'avancer l'activation des IPs. Elle nécessite de multiplier le nombre de signaux de contrôle et d'augmenter le nombre d'états de la FSM. Cette solution reste fastidieuse à mettre en oeuvre. Elle nécessite d'une part les mêmes étapes que la solution précédente, et d'autre part la modification de la FSM initiale. Le coût en modification augmentant avec la complexité de la FSM.

IV.1.2. Objectif et contribution

L'objectif est ici de résoudre le problème de différence de comportement lors du passage d'un assemblage de DSP-IP fonctionnels vers un assemblage de DSP-IP RTL.

Les contributions sont :

- la formalisation du problème de différence de comportement dans le cas des structures d'assemblage sans boucle et sans IP dépendant du temps (cas de la figure 3a). Nous proposons *une méthode de correction automatique de retard par insertion de registres* pour résoudre ce problème. Nous élaborons deux nouveaux algorithmes permettant de

trouver une solution optimale en latence et en surface. Notons que ces algorithmes peuvent aussi s'appliquer à la troisième technique. Il ne manquera que l'automatisation de la dernière étape (i.e. modification de la FSM).

- la reformulation du problème de différence de comportement et la proposition d'un algorithme dans le cas des applications avec boucle (cas de la figure 3b). Nous montrons que l'insertion de registres ne permet pas de corriger le problème de différence de comportement dans le cas des boucles. Nous discutons les alternatives d'implémentation.

IV.1.3. Etat de l'art et position de notre contribution

La méthode de correction de retard insert et manipule des registres. La manipulation de registres est appelée *retiming* [MAH 99] dans la littérature. Le retiming consiste à déplacer des registres à travers les éléments logiques d'un circuit décrit au niveau porte logique. Cette opération permet d'optimiser un circuit sans modifier son comportement initial. Ces registres sont déplacés en vue d'optimiser la fréquence d'horloge, de minimiser la latence ou de réduire le nombre de registres dans le circuit. Notre contribution se distingue du retiming par deux points :

- elle permet de corriger une erreur de comportement vis-à-vis d'un modèle de référence alors que le retiming est inefficace pour cela.
- restreinte au cas des applications sans boucle, l'optimisation en nombre de registres est plus efficace avec les algorithmes proposés qu'avec ceux proposés par le retiming.

IV.1.4. Plan

Ce chapitre se divise en 5 sections. La section 2 introduit la méthode de correction de retard dans le cas des assemblages sans boucle, et détaille les différentes étapes. La faisabilité de la méthode sera alors démontrée. Pour faciliter la compréhension, les différentes

démonstrations sont laissées en annexe. La section 3 reformule le problème de différence de comportement dans le cas des applications avec boucle. Nous montrons que l'insertion de registres est inefficace et explorons les alternatives. Dans la section 4, la méthode de correction sera appliquée sur un exemple. Le coût de la correction par insertion de registres sera alors comparé au coût d'une correction par modification de la FSM. La section 5 conclut ce chapitre.

IV.2. Flot de correction de retard dans le cas des applications sans boucle

La figure 4 illustre le flot adopté de correction de retard. Les entrées de ce flot sont : une description fonctionnelle, une description RTL (incluant chemin de données et FSM). La sortie de ce flot est une description RTL corrigée produisant les mêmes sorties numériques que la description fonctionnelle.

La localisation et le calcul du nombre de registres à insérer nécessitent l'utilisation d'une forme intermédiaire (appelée *graphe différentiel d'évolution*) mettant en évidence les retards présents dans le modèle RTL et absents du modèle fonctionnel. Pour cela, chaque description (i.e. fonctionnelle et RTL) est représentée par un graphe (dit *graphe fonctionnel d'évolution* et *graphe RTL d'évolution* suivant le niveau d'abstraction) représentant ses retards propres. Le graphe d'évolution différentiel est construit en faisant un à un, la différence des poids des arcs du graphe RTL par ceux du graphe fonctionnel. Cette différence permet de ne laisser apparaître que les retards additifs dus aux contraintes d'implémentation en supprimant tous les retards liés à la fonctionnalité.

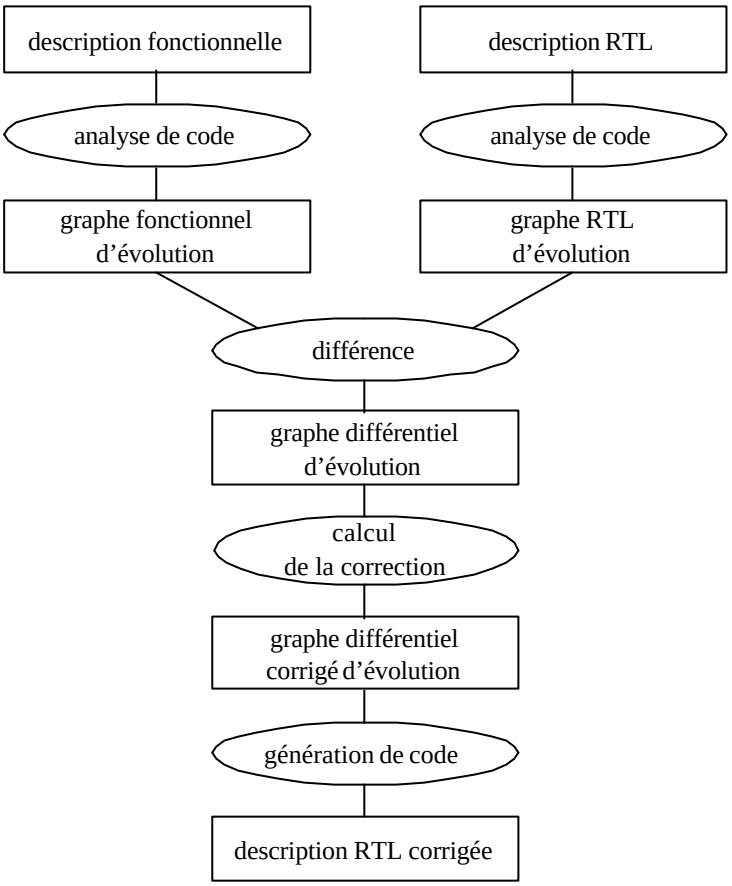


Figure 4. Flot de correction de retard

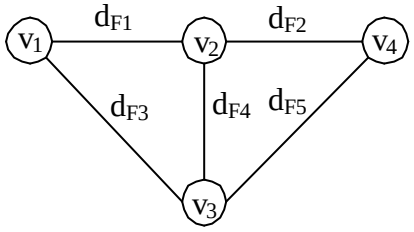
Un algorithme de correction de retard détermine à partir du graphe différentiel, les corrections nécessaires permettant de compenser ces retards additifs. Finalement, une étape de génération de code produit la description RTL finale en insérant le nombre adéquat de registres à la place adéquate.

Les différentes étapes de ce flot sont détaillées dans les sections qui suivent.

IV.2.1. Graphe d'évolution fonctionnel et RTL

La description fonctionnelle de l'application traitée est représentée par un *graphe fonctionnel d'évolution* modélisant le flot de données ainsi que les retards induits par chaque IP fonctionnel (figure 5). Ce graphe se compose de sommets et d'arcs orientés. Un sommet du graphe représente un DSP-IP fonctionnel, un port d'entrée ou de sortie de l'application. Un arc

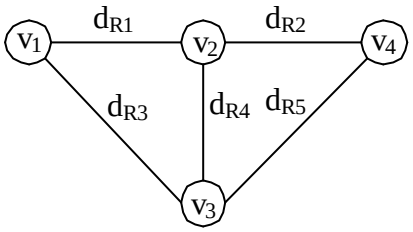
orienté modélise le flot de donnée d'un sommet vers un autre sommet. Les arcs sont pondérés : le poids d'un arc correspond au nombre d'itérations pour qu'une donnée présente à l'entrée d'un IP soit effectivement présente à la sortie de cet IP (notion de *latence*). Au niveau signal au sens traitement numérique du signal, seul l'ordre des échantillons du signal est important (*temps logique*). Donc, la $N^{ième}$ donnée produite à la $N^{ième}$ itération de l'IP correspond au $N^{ième}$ échantillon du signal. Pour uniformiser les unités au niveau fonctionnel et RTL, nous ne parlerons plus de nombre d'itérations, mais de nombre d'échantillons.



sommet : DSP-IP fonctionnel, port d'entrée ou de sortie de l'application complète
arc : flot de données d'un sommet à un autre
poids d'un arc : nb d'échantillons du signal pour qu'une donnée transite de l'entrée d'un sommet à l'entrée d'un autre

Figure 5. Graphe fonctionnel d'évolution

Comme la description fonctionnelle, la description RTL de l'application est représentée par un *graphe RTL d'évolution* modélisant le flot de données ainsi que les retards induits par chaque IP RTL (figure 6). Ce graphe se compose de sommets et d'arcs orientés. Un sommet du graphe représente un DSP-IP RTL, un port d'entrée ou de sortie de l'application. Un arc orienté modélise le flot de données d'un sommet vers un autre sommet. Les arcs sont pondérés : le poids d'un arc correspond au nombre de cycles d'horloge pour qu'une donnée présente à l'entrée d'un IP soit effectivement présente à la sortie de cet IP (notion de *latence*). On se ramène à la notion d'*échantillon* en échantillonnant le signal (au sens RTL) à la fréquence d'horloge. Ainsi, cet échantillonnage supprime le temps physique pour ne conserver que les valeurs du signal dans leur ordre d'apparition (*temps logique*). Puisque la fréquence d'horloge correspond au quantum de temps, la latence pondérant chaque arc est alors un entier.



sommet : DSP-IP RTL, port d'entrée ou de sortie de l'application complète
arc : flot de données d'un sommet à un autre
poids d'un arc : nb d'échantillons du signal pour qu'une donnée transite de l'entrée d'un sommet à l'entrée d'un autre

Figure 6. Graphe RTL d'évolution

Par construction, les graphes d'évolution fonctionnel et RTL ont la propriété d'être des graphes isomorphes[2] car la topologie d'assemblage réalisée au niveau fonctionnel est reproduite au niveau RTL.

Puisque seul le cas (a) de la figure 3, les DSP-IP ont alors la propriété d'être des *fonctions indépendantes du temps*[3]. Cette propriété sera largement exploitée dans les démonstrations qui suivent.

Nous formulons l'hypothèse que les DSP-IP ont un débit de données constant (*flot de données statique*). Dans le cas contraire (i.e. IP dont le débit dépend des données), une FIFO est généralement placée à la sortie de l'IP pour réguler le flot de données. Ainsi, la composition de l'IP et de sa FIFO peut être vue comme un seul IP de débit constant. On est alors ramené au cas des IP en flot de données statique.

IV.2.2. Obtention du graphe différentiel et condition suffisante de conservation du comportement

Le graphe fonctionnel d'évolution met en évidence les retards propres à la fonctionnalité alors que le graphe RTL d'évolution inclut aussi les retards liés aux contraintes d'implémentation. Seuls les retards liés à l'implémentation doivent être compensés à l'aide de registres, les autres étant inhérents à la fonctionnalité. Ainsi, un nouveau graphe est créé pour

[2] Deux graphes sont dits *isomorphes* lorsqu'ils ont la même topologie d'interconnections
[3] *f* est dite *invariant dans le temps* si le signal d'entrée *x[n-i]* produit la réponse *y[n-i]* par *f*, où *x[n]* est un signal arbitraire et *y[n]* est sa réponse correspondante.

ne laisser apparaître que les retards liés à l'implémentation. Ce graphe (appelé *graphe différentiel d'évolution*) est simplement obtenu en effectuant la soustraction un à un, des poids des arcs du graphe RTL par ceux du graphe fonctionnel (figure 7). Cette soustraction est possible puisque les deux graphes sont isomorphes.

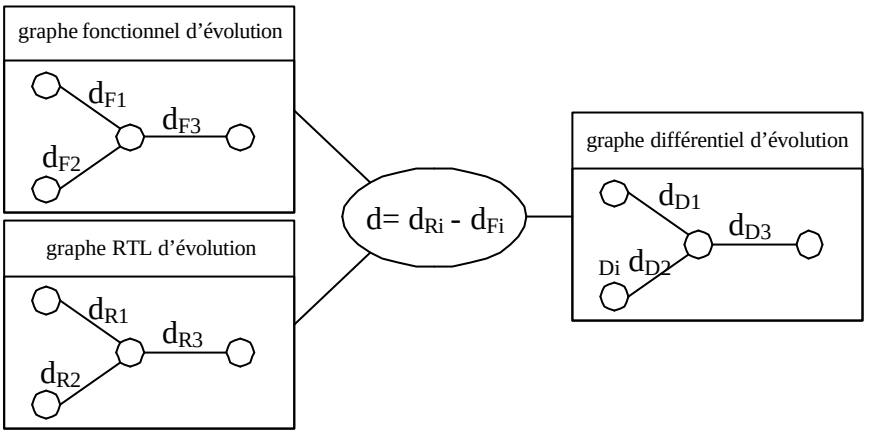


Figure 7. Graphe différentiel d'évolution

Le théorème 1 exprime une condition suffisante sur le graphe différentiel d'évolution pour que le modèle RTL et le modèle fonctionnel aient le même comportement. Ce théorème introduit la notion de *graphe équilibré* (définition 1).

Définition 1. *Graphe équilibré*

Enoncé : un graphe différentiel d'évolution est dit *équilibré* si et seulement si, pour tout sommet, la somme des retards est indépendante du chemin depuis une entrée jusqu'au sommet considéré. I.e. Pour tout sommet, la somme des retards est constante quelque soit le chemin emprunté depuis une entrée jusqu'au sommet considéré.

Théorème 1. *Condition suffisante de conservation du comportement*

Enoncé : si un graphe différentiel d'évolution est équilibré alors le modèle fonctionnel et le modèle RTL ont le même comportement.

Démonstration : c.f. annexe A.2.

Dans le cas où la condition du théorème n'est pas satisfaite, il devient alors nécessaire d'apporter des corrections au graphe différentiel d'évolution. Ces corrections se traduisent par l'ajout de constantes positives aux poids existants en vue d'obtenir un graphe équilibré. Concrètement, ajouter N à l'un des arcs revient à ajouter N registres au niveau du modèle RTL (figure 8).

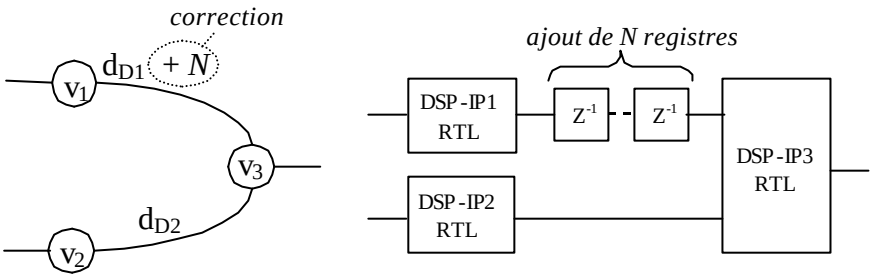


Figure 8. Correspondance le graphe différentiel d'évolution (gauche) et le modèle RTL (droite) pour les corrections apportées

Reformulation du problème

Le problème de différence de comportement devient alors : quelles valeurs entières positives doit-on ajouter sur les arcs du graphe différentiel d'évolution pour que le graphe devienne un graphe équilibré?

IV.2.3. Algorithmes de correction de retard

Deux algorithmes ont été développés pour déterminer les corrections à apporter au graphe différentiel en vue d'obtenir un graphe équilibré. Le premier détermine une solution minimisant la latence alors que le deuxième détermine une solution optimale en nombre de registres insérés (par voie de conséquence, optimisant la surface). Suivant les contraintes d'implémentation, le concepteur peut choisir l'algorithme qui lui convient.

IV.2.3.1. Algorithme de correction de retard pour l'optimisation en latence

Le premier algorithme de correction de retard a pour objectif de déterminer une solution optimale en latence. La latence d'un système est définie comme le temps qui s'écoule entre le moment où commence un stimulus et le moment où apparaît la réponse [REN 03]. Les applications de traitement numérique du signal nécessitent généralement l'utilisation de valeurs passées pour le calcul de la sortie courante. Ainsi, la latence est définie par un intervalle [latence minimale, latence maximale]. Au niveau du graphe RTL d'évolution, la latence minimale (resp. latence maximale) d'une sortie correspond au temps mis pour parcourir le plus court (resp. le plus long) chemin depuis les entrées jusqu'à la sortie considérée. Parmi l'ensemble des solutions possibles, l'algorithme détermine une solution minimisant aussi bien la latence minimale que maximale du modèle RTL. La convergence de l'algorithme vers une solution est démontrée en annexe A.3. La démonstration de l'annexe A.4. montre l'optimalité de la solution vis-à-vis du critère de la latence.

L'algorithme proposé (tableau 2 et algorithme 1) est analogue à un algorithme de recherche du plus long chemin de Bellman [SAK 84a]. Comme dans l'algorithme de Bellman, cet algorithme est un algorithme de marquage : il apporte des corrections de sommet en sommet sans retour en arrière sur les sommets déjà traités. La fonction de marquage $L(v)$ associe au sommet v, le retard cumulé depuis une entrée jusqu'au sommet v. Initialement, les sommets sans prédécesseur sont marqués par la valeur 0 (ligne 1). A chaque itération (ligne 2 à 9),

l'algorithme recherche un sommet non marqué (ligne 3) dont tous les sommets prédécesseurs ont déjà été marqués. Ce sommet existe à chaque itération car le graphe différentiel d'évolution est sans boucle. Parmi tous les arcs incidents à ce sommet, l'algorithme détermine l'arc qui correspond au plus long chemin (ligne 4 à 6). Une correction est apportée en vue d'égaliser le retard cumulé sur les autres arcs vis-à-vis du plus long chemin (ligne 7). L'algorithme s'arrête lorsque tous les sommets sont marqués (ligne 2 et 8).

$G = (V,E)$	Graphe différentiel à équilibrer de sommets V et d'arcs E
$G_1 = (V_1,E_1)$	Sous graphe de G
$L(v)$	Fonction de marquage
$d(e)$	Retard induit par l'arc e
$c(e)$	Correction apportée à l'arc e
$\tilde{A}_G^-(v)$	Ensemble des sommets prédécesseurs de v sur le graphe G
$I(e)$	Extrémité initiale de l'arc e
$max(g(x)/x \hat{I} A)$	Maximum des valeurs $g(x)$ pour x appartenant à l'ensemble A

Tableau 2. Notations de l'algorithme 1

1 : $V_1 = \{v \in V / \Gamma_G^- = \varnothing\}$ et $\forall v \in V_1, L(v) = 0$
2 : **Tant que** V_1 V **faire :**
3 : Soit un sommet v tel que $v \notin V_1$ et $\Gamma_G^-(v) \subset V_1$
4 : Soit E^- l'ensemble des arcs arrivant en v
5 : $\forall e \in E^-, p(e) = L(I(e)) + d(e)$
6 : $L(v) = \max(p(e) / e \in E^-)$
7 : $\forall e \in E^-, c(e) = L(v) - p(e)$
8 : Incrémenter V_1 en faisant $V_1 = V_1 \cup \{v\}$
9 : **Fin tant que**

Algorithme 1. Algorithme de correction de retard pour l'optimisation en latence

Exemple 1. *Application de l'algorithme 1 sur un exemple de graphe différentiel*

Le graphe de la figure 9 donne un exemple de graphe différentiel avant et après application de l'algorithme de correction de retard. Les annotations *L(v)* sont placées au dessus des sommets et les corrections apportées sont notées par '+N'. Le tableau 3 montre les corrections obtenues à chaque itération de l'algorithme. Dans le tableau, un arc est noté $[v_i, v_f]$ où v_i et v_f désignent les sommets extrémités de l'arc.

Le graphe final obtenu est bien un graphe équilibré. En effet, pour chaque sommet, les longueurs des chemins allant au sommet considéré sont maintenant constantes. Une simple énumération des chemins permet de le vérifier.

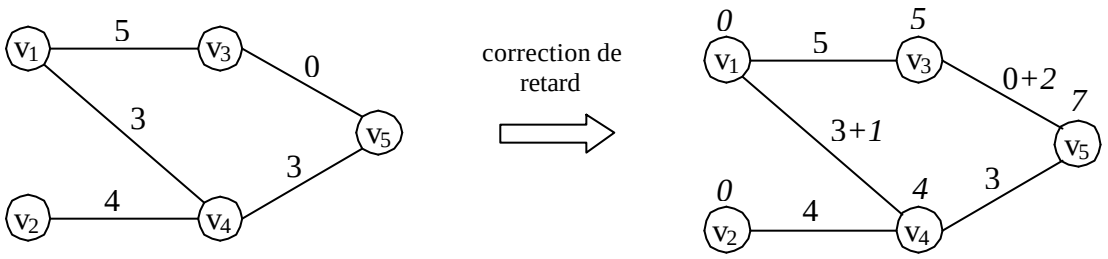


Figure 9. Exemple de graphe différentiel avant (gauche) et après (droite) application de l'algorithme de correction de retard

Etape	sommet *v*	ð(e) = L(I(e)) + d(e)	L(v) = max(ð (e_i))	c(e) = L(v) – d(e)
0	v_1, v_2		$L(v_1) = L(v_2) = 0$	pas de correction
1	v_3	ð ($[v_1, v_3]$) = 0 + 5 = 5	$L(v_3) = 5$	pas de correction
2	v_4	ð ($[v_1, v_4]$) = 0 + 3 = 3 ð ($[v_2, v_4]$) = 0 + 4 = 4	$L(v_4) = 4$	$c([v_1, v_4]) = 1$
3	v_5	ð ($[v_3, v_5]$) = 5 + 0 = 5 ð ($[v_4, v_5]$) = 4 + 3 = 7	$L(v_4) = 7$	$c([v_3, v_5]) = 2$

Tableau 3. Descriptif des étapes réalisées par l'algorithme de correction de retard

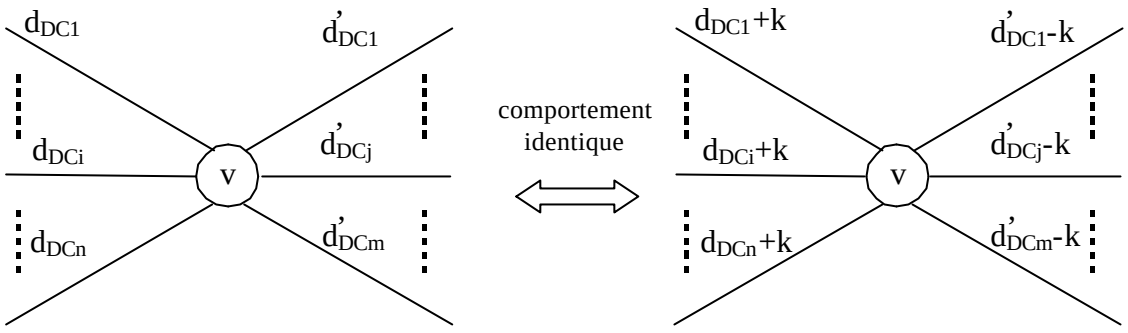
IV.2.3.2. Algorithme de correction de retard pour l'optimisation en nombre de registres insérés

Le second algorithme (algorithme 2 et tableau 4) de correction de retard a pour objectif de déterminer une solution optimale en nombre de registres. Il recherche une solution optimale en nombre de registres insérés à partir de la solution fournie par l'algorithme (algorithme 1) de correction optimale en latence. A la différence de l'algorithme 1, la recherche d'une solution optimale est ici faite en partant des sorties pour revenir vers les entrées. A l'instar de l'algorithme du simplexe [SAK 84b], l'algorithme 2 détermine de proche en proche, une nouvelle solution améliorant la solution courante en effectuant des *factorisations de retard* successives (théorème 2). Les factorisations de retard réalisées ici sont analogues aux déplacements de registres effectués par le retiming [MAH 99] à la différence que la factorisation de retard s'applique sur une granularité plus forte : le retiming travaille au niveau porte logique alors que la factorisation de retard travaille au niveau IP. Comme le prouve la démonstration du théorème 2 (c.f. annexe A.5), la factorisation de retard conserve également le comportement du modèle RTL lorsqu'elle est appliquée sur le graphe différentiel au niveau de l'IP.

Théorème 2. *Factorisation de retard*

Enoncé : soit un graphe différentiel corrigé d'évolution. Soit *v* un sommet du graphe et $k \in Z$ une constante. Alors, le comportement du sommet *v* est inchangé si on supprime *k* corrections aux poids des arcs d'extrémités initiales *v* et si on ajoute ces *k* corrections aux poids des arcs d'extrémités terminales *v* (figure 10).

Démonstration : c.f. annexe A.5.



d_{DC} : poids des arcs du graphe différentiel corrigé

Figure 10. Factorisation de retard

L'algorithme 2 est initialisé (ligne 1) par la correction obtenue par l'algorithme de correction optimisé en latence. Il se compose de deux boucles imbriquées. La boucle externe (ligne 3 à 22) parcourt tous les sommets du graphe en partant du sommet terminal et en revenant vers le sommet initial (ligne 2, 4 et 21). Pour chaque sommet intermédiaire, la boucle interne (ligne 7 à 17) recherche une solution améliorant la solution courante. Pour cela, l'algorithme réalise une série de factorisations de retard (ligne 12 et 13) depuis ce sommet jusqu'au sommet initial. Comme une solution ne peut être négative, la quantité factorisée sera le minimum des corrections sur les arcs sortants de ce sommet (ligne 9 et 10).

$G = (V, E)$	Graphe à équilibrer de sommets V et d'arcs E
G_1, G_2, G_3	Sous graphes de G
$C = \{c(e) / e \in E\}$	Ensemble des corrections
$\tilde{A}_G^+ (v)$	Ensemble des sommets successeurs à v sur le graphe G
$T(e)$	Extrémité terminale de l'arc e
$min(g(x)/x \hat{I} A)$	Minimum des valeurs $g(x)$ pour x appartenant à l'ensemble A
$f(c) = \sum_{e \in E} c(e)$	Fonction de coût : nombre de registres insérés

Tableau 4. Notations de l'algorithme 2

1 : C = ensemble des corrections obtenues par l'algorithme 1
2 : $V_1 = \{v \in V / \Gamma_G^+ = \varnothing\}$
3 : **Tant que** $V_1 \quad V$ **faire :**
4 : Soit un sommet v_1 tel que $v_1 \notin V_1$ et $\Gamma_G^+(v_1) \subset V_1$
5 : $V_2 = \{v \in V / \exists$ un chemin de v à $v_1\}$
6 : $v_3 = v_1$, $V_3 = \{v_3\}$, $C_{new} = C$
7 : **Tant que** $f(C_{new}) \geq f(C)$ et $V_3 \neq V_2$ et $k \neq 0$ **faire :**
8 : Soit E^-, l'ensemble des arcs arrivant en v_3
9 : Soit E^+, l'ensemble des arcs sortants de v_3
10 : $k = \min(c(e) / e \in E^+)$
11 : **Si** ($k \neq 0$) **faire :**
12 : $\forall e \in E^+, c_{new}(e) = c_{new}(e) - k$
13 : $\forall e \in E^-, c_{new}(e) = c_{new}(e) + k$
14 : **Fin si**
15 : Incrémenter V_3 en faisant $V_3 = V_3 \cup \{v_3\}$
16 : Soit un sommet $v_3 \in V_2$ tel que $v_3 \notin V_3$ et $\Gamma_G^+(v_3) \subset V_3$
17 : **Fin tant que**
18 : **Si** ($f(C_{new}) \leq f(C)$) **faire :**
19 : $C = C_{new}$
20 : **Fin si**
21 : Incrémenter V_1 en faisant $V_1 = V_1 \cup \{v_1\}$
22 : **Fin tant que**

Algorithme 2. Algorithme de correction de retard pour l'optimisation en nombre de registres

La boucle interne s'arrête (ligne 7) dans l'un des trois cas : une solution meilleure a été trouvée lors de la propagation de la factorisation ; la propagation a été faite jusqu'au bout sans trouver de solution meilleure ; la factorisation est effectuée sur une quantité nulle. Si la propagation trouve une solution améliorant la solution courante (ligne 18 et 20), cette nouvelle solution servira de solution de base (ligne 19) pour le nouveau sommet pris par la boucle externe.

L'algorithme 2 a une complexité algorithmique de l'ordre de $O(V^2)$. Il est donc plus efficace que les algorithmes de retiming visant l'optimisation en nombre de registres. Ces derniers sont de complexité $O(V^3)$ [MAH 99]. Cela vient du fait qu'il a été conçu dans le cadre des graphes sans boucle alors que le retiming se généralise au graphe avec boucles.

Exemple 2. *Application de l'algorithme 2 sur un exemple de graphe différentiel*

L'algorithme 2 est appliqué sur le graphe différentiel d'évolution de la figure 11 (haut de la figure). Cette figure montre comment la solution optimale en latence est progressivement transformée en une solution optimale en nombre de registres. Par convention, un sommet grisé est le sommet considéré par la boucle externe. Une flèche au dessus d'un sommet illustre la propagation de la factorisation (boucle interne). La valeur sur cette flèche est la quantité factorisée. Les poids des arcs sont les valeurs de la correction. En particulier les poids en italique sont les valeurs de la correction modifiées par la factorisation.

L'algorithme est d'abord initialisé par la solution obtenue par l'algorithme 1. Cette solution a un coût de 12 registres. A la première itération de la boucle externe, une factorisation est faite à partir du sommet v_4. Cette factorisation permet de gagner 2 registres. Donc, l'algorithme passe à la deuxième itération en considérant le sommet v_3. La factorisation à partir de v_3 n'améliore pas la solution courante. Donc, la factorisation est propagée au sommet v_2. Cette dernière est fructueuse car elle permet de gagner 2 registres. La boucle externe traite alors le sommet v_2. La factorisation sur v_2 propage une valeur nulle. La boucle externe traite alors le sommet v_1. Puisque v_1 est l'entrée du graphe, l'algorithme se termine. La dernière solution obtenue est la solution optimale en nombre de registres.

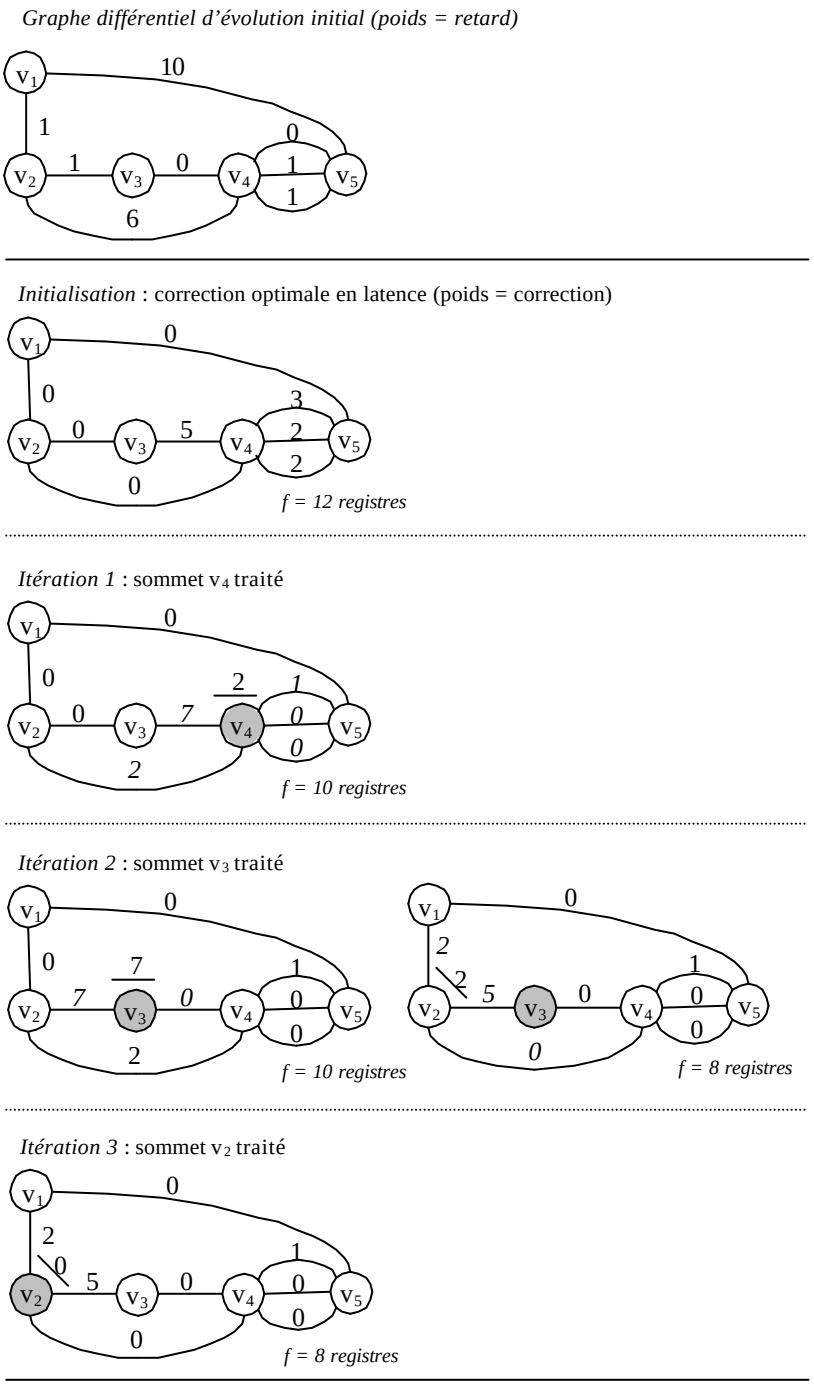


Figure 11. Application de l'algorithme 2 sur un exemple de graphe différentiel

IV.2.4. Génération de la description RTL corrigée

Les corrections apportées au graphe différentiel et obtenues par les algorithmes de correction de retard sont reportées sur le modèle RTL. Ces corrections se traduisent

concrètement par l'insertion de blocs retard Z^{-N} paramétrables entre les DSP-IP RTL. Un bloc retard Z^{-N} correspond à N registres en série.

Après correction, le modèle RTL a le même comportement que le modèle fonctionnel : les signaux produits par les deux modèles sont alors identiques numériquement.

IV.3. Correction de retard dans le cas des applications avec boucle

IV.3.1. Formalisation

Dans le cas des applications avec boucle, nous formulons les deux hypothèses suivantes. La première hypothèse restreint l'étude à la sous-classe des applications dont les boucles sont atteignables depuis l'une des entrées de l'application. La deuxième hypothèse admet sans démonstration que le théorème de conservation du comportement reste valable pour les applications avec boucle.

Hypothèse 1. *Tous les sommets du graphe d'évolution différentiel sont atteignables*

On suppose que tous les sommets du graphe d'évolution différentiel sont atteignables : i.e. pour tout sommet *v*, il existe un chemin partant d'une entrée et arrivant en *v*.

Hypothèse 2. *Condition suffisante de conservation de comportement*

La condition suffisante (i.e. si le graphe différentiel est un graphe équilibré alors le modèle RTL et fonctionnel ont un même comportement) démontrée pour le cas des applications sans boucle est supposée encore valable dans le cas des assemblages ayant des boucles.

Le cas des applications avec boucle induit une propriété (théorème 3) liant le graphe équilibré aux cycles[4] présents dans le graphe différentiel. Cette propriété sera largement exploitée par la suite.

Théorème 3. *Relation entre graphe équilibré et cycle*

Enoncé : un graphe G est équilibré $\Leftrightarrow \forall\, \Gamma$ cycle de G, $\overline{L}(\Gamma) = 0$

où la longueur algébrique $\overline{L}$ du cycle Γ est égale à $\overline{L}(\Gamma) = \sum_{e \in \Gamma} g(e) \cdot d(e)$

avec : $g(e) = +1$ si e est dans le sens de parcours de Γ, $g(e) = -1$ si e est dans le sens inverse et $d(e)$ est le poids de l'arc e.

IV.3.2. Problématique : existence d'une solution par insertion de registres

Pour les assemblages sans boucle, nous avons vu qu'il est toujours possible de corriger un modèle RTL en insérant des registres entre les IP RTL. Est-ce encore le cas pour les assemblages comprenant des boucles?

Le contre-exemple de la figure 12 montre qu'il n'est pas toujours possible de trouver une solution. D'après la condition suffisante de conservation du comportement, le modèle RTL et le modèle fonctionnel ont un même comportement si le graphe différentiel d'évolution est équilibré. En exploitant le théorème 3 ($\overline{L}(\Gamma) = 0$), on obtient un système d'équations que doit satisfaire le graphe différentiel pour être équilibré. Puisque l'exemple ne contient qu'un seul cycle, ce système d'équations se réduit à une unique équation :

$$(d_{D2} + c_2) + (d_{D5} + c_5) + (d_{D6} + c_6) + (d_{D4} + c_4) = 0$$

[4] Un *cycle* est un chemin non orienté partant d'un sommet pour arriver au même sommet

où d_{Di} est le poids de l'arc e_i sur le graphe différentiel et c_i (inconnue à déterminer) est la correction apportée à cet arc. En supposant que tous les arcs ont des poids égaux à 1, l'équation devient alors : $c_2 + c_5 + c_6 + c_4 = -4$ [1]

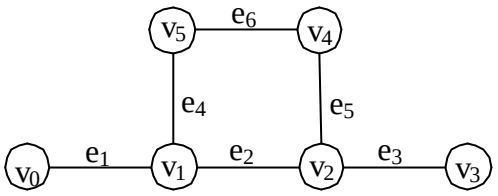


Figure 12. Contre-exemple de l'existence d'une solution par insertion de registres

L'insertion de *C* registres correspond à ajouter un poids $C \geq 0$ sur le graphe différentiel. Or, l'équation [1] montre clairement qu'il n'est pas possible de trouver une solution telle que tous les c_i soient positifs.

Pour obtenir une solution aux équations de contraintes ($\overline{L}(\Gamma) = 0$) pour les assemblages avec boucle, il est alors nécessaire d'étendre la résolution à Z. Tout assemblage avec boucle admet toujours une solution dans Z (démonstration en annexe A.6.). Néanmoins, avoir une correction $C \leq 0$ est équivalent à supprimer des registres sur les fils interconnectant les DSP-IP RTL. Cela est impossible puisqu'il ne contient pas de registre.

IV.3.3. Alternatives à l'insertion de registres

Nous avons vu dans la section précédente, qu'il est impossible de corriger une application avec boucle en insérant des registres. Dans cette section, nous étudions les alternatives à l'insertion de registres.

IV.3.3.1. Le concept de registres négatifs

Le formalisme étendu au cas des boucles montre que les corrections apportées peuvent être négatives. Conceptuellement, cela correspond à insérer des registres « négatifs » dans le

circuit. Par registres «négatifs », nous entendons des registres avançant le signal au lieu de le retarder. Le concept de registre « négatif » a été introduit par Hassoun dans ses travaux sur le retiming architectural [HAS 98][HAS 95]. Ses travaux décrivent deux façons d'implémenter des registres négatifs au niveau porte logique:

- *l'implémentation par précalcul* : le précalcul [HAS 97a] consiste à remplacer une partie du circuit par une fonction qui (pré)calcule (i.e. anticipe) la sortie du registre négatif en utilisant des signaux provenant d'étages précédents de pipeline. Le précalcul nécessite donc une modification d'une partie du circuit. En particulier, [HAS 97a] souligne le fait que cette technique a tendance à dupliquer des opérateurs pour réaliser la fonction de précalcul. Compte tenu de la granularité d'un IP, cette technique ne peut alors être appliquée dans le cadre de l'assemblage d'IP car la duplication d'IP entraînerait un surcoût en surface trop important. Il devient alors nécessaire d'appliquer le précalcul sur une granularité plus faible, mais la conséquence est alors de devoir modifier les IP en interne.
- *l'implémentation par prédiction* : la prédiction [HAS 97b] consiste à spéculer la sortie du registre négatif un cycle avant l'arrivée de la donnée. La spéculation est propagée dans le circuit tant que la donnée réelle n'est pas présente à l'entrée du registre. Lorsque présente, si la prédiction est correcte, la prédiction continue de progresser dans le circuit. Dans le cas contraire, la prédiction est corrigée. Cette technique est réalisée en insérant une FSM de prédiction dans le circuit. Cette FSM a pour tâche de spéculer la sortie du registre négatif et d'apporter une correction en cas d'erreur de spéculation. Cette technique présente deux problèmes majeurs [HAS 95] :
    - *la prédiction nécessite de changer l'interface d'entrée/sortie du circuit* : car le taux de consommation en entrée et de production en sortie devient variable

(pour chaque registre négatif, 2 cycles d'horloges sont ajoutés si la prédiction est erronée et 1 cycle sinon).

- *difficulté du choix de la valeur de prédiction* : le choix de la valeur prédite par la FSM dépend de l'application. Cette valeur est choisie en faisant une étude statistique empirique. Ce choix devient alors difficilement automatisable. Dans l'exemple du routeur chaotique [HAS 96], le choix de la valeur prédite est simple car le signal contenant le registre négatif est quasiment constant. Ainsi, en utilisant la valeur précédente comme prédiction, les erreurs de prédiction restent rares. Dans le cadre des applications de traitement du signal, les signaux sont trop aléatoires pour pouvoir faire des prédictions fiables.

[HAS 96] présente les résultats obtenus par ces deux techniques pour six applications. Dans le meilleur des cas, le surcoût en surface est de 14% pour un gain moyen en fréquence d'horloge de 30 %. Le précalcul et la prédiction restent alors difficilement applicables dans un contexte industriel.

IV.3.3.2. Modification de la FSM

Un moyen de correction est de modifier la FSM globale du circuit. Pour cela, il suffit de faire travailler les IP de la boucle à une fréquence C+1 fois plus grande. Néanmoins, augmenter la fréquence a pour conséquence de réduire la fréquence maximale d'application du circuit et d'augmenter la puissance de consommation.

IV.3.3.3. Ajout de registres dans le modèle fonctionnel

Puisque le graphe différentiel est obtenu par différentiation du graphe d'évolution RTL par le graphe d'évolution fonctionnel, ajouter une correction négative sur le graphe revient à insérer un registre positif (ou retard) entre les IP fonctionnels. Toutefois, ajouter des retards dans le modèle fonctionnel change la fonctionnalité de l'application. A ce niveau, seul le

concepteur peut être juge pour savoir si cette insertion reste tolérable ou non suivant les contraintes de fonctionnalité.

IV.3.3.4. Bilan sur les alternatives

Dans cette section, nous avons étudié des alternatives à l'insertion de registre pour les applications avec boucle. Nous avons vu que les travaux actuels de recherche ne donnent pas encore des résultats suffisants pour être directement applicables. Ils aboutissent plus à des compromis : préserver le comportement mais au détriment de la surface, de la puissance de consommation ou de la fréquence d'application du circuit.

IV.3.4. Algorithme de correction de retard dans le cas des boucles

En attendant d'avoir une implémentation admissible, nous proposons un premier algorithme (tableau 5 et algorithme 3) permettant d'équilibrer le graphe différentiel. La solution obtenue pourra par la suite être optimisée par factorisation de retard. La factorisation de retard reste valable dans le cas des boucles puisque la démonstration du théorème de factorisation de retards ne fait pas d'hypothèse sur le fait que le graphe différentiel est cyclique ou non. Le choix des factorisations dépend du type d'implémentation des corrections négatives. L'algorithme d'optimisation reste actuellement en suspend.

L'algorithme 3 extrait l'arbre Ψ des plus courts chemins[5] par l'algorithme de Dijkstra [SAK 84b]. Chaque sommet *v* est annoté par $L_{min}(v)$, la longueur du plus court chemin depuis l'entrée jusqu'à *v*. Puis, l'algorithme 3 apporte des corrections sur les arcs n'appartenant pas à cet arbre.

La démonstration de l'annexe A.7 montre que l'algorithme permet d'obtenir un graphe différentiel équilibré. Il est intéressant de noter que la démonstration ne fait pas intervenir le fait que Ψ l'arbre des plus courts chemins. Ainsi, l'algorithme reste valable en utilisant (ligne

[5] *Arbre des plus courts chemins* : arborescence qui détermine pour chaque sommet d'un graphe, le plus court chemin de l'entrée à ce sommet.

2) tout algorithme permettant d'extraire un arbre à partir d'un graphe. Le choix de l'algorithme dépend du critère à optimiser (latence, surface ou autres). Ici, nous avons choisi l'algorithme de Dijkstra car permettant d'obtenir une solution optimale en latence. En effet, la démonstration de l'annexe A.3 (démonstration de l'optimalité en latence de l'algorithme 1) ne fait pas d'hypothèse sur la cyclicité ou non du graphe différentiel. L'argumentaire de cette démonstration reste alors encore valable dans le cas des boucles.

$G_D = (V_D, E_D)$	Graphe différentiel à équilibrer
Ψ	Arbre des plus courts chemins
$L_{min}(v)$	Longueur du plus court chemin jusqu'à v
$d_D(e)$	Retard induit par l'arc e
$c(e)$	Correction apportée à l'arc e
$I(e)$	Extrémité initiale de l'arc e
$T(e)$	Extrémité terminale de l'arc e

Tableau 5. Notations de l'algorithme 3

1 :	Initialiser : $\forall e \in V, c(e) = 0$
2 :	Extraire l'arbre Ψ des plus courts chemins
3 :	Poser $\forall e \in \Psi, L_{\min}(T(e)) = L_{\min}(I(e)) + d_D(e)$
4 :	Correction : $\forall e \notin \Psi, c(e) = L(T(e)) - L(I(e)) - d_D(e)$

Algorithme 3. Algorithme de correction de retard

Exemple 3. *Application de l'algorithme 3 sur un exemple de graphe différentiel*

Le graphe de la figure 13 donne un exemple de graphe différentiel avant et après application de l'algorithme de correction de retard. Les annotations $L_{min}(v)$ sont notées au dessus des sommets et les corrections apportées sont notées par '+N'. Les arcs en gras correspondent aux arcs de l'arbre des plus courts chemins.

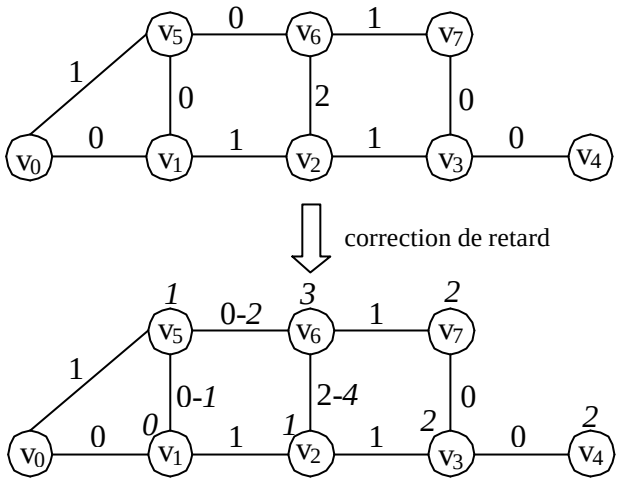


Figure 13. Exemple de graphe différentiel avant (haut) et après (bas) application de l'algorithme de correction de retard

IV.4. Expérimentation et résultats

La méthode de correction de retard a été expérimentée sur un exemple d'application de traitement numérique du signal. Les objectifs de cette expérimentation sont :

- d'illustrer le problème de différence de comportement et sa résolution dans le cas des assemblages sans boucle.
- de comparer la correction de retard par insertion de registres à un autre type de correction : une correction apportée par modification de la FSM du modèle RTL. Les résultats seront comparés en terme de performances de l'architecture RTL et en temps de conception.

L'exemple choisi se compose de deux branches parallèles convergeant sur un additionneur (figure 14). La première branche contient trois filtres à réponse impulsionelle finie (*filtre FIR*) alors que la seconde branche contient un seul filtre FIR. La sortie de l'additionneur est filtrée par un dernier filtre FIR. Un modèle fonctionnel ainsi qu'un modèle RTL ont été réalisés en

Matlab et en VHDL en assemblant les différents éléments suivant la même topologie d'assemblage.

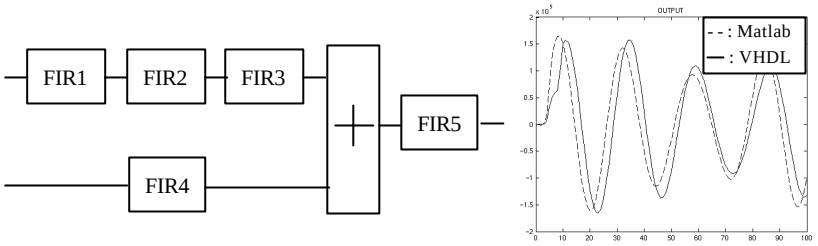


Figure 14. Schématique de l'application (gauche) et visualisation de la différence de comportement en sortie (droite).

Une différence de comportement a été détectée à la sortie de l'additionneur. A la simulation, les sorties produites par le modèle RTL ne sont pas les mêmes que celles produites par le modèle fonctionnel (i.e. les deux courbes de la figure 14 ne se superposent pas parfaitement). Ce problème est dû à un registre de sortie présent dans les filtres FIR RTL et absents des filtres FIR fonctionnels. Ce registre induit un retard supplémentaire du filtre RTL par rapport au filtre fonctionnel. Le graphe différentiel de l'application est illustré par la figure 15. La première branche cumule trois retards jusqu'à l'entrée de l'additionneur alors que la seconde branche cumule un unique retard. Il est nécessaire d'apporter une correction de deux retards sur la seconde branche en vue d'égaliser les deux branches (en italique sur la figure 15).

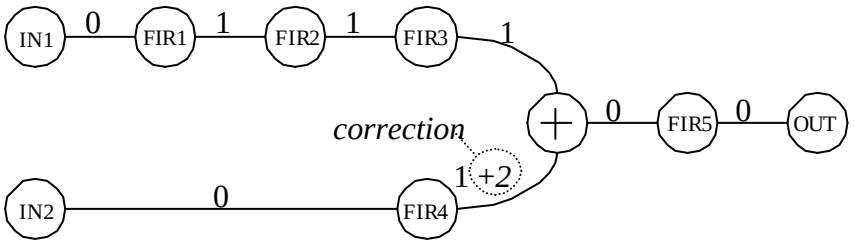


Figure 15. Graphe différentiel corrigé de l'application

Une première correction de retard a été apportée au modèle RTL initial en insérant deux registres sur la seconde branche (figure 16).

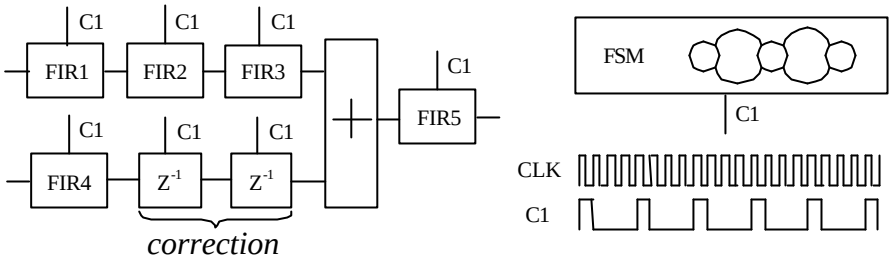


Figure 16. Correction de retard par insertion de registres

A partir du graphe différentiel corrigé, une seconde correction a été réalisée en modifiant la FSM du modèle RTL (figure 17). Initialement, cette FSM produit un signal de contrôle à une fréquence quatre fois plus petite que la fréquence d'horloge. La FSM a été modifiée en produisant un second signal de contrôle identique au signal initial, mais retardé de 2 impulsions (soit 8 cycles d'horloge). Ainsi, le FIR de la seconde branche commence son traitement en retard par rapport au filtre de la première branche. Ces 2 impulsions correspondent à la correction de 2 sur le graphe différentiel (figure 15).

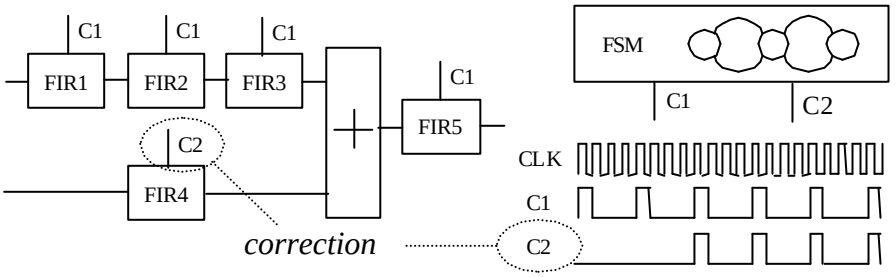


Figure 17. Correction de retard par modification de la FSM

Après ces deux corrections, la sortie du modèle RTL produit exactement les mêmes valeurs numériques que le modèle fonctionnel.

correction / critère	sans	insertion de registres		modification de la FSM	
	global	global	surcoût[6]	global	surcoût[3]
surface	5415 µm²	5639 µm²	4,1 %	5501 µm²	1,6 %
fréq max d'horloge	81,3 MHz	81,5 MHz	0,2 %	81,6 MHz	0,3 %
tps de modification			5 min		40 min

Tableau 6. Comparaison de la correction de retard par insertion de registres et par modification de la FSM

La première ligne du tableau 6 compare les deux types de correction de retard en terme de performances (surface et fréquence maximale d'horloge). Ces résultats ont été obtenus à l'aide de Design Compiler de Synopsys. Au niveau de la surface ajoutée par la correction, la correction de retard par insertion de registres entraîne un surcoût (4,1%) nettement plus important qu'une correction par modification de la FSM (1,6%). Cela est dû au fait que la première correction ajoute des registres propageant des vecteurs de bits (ici, de taille 16) alors que la deuxième correction modifie le modèle RTL initial en apportant des registres sur un seul bit. Le surcoût de la correction par insertion de registres peut sembler important pour notre exemple. Toutefois, il faut considérer ce surcoût comme un majorant car le modèle choisi est relativement simple (i.e. les FIR sont des filtres d'ordre 3).

Pour la fréquence d'horloge maximale admissible, les deux corrections sont équivalentes et ne modifient pas celle du modèle RTL avant correction. En effet, le chemin critique se situe au niveau du chemin de données. La modification de la FSM est suffisamment simple pour que le chemin critique devienne un des chemins de la FSM modifiée. De même, la correction par insertion de registres est réalisée entre les DSP-IP et donc, n'interfère pas avec les parties combinatoires du chemin de données.

[6] surcoût = (valeur avec correction – valeur sans correction) / valeur sans correction

La seconde ligne du tableau 6 compare les deux corrections en terme de temps de modification nécessaire pour corriger le modèle RTL initial. Actuellement, les deux corrections sont faites manuellement à partir des informations données par le graphe différentiel corrigé. La correction par insertion de registres est beaucoup plus simple à mettre en œuvre (5 min pour l'exemple) que la correction par modification de la FSM (40 min). En effet, le premier type de correction consiste juste à connecter des blocs retards pré-conçus et paramétrables aux endroits indiqués par le graphe différentiel et à instancier les valeurs des corrections comme valeur de paramètres. Ce temps de modification augmente linéairement avec le nombre d'endroits où des modifications doivent être apportées. Le systématisme de la mise en œuvre de cette correction montre que cette tâche pourrait être automatisée. La correction par modification de la FSM est plus complexe à appliquer car elle nécessite de modifier globalement l'architecture de la FSM et à repenser ses états. Le coût de modification a tendance à augmenter exponentiellement lorsque les signaux de contrôle deviennent dépendants les uns des autres. Cette modification devient alors coûteuse en temps de conception et rapidement source d'erreurs.

IV.5. Conclusion

Nous avons vu dans ce chapitre, que le passage d'un assemblage de DSP-IP fonctionnels vers un assemblage de DSP-IP RTL pouvait produire une différence de comportement entre les deux assemblages. E.g. pour le même stimulus d'entrée, les deux assemblages ne produisent plus les mêmes valeurs numériques.

Les causes de ce problème sont dus à des retards présents dans les DSP-IP RTL et absents des DSP-IP fonctionnels. Ces retards additifs sont liés aux contraintes d'implémentation. Dans le cas des applications sans boucle, un moyen de compenser ces retards additifs est d'ajouter des registres dans le chemin de données entre les DSP-IP RTL. Cette solution

présente l'avantage d'être non intrusive. Réalisée manuellement, cette solution reste consommatrice en temps de conception et source d'erreurs pour localiser les endroits où la différence de comportement intervient, pour déterminer comment apporter les corrections et pour implémenter les corrections. De plus, des optimisations peuvent être perdues lors de la recherche d'une solution, car la résolution du problème de correction de retards par insertion de registres n'admet pas une solution unique, mais plusieurs solutions.

Nous avons défini une méthode automatique (appelée *méthode de correction de retard*) permettant la détermination du nombre et la localisation des registres de correction dans le cas des applications sans boucle. Le problème de différence de comportement est d'abord formalisé comme un problème de théorie des graphes. A partir des graphes des modèles fonctionnel et RTL, deux algorithmes ont été développés. Le premier fournit une solution optimale en latence alors que le deuxième obtient une solution optimale en nombre de registres insérés. La validité de la méthode ainsi que l'optimalité des corrections sont démontrées mathématiquement.

La correction de retard par insertion de registres a été comparée expérimentalement à une correction de retard par modification de la FSM globale, autre moyen de correction possible. Il a été constaté qu'apporter une correction par insertion de registres demande un coût de modification du modèle RTL initial beaucoup moins important qu'une correction par modification de la FSM. Ce coût par modification de la FSM a tendance à augmenter rapidement avec la complexité de la FSM et de devenir source d'erreurs. En contrepartie, le surcoût en surface ajoutée par les registres est plus grand que le surcoût d'une correction par modification de la FSM.

Pour les applications contenant des boucles, nous montrons que l'insertion de registres ne peut être employée et discutons les alternatives d'implémentation : nous constatons alors que ces alternatives donnent des résultats insuffisants pour être applicables. Le problème de

différence de comportement est reformulé pour le cas des boucles. Un premier algorithme est fourni pour donner une solution permettant d'équilibrer le graphe différentiel. L'optimisation de cette solution dépend du type d'implémentation de la correction. En attendant une alternative d'implémentation efficace, l'optimisation de la solution est laissée en perspective.

Chapitre V. Implémentation de la méthodologie en un flot semi-automatique de conception

Résumé : dans ce chapitre, nous présentons l'implémentation de la méthodologie proposée (chapitre III) en un flot de conception semi-automatique complet (appelé *SigMATor*). SigMATor utilise COLIF comme forme intermédiaire pour représenter l'application à chaque étape du flot. COLIF, développé au sein de l'équipe SLS, est un langage permettant la description topologique d'une application ainsi que la modélisation hétérogène en langages et en niveaux d'abstraction. Des outils indépendants ont été développés pour réaliser chaque étape du flot. Nous présentons également l'interface graphique permettant de lier les différents outils sous un environnement homogène et convivial.

Sommaire :

V.1. Introduction

Le chapitre III a présenté la méthodologie proposée de conception et de validation. Cette méthodologie est définie par un ensemble d'étapes systématiques pour concevoir une macro-cellule ASIC à partir d'une description fonctionnelle.

Plusieurs outils ont été développés pour que cette méthodologie (i.e. ensemble de méthodes indépendantes des outils) devienne un flot semi-automatique (i.e. ensemble d'outils mappant les méthodes) de conception complet. Nous entendons par semi-automatique le fait que l'assemblage de DSP-IP est pleinement automatisé alors que leurs conceptions restent manuelles. Partant d'une description en Matlab [ZER 01][ZER 00] pour obtenir une architecture RTL en VHDL, ce flot est appelé *SigMATor* pour *SIGnal processing MATlab TO Rtl flow*.

Les objectifs de ces outils sont :

- réduire le temps de conception lors de l'application de la méthodologie.
- libérer le concepteur des tâches fastidieuses et sources d'erreurs pour qu'il puisse concentrer ses efforts sur les tâches essentielles : la modélisation et la validation fonctionnelle ainsi que l'exploration architecturale.

Le développement de ces outils est un travail effectué en commun avec une équipe de quatre stagiaires [AID 03][CAR 03][WAS 03].

V.1.1. Plan

Ce chapitre se compose de 5 sections. La section 2 présente la forme intermédiaire utilisée (COLIF). La section 3 est dédiée à l'organisation et aux conventions de nom de la librairie d'IP sur laquelle vont travailler les différents outils. La section 4 décrit le synoptique général du flot ainsi que l'interaction entre les outils. La section 5 conclut ce chapitre.

V.2. Forme intermédiaire utilisée : COLIF

La transformation du modèle fonctionnel (ici, en Matlab) en un modèle RTL (en VHDL) nécessite l'exploitation d'une forme intermédiaire commune permettant la représentation des différents DSP-IP et de leurs interconnexions durant toutes les étapes du flot de conception [ZER 02]. La forme intermédiaire employée est COLIF [CES 01a] [CES 01b], langage développé au sein de l'équipe SLS.

COLIF (*Co-design Language-Independent Format*) est un langage permettant la description topologique d'une application ainsi que la modélisation hétérogène en langages et en niveaux d'abstraction. Il est basé sur la séparation entre la communication et le comportement. Ce modèle permet la spécification des systèmes complexes par une approche « diviser pour mieux régner ».

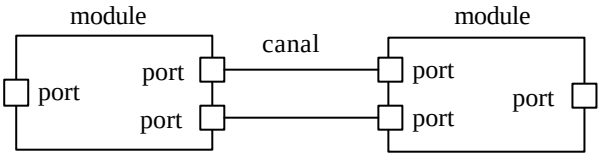


Figure 18. Concepts fournis par COLIF : module, port, canal

COLIF offre trois concepts de base nécessaires à toute représentation topologique (figure 18) : le *module*, le *port* et le *canal*. Indépendamment du langage et du niveau d'abstraction, un système complexe peut être modélisé en COLIF comme un ensemble de *modules* hiérarchiques représentant une *architecture* abstraite. Les différents modules communiquent par des *canaux de communication* (appelé aussi *net* dans la terminologie COLIF) via leurs *interfaces*. Un module peut être *hiérarchique*, contenant un ou plusieurs modules, soit encore, il peut représenter une feuille dans la hiérarchie, présentant une *description élémentaire*. Le comportement peut être une tâche élémentaire ou un processus complexe. Chaque module est

connecté aux canaux de communication par une interface. L'interface de chaque module contient un ensemble de *ports* (des points de communication d'un module vers l'extérieur) et les *opérations effectuées sur ces ports* (spécifiant l'interaction entre un module et le reste du système). COLIF permet une représentation hiérarchique des trois concepts *module*, *port* et *net*.

Pour faciliter l'intégration d'outil et l'échange d'informations entre les différents environnements de synthèse, COLIF se base sur le langage XML [CES 00a]. De plus, COLIF est lié avec un outil de visualisation nommé *cview*.

Dans notre flot, le choix de COLIF comme forme intermédiaire est motivé par trois raisons :

- *COLIF permet de représenter les descriptions topologiques et hiérarchiques* : puisque la méthodologie proposée est basée sur l'assemblage d'IP, COLIF présente tous les concepts (module, port, canal, hiérarchie) nécessaires au développement de notre flot.
- *COLIF est une modélisation indépendante du langage et du niveau d'abstraction* : cela permet une ouverture vers des descriptions d'entrée et de sortie autres que Matlab et VHDL.
- *COLIF est utilisé par ROSES* [DZI 03] : cela permettra l'insertion ultérieure de notre flot dans un flot plus vaste orienté vers la conception des circuits intégrés monopuces.

V.3. Organisation de la librairie et convention de noms

La méthodologie de conception proposée se base sur l'utilisation d'IP préconçus organisés dans une librairie. Nous normalisons ici l'organisation de cette librairie ainsi que le nom des différents répertoires et fichiers pour deux raisons :

- L'exploitation d'outils automatiques nécessite des règles strictes pour qu'ils puissent retrouver les informations utiles dans la librairie (e.g. nom des ports d'un IP).

– Le maintient efficace des librairies et la portabilité vers les clients.

Pour que ces règles ne soient pas trop contraignantes lorsqu'un utilisateur souhaite ajouter un nouvel outil dans la librairie, un outil permettant l'insertion facile d'un nouvel IP est disponible (c.f. §V.4.1.).

A la racine de l'arborescence, la librairie d'IP se compose de quatre répertoires (figure 19) : LIB, UTILS, COLIF et IP. Le répertoire LIB contient les résultats de compilation des sources VHDL. Le répertoire UTILS regroupe des fonctions basiques utilisées pour la description d'IP en Matlab et en VHDL (e.g. différents types d'arrondi). Le répertoire COLIF contient les structures COLIF caractérisant les interfaces des IP. Le répertoire IP regroupe une arborescence de répertoires dont un répertoire-feuille contient les descriptions d'un IP Matlab et VHDL.

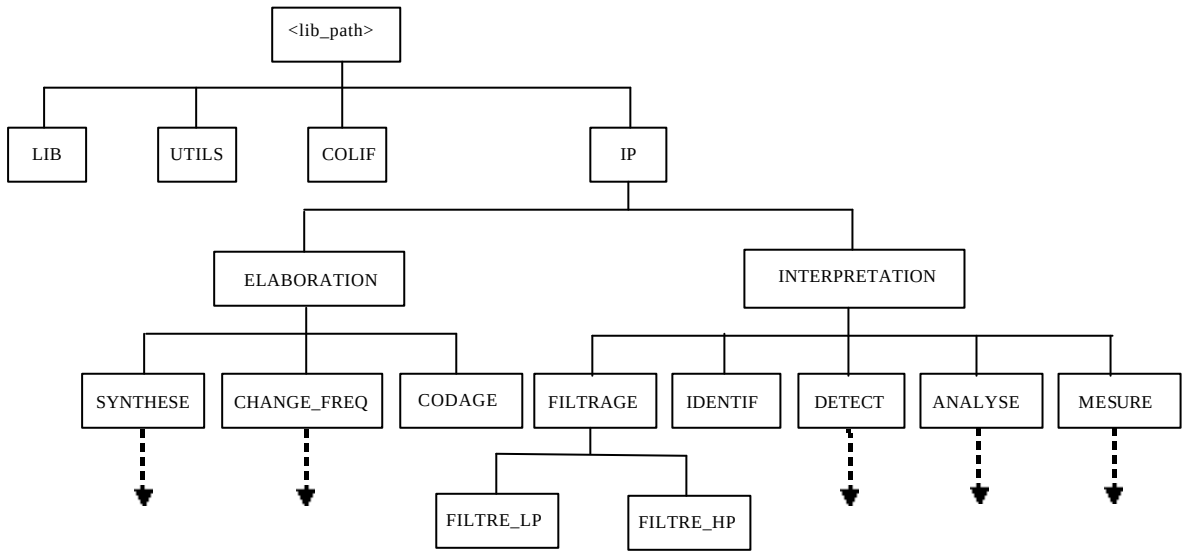


Figure 19. Organisation des répertoires

Pour faciliter la reconnaissance d'un IP, L'arborescence suit la classification des fonctions de traitement du signal décrite par [COT 00]. Ces fonctions peuvent se diviser en deux catégories : l'élaboration des signaux (i.e. incorporation des informations) et l'interprétation

des signaux (extraction des informations). Les principales fonctions intégrées dans ces deux parties sont les suivantes :

- *élaboration des signaux* :
    - *synthèse* : création de signaux de forme appropriée en procédant par exemple à une combinaison de signaux élémentaires.
    - *modulation, changement de fréquence* : moyen permettant d'adapter un signal aux caractéristiques fréquentielles d'une voie de transmission.
    - *codage* : traduction en code binaire (quantification), etc.
- interprétation des signaux :
    - *filtrage* : élimination de certaines composantes indésirables.
    - *détection* : extraction du signal d'un bruit de fond. (corrélation).
    - *identification* : classement d'un signal dans des catégories préalablement définies.
    - *analyse* : isolement des composantes essentielles ou utiles d'un signal de forme complexe (i.e. transformée de Fourier).
    - *mesure* : estimation d'une grandeur caractéristique d'un signal avec un certain degré de confiance (valeur moyenne, etc.).

La figure 20 montre l'organisation de répertoires et les fichiers d'un répertoire-feuille. Ce répertoire contient tous les fichiers nécessaires à la compilation, simulation et validation d'un IP fonctionnel et RTL. Ainsi, les IP sont indépendants les uns des autres. La variable <IP> est substituée par le nom de l'IP lors de la création de ce répertoire.

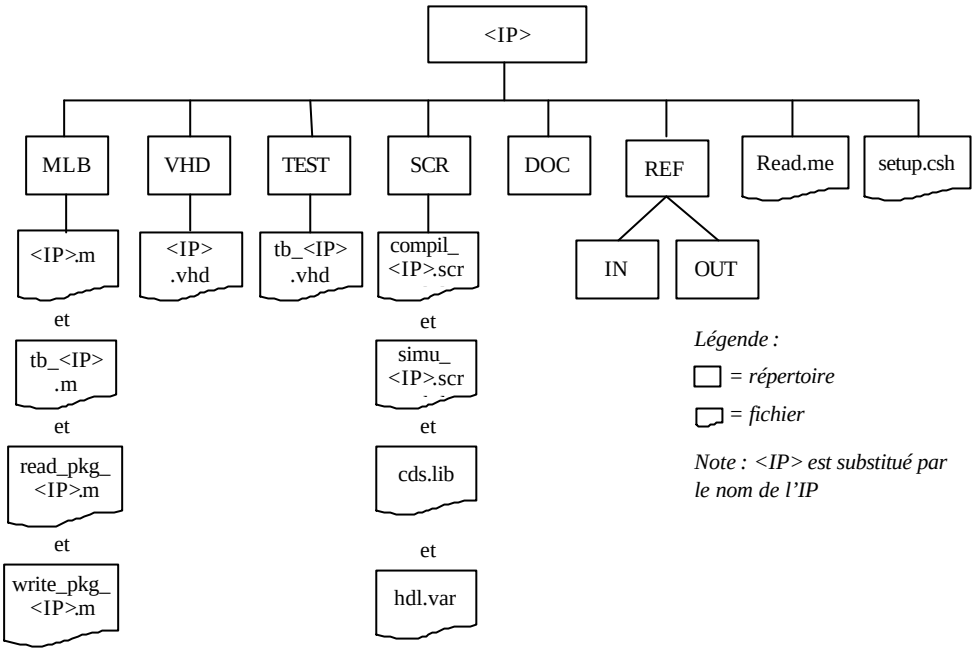


Figure 20. Organisation des répertoires et des fichiers pour un IP

Les répertoires et fichiers sont :

- *le répertoire MLB* : contient les sources de la description fonctionnelle en Matlab. En particulier, il regroupe les fichiers :
  - <IP>.m : description du DSP-IP fonctionnel.
  - tb_<IP>.m : testbench de DSP-IP. Il exécute le modèle fonctionnel, compile le modèle VHDL, exécute la simulation VHDL, compare les résultats de simulation du modèle Matlab avec ceux du modèle VHDL.
  - write_pkg_<IP>.m : fonction Matlab qui écrit les paramètres définis pendant la modélisation fonctionnelle, en un fichier VHDL exploitable par le modèle RTL. Ainsi, le modèle fonctionnel et RTL partagent le même fichier de valeurs de paramètre. Cela assure une cohérence entre les valeurs de paramètre partagées entre les deux niveaux d'abstraction.

- read_pkg_<IP>.m : fonction Matlab qui lit un fichier VHDL de valeurs de paramètre pour les exploiter dans le modèle fonctionnel.

– *le répertoire VHD* : contient les sources de la description RTL en VHDL. En particulier, il est composé de :

- <IP>.vhd : description du DSP-IP RTL. La description de l'architecture de cet DSP-IP est laissée au concepteur.

– *le répertoire TEST* : contient les sources du testbench de la description RTL en VHDL. En particulier, il regroupe :

- tb_<IP>.vhd : description du tesbench du DSP-IP RTL.

– *le répertoire SCR* : contient les scripts de compilation et de simulation de l'IP RTL. En particulier, il contient :

- compil_<IP>.scr : fichier de compilation du DSP-IP RTL.
- simu_<IP>.scr : fichier de simulation de l'IP RTL.
- hdl.var et cds.lib : fichiers utilisés pour la configuration du simulateur NC_sim de cadence. Complètement généré.

– *le répertoire REF/IN* : stimuli d'entrée des DSP-IP fonctionnel et RTL.

– *le répertoire REF/OUT* : résultats de simulation des DSP-IP fonctionnel et RTL.

– *le répertoire DOC* : documentations techniques des DSP-IP fonctionnel et RTL.

– *le fichier Read.me* : informations sur l'organisation des répertoires et l'utilisation des DSP-IP.

– *Le fichier setup.csh* : fichier de configuration des outils.

V.4. Flot de conception et de validation

La figure 21 représente le synoptique de l'ensemble des outils. Le langage COLIF est utilisé pour décrire l'application à chaque étape du processus de conception. La description

d'entrée est réalisée en langage Matlab [MAT 03]. La sortie est une architecture RTL en VHDL. La figure 21 est analogue à la figure 37 de la méthodologie (chapitre III et annexe B.3.), mais est utilisée ici pour montrer les différents outils et les différents modèles COLIF obtenus.

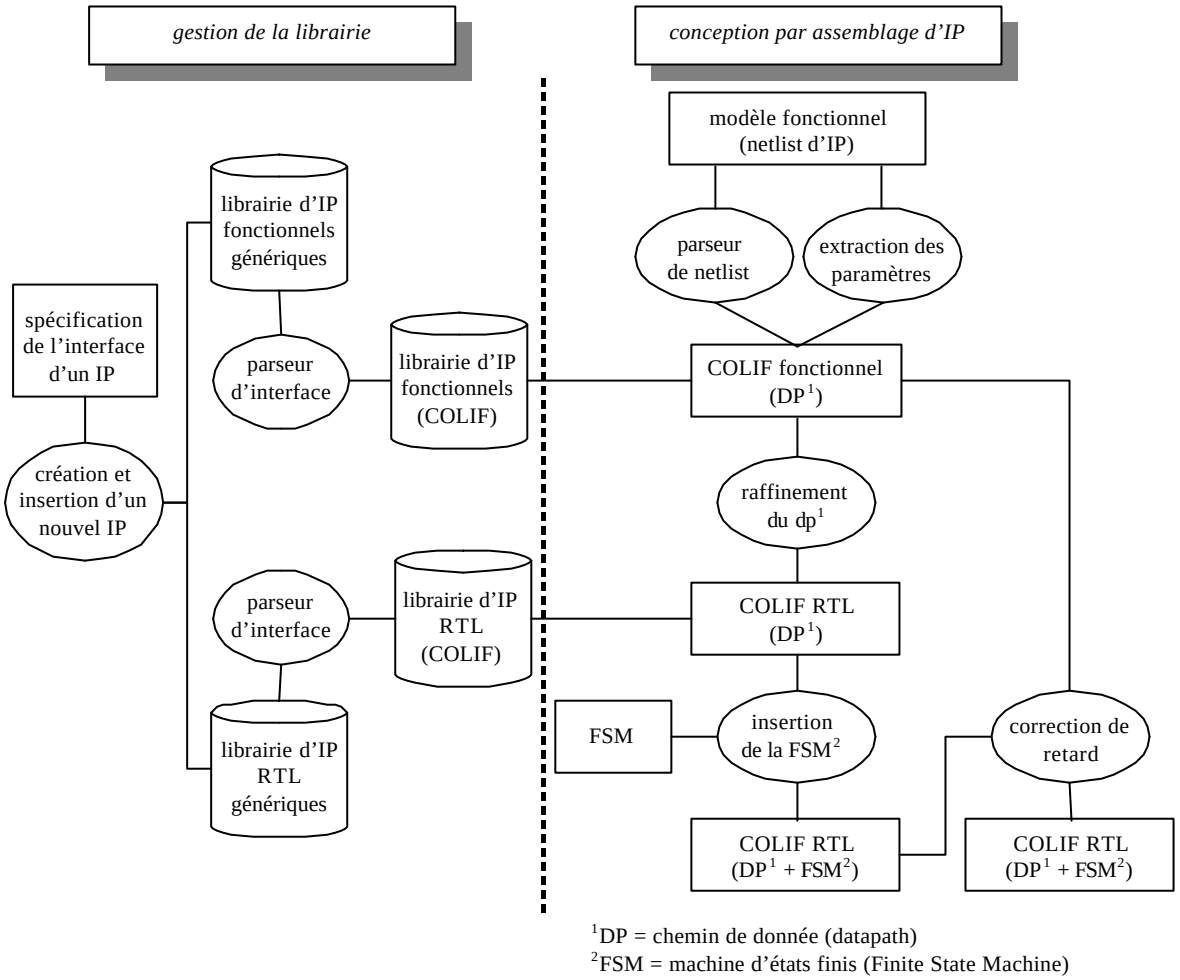


Figure 21. Synoptique du flot et interactions entre les outils

Le flot se compose de deux parties :

– la partie «*gestion de la librairie*» : gère la librairie de DSP-IP. Cette partie est constituée de trois outils :

- *création et insertion d'un nouvel DSP-IP* : cet outil a pour tâche de générer l'environnement de conception et de validation d'un DSP-IP lorsqu'un nouvel DSP-IP doit être inséré dans la librairie.
- *deux parseurs d'interface* : le premier parseur analyse la partie déclarative (i.e. interface au niveau fonctionnel) de chaque DSP-IP Matlab et fournit une structure COLIF contenant toutes les informations caractérisant l'interface. Le second parseur analyse également la partie déclarative (i.e. interface au niveau RTL) de chaque DSP-IP RTL et fournit de même une structure COLIF contenant toutes les informations relatives à l'interface.

– la partie « *conception par assemblage d'IP* » : a pour rôle de réaliser toutes les étapes du flot de conception. Il se compose de cinq outils :

- *parseur de netlist* : ce parseur analyse le modèle d'assemblage en Matlab et génère la netlist COLIF équivalente.
- *extraction des paramètres* : à partir des valeurs de paramètre inclues dans le modèle Matlab, cet outil génère un fichier VHDL contenant ces valeurs de paramètre.
- *raffinement du chemin de données* : à partir de la netlist COLIF décrivant le modèle d'assemblage au niveau fonctionnel, cet outil génère la netlist COLIF équivalente au niveau RTL.
- *insertion de la FSM* : à partir d'une FSM décrite en VHDL par le concepteur, cet outil instancie cette FSM dans le modèle RTL et génère les connections entre la FSM et le chemin de données.
- *correction de retard* : cet outil a pour tâche d'exécuter les algorithmes permettant l'insertion de registres pour la correction du problème de différence de comportement.

Tous les outils sont indépendants les uns des autres. Une GUI (*Graphical User Interface*) a été développée pour lier les différents outils sous un environnement commun, homogène et convivial.

Actuellement, le parseur de netlist, le raffinement, la correction de retard et la GUI sont développés et opérationnels. La création d'un nouvel IP, l'extraction de paramètres, le parseur d'interface Matlab sont en cours de développement. Le parseur d'interface VHDL, l'insertion de la FSM et la génération de code VHDL sont à l'état de spécification écrite et ces étapes sont à ce jour réalisées manuellement.

Les différents outils seront décrits dans les sections qui suivent.

V.4.1. Création et insertion d'un nouvel IP

Cet outil a pour tâche de générer automatiquement l'environnement de conception et de validation nécessaire à l'insertion d'un nouvel IP dans la librairie. Cet outil crée l'interface des DSP-IP fonctionnels et RTL, l'organisation des répertoires ainsi que les patrons des différents fichiers utilisés.

Lors de la création de l'environnement de conception et de validation d'un DSP-IP, certains fichiers sont générés complètement ou partiellement. La figure 22 donne un exemple de fichier généré : la description de l'IP RTL. Dans cet exemple, les notations <XXX> désignent les variables spécifiées par le concepteur et substituées par l'outil.

```
-- ************************************
--      INTERNAL USE ONLY
--      <company_name> - <division_name>
-- ************************************
--
-- Project name   :
-- Library name   :
-- Global package :
--
-- Entity         : <ip_name>
-- Architecture   : rtl
-- Designer name  : <author_name>
-- Company name   : <company_name>
-- Division       : <division_name>
--
-- Description    : TO_BE_DEFINED
--
-- Parameters     : TO_BE_DEFINED
--
-- Input ports    : TO_BE_DEFINED
--
-- Output ports   : TO_BE_DEFINED
--
-- Creation date  : <date>
(<author_name>)
--      Modification    :
--
-- ******** REVISION HISTORY ***********
--
--   Date     Modification    Initials
--   ----     ------------    --------
--  <date>     Creation  <author_initial>

library ieee;
use ieee.STD_LOGIC_1164.all;
use ieee.STD_LOGIC_unsigned.all;
use ieee.STD_LOGIC_signed.all;
use ieee.std_logic_misc.all;
use ieee.std_logic_arith.all;

library UTILS;
use UTILS.utils_pkg.all;
```

page 1

```
---------------------------------------
entity <ip_name> is
  generic (
    <param_name1> : <param_datatype1>;
        ...
    <param_nameN> : <param_datatypeN-1>
        );
  port (
    <in_name1> : in <in_datatype1>;
        ...
    <in_nameN> : in <in_datatypeN>;
    <out_name1> : out <out_datatype1>;
        ...
    <out_nameN> : out <out_datatypeN>
        );
end <ip_name>;

---------------------------------------
architecture rtl of <ip_name> is

-- data type and constant declaration
-- ----------------------------------
TO_BE_DEFINED

-- component declaration
-- ---------------------
TO_BE_DEFINED

-- signal declaration
-- ------------------
signal <in_name1>_s : <in_datatype1>;
...
signal <in_nameN>_s : <inport_datatypeN>;
signal <out_name1>_s : <out_datatype1>;
...
signal <out_nameN>_s : <out_datatypeN>;

TO_BE_DEFINED

---------------------------------------
begin

-- connect I/O to signals
-------------------------
<inport_name1>_s <= <inport_name1>;
...
<inport_nameN>_s <= <inport_nameN>;
<outport_name1> <= <outport_name1>_s;
...
<outport_nameN> <= <outport_nameN>_s;

TO_BE_DEFINED

end rtl;
```

page 2

Figure 22. Exemple de fichier généré (IP RTL)

V.4.2. Les parseurs d'interface

Le premier parseur analyse la partie déclarative (i.e. interface au niveau fonctionnel) de chaque DSP-IP Matlab et fournit une structure COLIF (module COLIF) contenant toutes les informations caractérisant l'interface. Les informations extraites sont :

- *le nom de la fonction* (i.e. le nom de l'IP)
- *une liste de valeurs de retour* correspondant aux signaux de sortie.
- *une liste de paramètres*. Ces paramètres peuvent se diviser en trois types :
    - les paramètres correspondants aux signaux d'entrée traités par l'IP.
    - une référence vers un fichier texte contenant la liste des valeurs de paramètre de l'application.
    - un identificateur (entier) permettant de distinguer une instance de l'IP des autres instances.

Par convention, pour distinguer les différents types de paramètres, nous adoptons la convention suivante :

- les (N-2) premiers paramètres concernent les signaux d'entrée (N étant le nombre de paramètres de l'IP).
- l'avant-dernier paramètre est la référence sur le fichier de paramètres.
- le dernier paramètre est l'identificateur.

L'interface d'un IP fonctionnel en Matlab est spécifiée par le patron de la figure 23.

```
function [<outport_list>]
     = <IP_name> (<inport_list>,<parameter_file>,<id>)
où :
- <outport_list> ⟵ <outport1_name> , … , <output_portN_name>
- <inport_list>  ⟵ <inport1_name> , … , <inportN_name>
```

Figure 23. Patron de la déclaration de fonction en Matlab

La figure 24 donne un exemple d'IP Matlab et la structure COLIF associée.

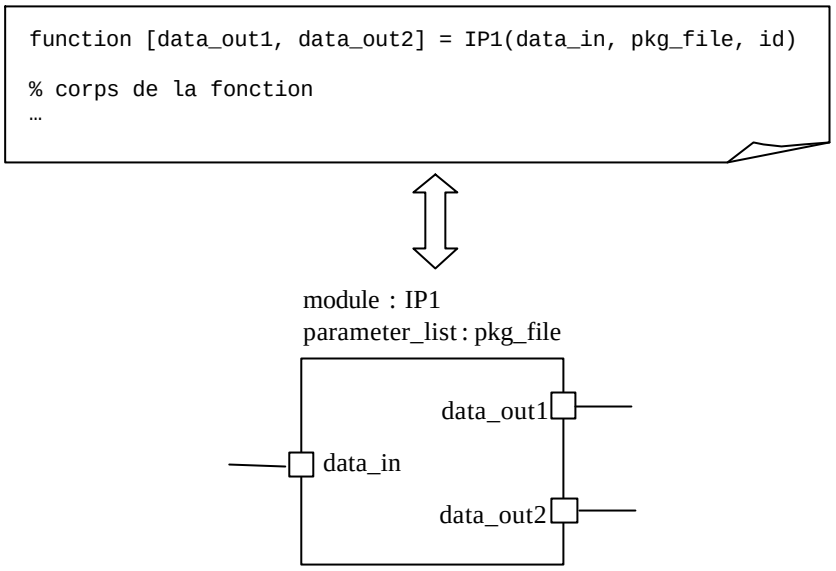


Figure 24. Exemple d'IP Matlab (haut) et structure COLIF associée (bas)

De même, le second parseur analyse la partie entité au sens VHDL du terme (i.e. interface au niveau RTL) de chaque IP RTL et fournit une structure COLIF (module COLIF) contenant les informations relatives à l'interface. Les informations extraites sont :

- *le nom de l'entité* (i.e. nom de l'IP)
- *une liste de paramètres*. Chaque paramètre est défini par un nom, un type de donnée et optionnellement par une valeur par défaut.
- *une liste de ports*. Chaque port est défini par un nom, une direction (IN ou OUT) et un type de donnée (std_logic, std_logic_vector, etc.). Il est important de noter qu'un IP RTL contient plus de ports qu'un IP fonctionnel. En effet, un IP fonctionnel contient les ports liés au flot de données alors qu'un IP RTL contient les ports liés au flot de données ainsi que des ports tels que le port d'horloge, de reset, de contrôle du flot de données, de tests, etc.

L'interface d'un IP fonctionnel en Matlab est spécifiée par le patron de la figure 25.

```
entity <IP_name> is
       generic( <param_list> );
       port ( <port_list> );
end <IP_name>;
```

où :

- <param_list> — <param_decl1> ; … ; <param_declN>
- <port_list> — <port_decl1> ; … ; <port_declN>
- <param_decl> — <param_name> : <param_type> := <default_val>
- <param_decl> — <param_name> : <param_type>
- <port_decl> — <port_name> : <direction> <port_type>
- <direction> — IN | OUT

Figure 25. Patron de la déclaration d'entité en VHDL

La figure 26 donne un exemple d'IP VHDL et la structure COLIF associée.

```
entity IP1 is
generic (
   nbit : integer );
port (
   data_in   : in  std_logic_vector(nbit downto 0);
   data_out1 : out std_logic_vector(nbit downto 0);
   data_out2 : out std_logic_vector(nbit downto 0);
   clk       : in  std_logic;
   nrst      : in  std_logic;
   enable    : in  std_logic);
end IP1;


-- architecture
…
```

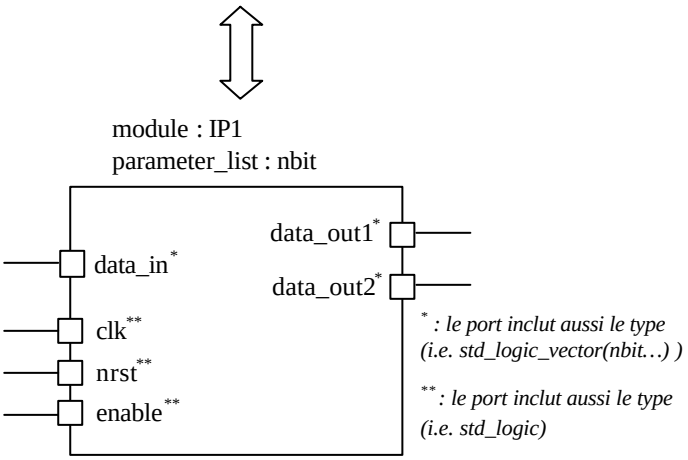


Figure 26. Exemple d'IP VHDL (haut) et structure COLIF associée (bas)

Ces analyses des interfaces des IP sont faites une fois et une seule fois pour toute lors de l'insertion d'un nouvel IP dans la librairie. Ces informations seront utilisées pour les différents outils de la partie « conception par assemblage d'IP ».

V.4.3. Le parseur de netlist

Ce parseur a été développé pour transformer la partie flot de donnée décrite au niveau fonctionnel en langage Matlab en une description fonctionnelle de ce même flot de donnée en langage COLIF.

Cet outil se compose d'un analyseur lexical et syntaxique du langage Matlab. Il parcourt le code Matlab pour identifier les appels de fonctions. Si la fonction appelée est un IP de la librairie, le module COLIF pré-caractérisé (c.f. §V.3.2) équivalent est instancié. Sinon, le parseur considère cette fonction comme étant d'un niveau inférieur dans la hiérarchie. Dans ce cas, l'analyse de code est récursivement faite sur la nouvelle fonction trouvée.

Dans les deux cas, les variables d'entrée et de sortie de la fonction sont utilisées comme canal COLIF entre les différents modules COLIF. Le canal aura alors le nom de la variable. Lorsque la variable est une variable d'entrée (resp. de sortie) d'une fonction, elle est traduite en COLIF comme étant une connexion entre un port d'entrée (resp. sortie) du module COLIF et le canal. La position de la variable dans la liste des paramètres de la fonction ou la liste des valeur de retour définit le port d'entrée ou de sortie qui sera connecté : si cette variable est la $N^{ième}$ valeurs de retour, elle sera connectée au $N^{ième}$ port de sortie du module COLIF issu du parseur d'interface (idem pour les ports d'entrée).

La figure 27 donne un exemple de modèle fonctionnel (en Matlab) et la structure COLIF associée après utilisation du parseur. Le but de cet exemple est de montrer le lien entre la description Matlab et la forme intermédiaire COLIF générée par le parseur. En particulier, l'exemple montre que l'application (nommée *dut*) contient trois IP (*IP1*,*IP2*,*IP3*). Ces IP

correspondent à trois appels de fonction au niveau du modèle Matlab et à trois instances de module au niveau de la forme intermédiaire COLIF. Au niveau des canaux, l'exemple montre que la variable *s2* est une variable de sortie de la fonction *IP1*. Un canal COLIF nommé *s2* est alors créé. Comme la variable *s2* est la deuxième variable de sortie de *IP1*, le canal sera connecté au deuxième port du module *IP1*, soit le port *data_out2*. De même, la variable *s2* est la deuxième variable d'entrée de la fonction *IP3*. Donc, le canal *s2* sera connecté au deuxième port d'entrée (*data_in2*) du module *IP3*.

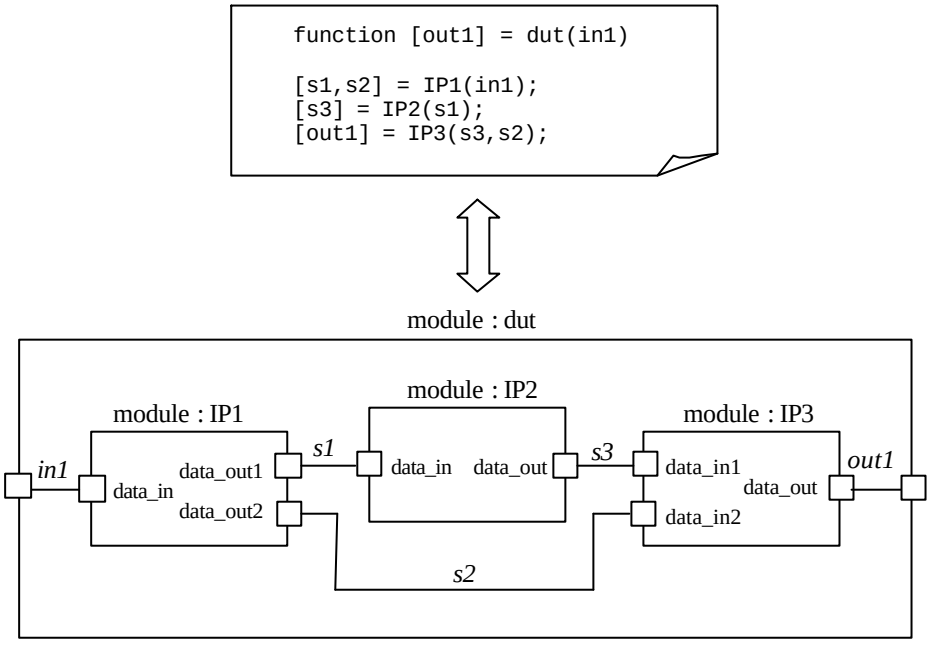


Figure 27. Exemple de modèle fonctionnel (haut) et structure COLIF associée (bas)

V.4.4. L'outil de raffinement

L'outil de raffinement a été développé pour transformer la structure COLIF représentant le modèle fonctionnel en une structure COLIF représentant le modèle RTL (partie chemin de donnée uniquement).

Cette transformation est réalisée en deux étapes :

- en substituant les modules COLIF décrivant l'interface des IP fonctionnels par les modules COLIF décrivant les interfaces des IP RTL.
- puis, en reproduisant la connectique ports-canaux telle que décrite dans le modèle fonctionnel. La reproduction de la connectique est faite par convention de nom : lorsqu'un port a le même rôle au niveau fonctionnel et au niveau RTL, alors il a le même nom pour l'IP-DSP fonctionnel et RTL.

Les ports n'appartenant pas au chemin de données (horloge, reset par exemple) sont laissés non connectés. La connexion de ces ports est actuellement faite à la main. Un fichier de diagnostic est généré par l'outil pour informer l'utilisateur sur les ports non connectés.

La figure 28 donne un exemple de structure COLIF avant et après raffinement.

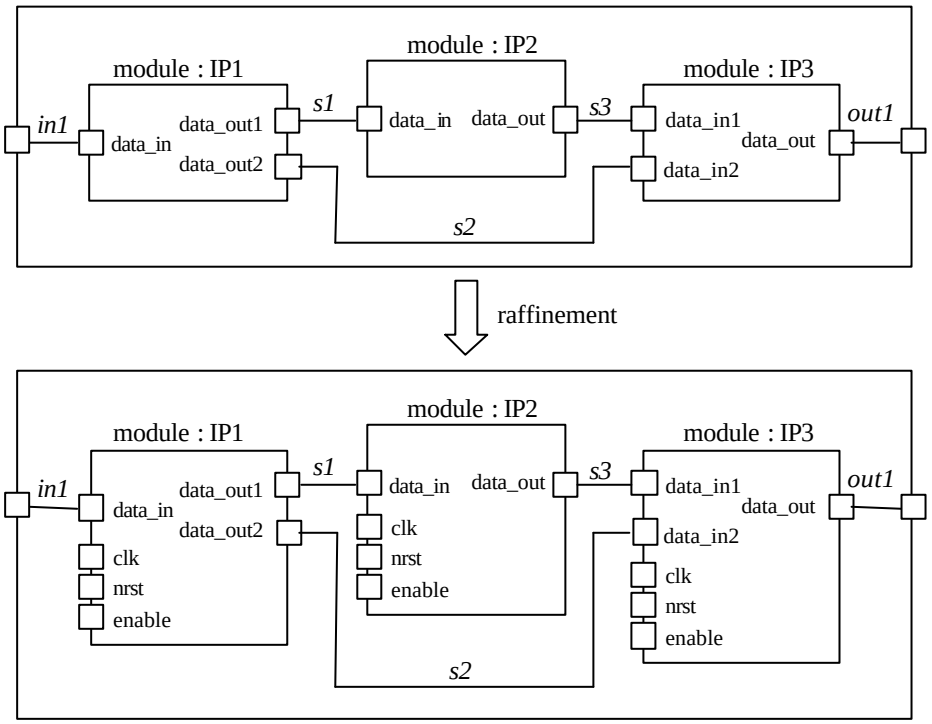


Figure 28. Exemple de structure COLIF avant et après raffinement

V.4.5. L'insertion de la FSM

Cet outil a pour but de d'insérer automatiquement une FSM décrite par le concepteur et de réaliser les connectiques entre les ports de la FSM et ceux des IP. Cet outil n'étant pas encore développé, nous donnons ici les grandes lignes de sa spécification. Cet outil se décompose en deux tâches principales :

- *la transformation de la FSM décrite en VHDL en un module COLIF* : pour cela, le parseur d'interface VHDL (c.f. §V.3.2) extrait les informations relatives à l'interface de la FSM et crée le module COLIF portant ces informations. Ce module COLIF est inséré dans la structure COLIF issue du raffinement.
- *la distribution des connectiques entre les ports de la FSM et ceux des DSP-IP* : l'idée est ici que la plupart des ports de type horloge, reset ou de contrôle du flux de données sont généralement connectés aux mêmes ports de la FSM. Donc, il est inutile au concepteur de spécifier tous les connections entre la FSM et les IP à tous les niveaux de la hiérarchie. Il est suffisant de spécifier les connections pour les plus hauts niveaux de la hiérarchie, ces connections seront propagées à travers les niveaux hiérarchiques.

La figure 29 donne un exemple de structure COLIF avant et après insertion de la FSM.

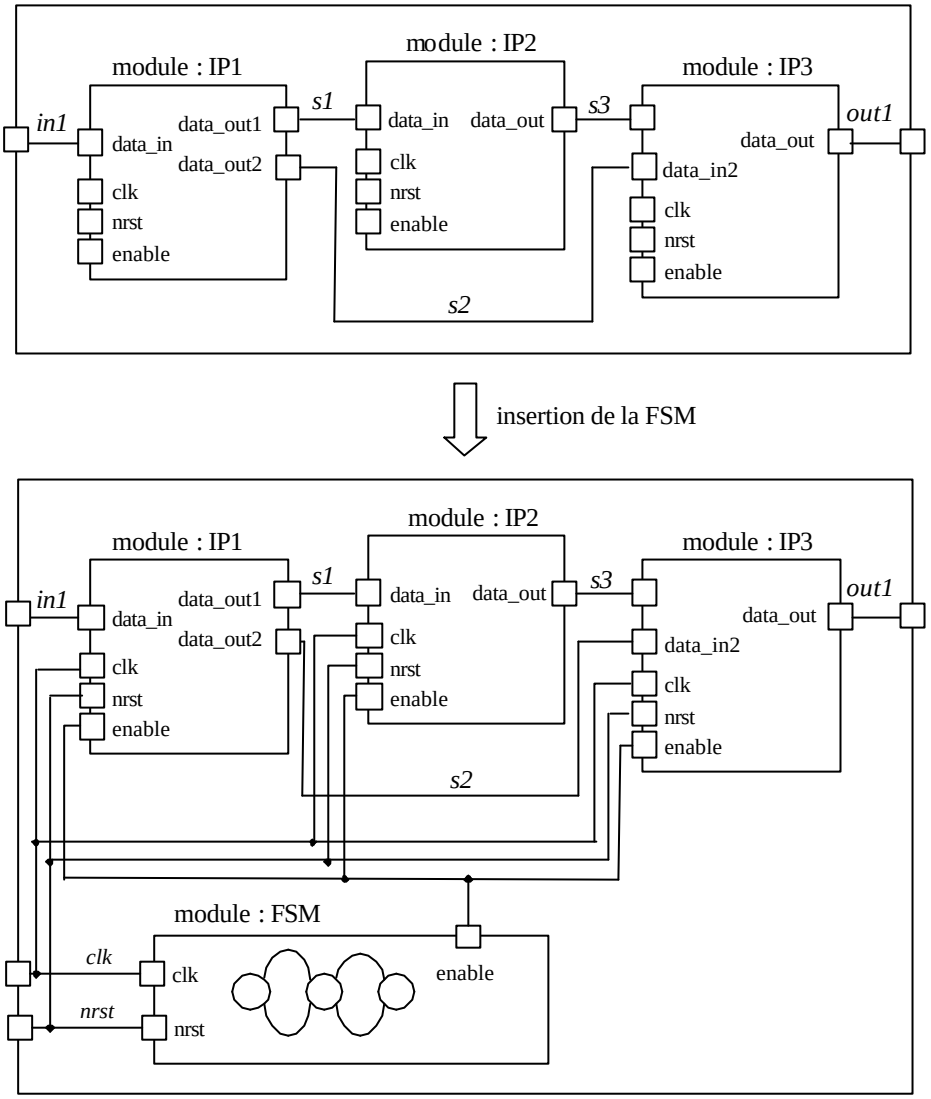


Figure 29. Exemple de structure COLIF avant et après insertion de la FSM

V.4.6. L'outil de correction de retard

L'outil de correction de retard met en œuvre les deux algorithmes décris dans le chapitre IV. Il réalise trois tâches principales :

- *construction du graphe différentiel* d'évolution à partir de la description COLIF fonctionnelle et de la description COLIF RTL.
- *application des algorithmes et détermination d'une solution* : la valeur de la correction est portée par les ports d'entrée des modules.

– *insertion des registres* : pour chaque port d'entrée de chaque module, un module *delay* est inséré en amont du port lorsque la correction déterminée est non nulle. Ce module correspond à une chaîne de registres dont le nombre de registres est paramétrable. La valeur de la correction est reportée dans ce paramètre.

La figure 30 donne un exemple de structure COLIF avant et après correction de retard.

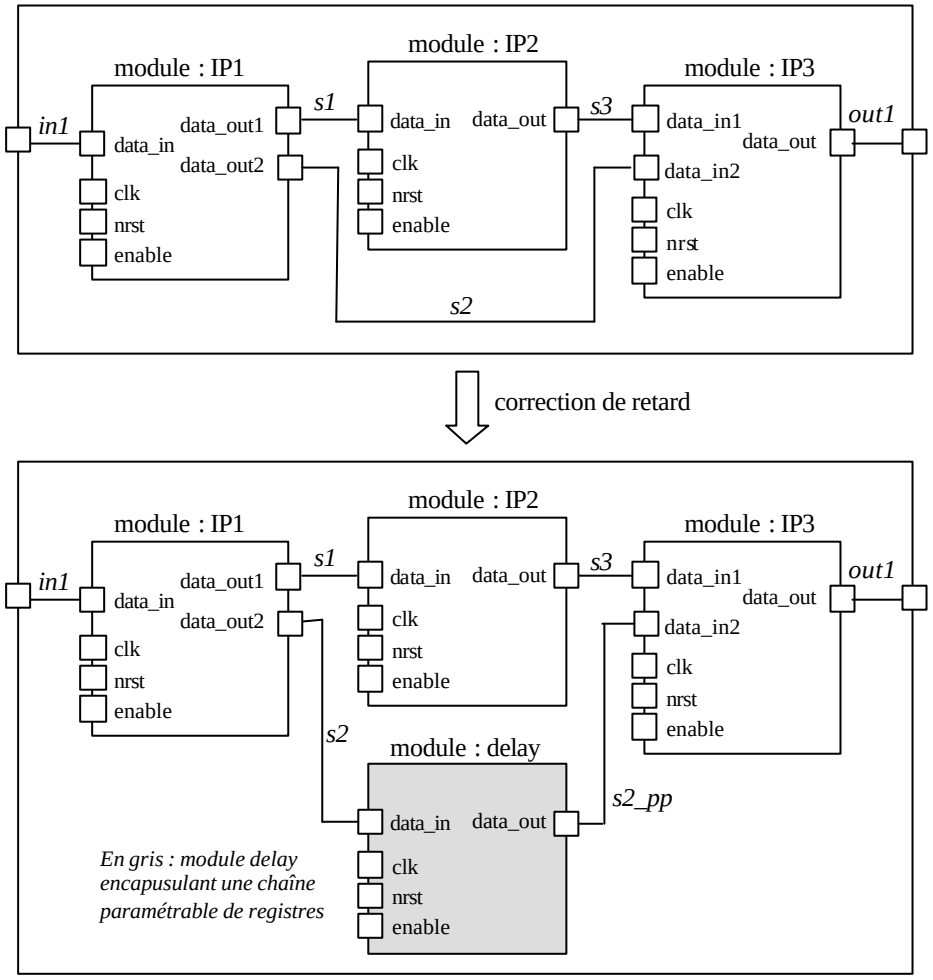


Figure 30. Exemple de structure COLIF avant et après correction de retard (FSM non dessinée pour soucis de clarté)

V.4.7. L'interface graphique d'utilisation (GUI)

L'interface graphique d'utilisation (ou *GUI*) (figure 31) a pour rôle de lier les différents outils sous un environnement commun, homogène et convivial. Elle a été développée à l'aide

de GUIDE [GUI 03]. Etant donné que la partie « gestion de la librairie » n'est pas encore développée, la GUI regroupe actuellement les outils relatifs à la partie « conception par assemblage ». Elle se compose de deux parties : *le menu* (haut de la figure 31) et *l'espace de travail* (bas de la figure 31).

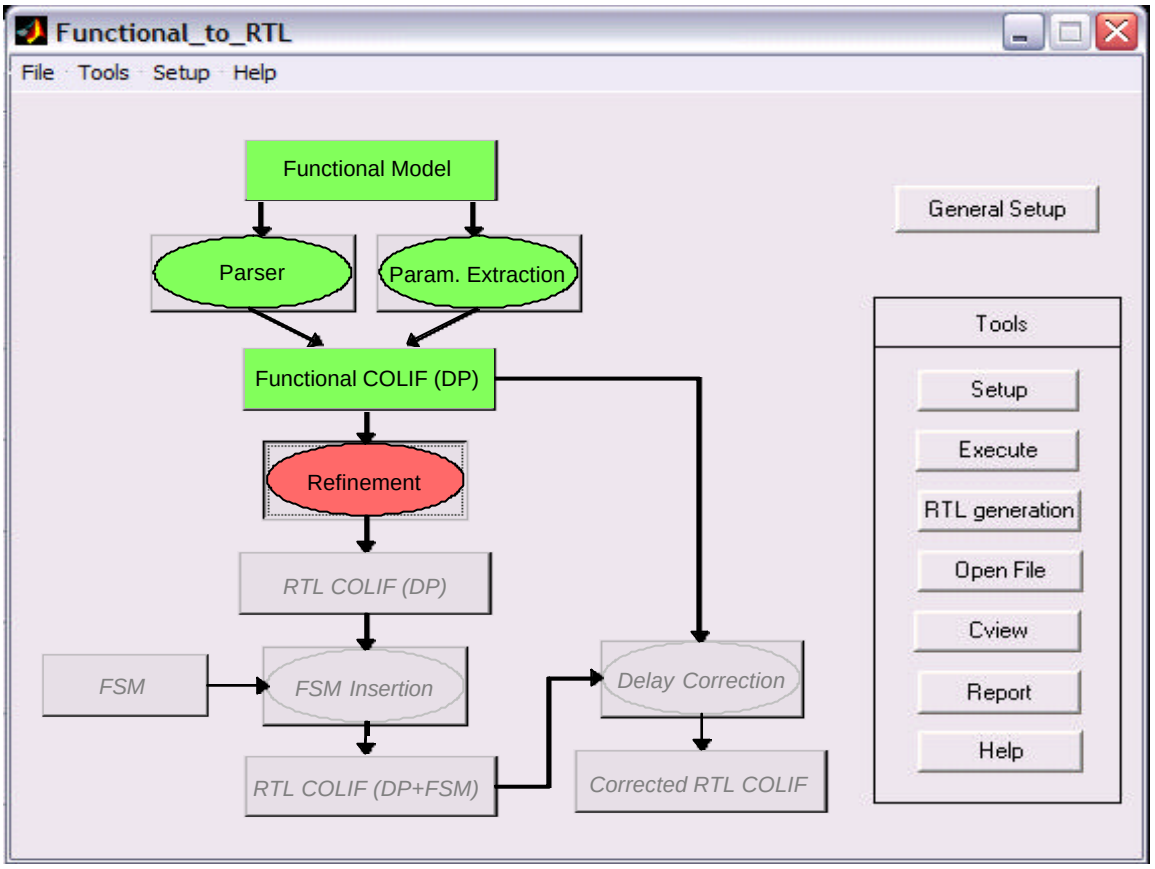


Figure 31. L'interface graphique d'utilisation

Le menu de la GUI inclue les fonctions standards d'une GUI :

– *File* : ce menu contient un sous-menu permettant la création d'un nouveau projet, la récupération d'un ancien projet, la sauvegarde du projet courant, l'envoie des données de la GUI dans l'espace de travail de Matlab, la récupération de ces données depuis l'espace de travail de Matlab et la fermeture de la GUI. Tous les outils de Matlab (e.g. Simulink) interagissent avec l'espace de travail de Matlab via des API, nous avons

choisi de conserver cette interopérabilité pour permettre à l'utilisateur de travailler soit à partir de la GUI ou soit de travailler depuis Matlab via des commandes en ligne.

- *Tools* : ce menu contient un sous-menu appelant les outils et visualisations disponibles. Il réalise les mêmes tâches que les boutons « tools » de l'espace de travail (description ci-dessous).
- *Setup* : ce menu appelle une fenêtre permettant de configurer la GUI. Cette fenêtre contient entre autre la saisie du chemin de la librairie, le nom du projet et la saisie des variables d'environnement. Ce menu est analogue au bouton « general configuration » de l'espace de travail. A terme, ce menu devrait inclure le choix de différents éditeurs de texte ou d'outils de visualisation suivant les préférences de l'utilisateur.
- *Help* : aide à l'utilisation de la GUI.

l'espace de travail (bas de la figure 31) a pour rôle de permettre l'exécution des différents outils, le contrôle des modèles résultants de chaque outil et la visualisation de l'avancement du travail dans le flot. L'espace de travail se compose de deux parties. La partie gauche schématise le synoptique du flot. Elle a pour tâche de schématiser l'interaction entre les outils ainsi que la progression du flot. Un code couleur permet de visualiser cette progression :

- la couleur verte signifie que l'étape du flot a déjà été réalisée avec succès.
- la couleur rouge montre l'étape en cours.
- la couleur grise signifie que l'étape n'a pas encore été réalisée.
- le texte de l'étape en italique signifie que l'étape ne peut encore être réalisée (e.g. la correction de retard n'est réalisable que lorsque le raffinement et l'insertion de la FSM ont déjà été réalisés). Chaque étape du flot correspond à un bouton de la GUI activable par la souris.

La partie droite de l'espace de travail correspond aux différentes actions (figure 31) pouvant être entreprises sur l'étape en cours (en rouge sur la GUI). Ces actions sont :

– *Setup* (figure 32) : appel d'une fenêtre pour la saisie des options de l'outil sélectionné (e.g. nom du fichier de sortie d'un outil).

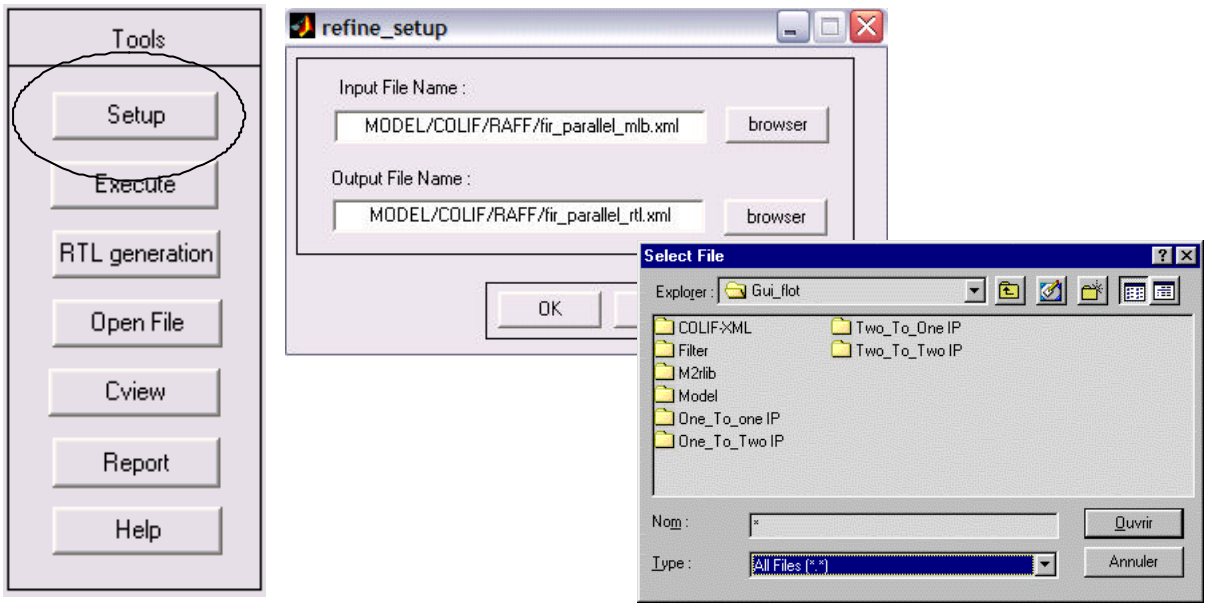


Figure 32. Exemple d'utilisation du bouton « setup »

– *Execute* (figure 33) : exécution de l'outil.

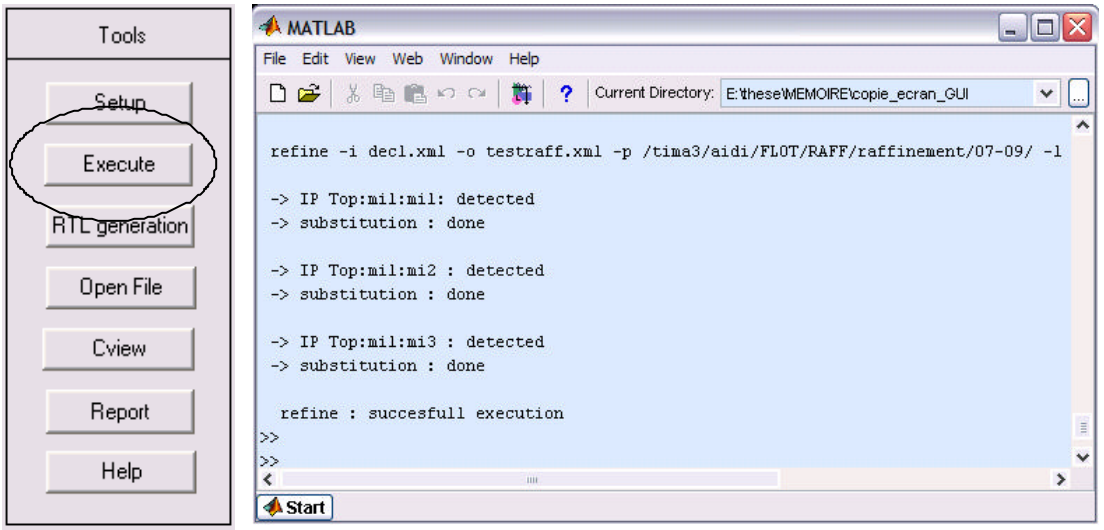


Figure 33. Exemple d'utilisation du bouton « execute »

– *RTL generation* (figure 34) : génération de code RTL. A toute étape du flot, il est possible de générer le code RTL correspondant. Avant la fin du flot, le code RTL généré est partiel et l'utilisateur est alors libre de le compléter manuellement.

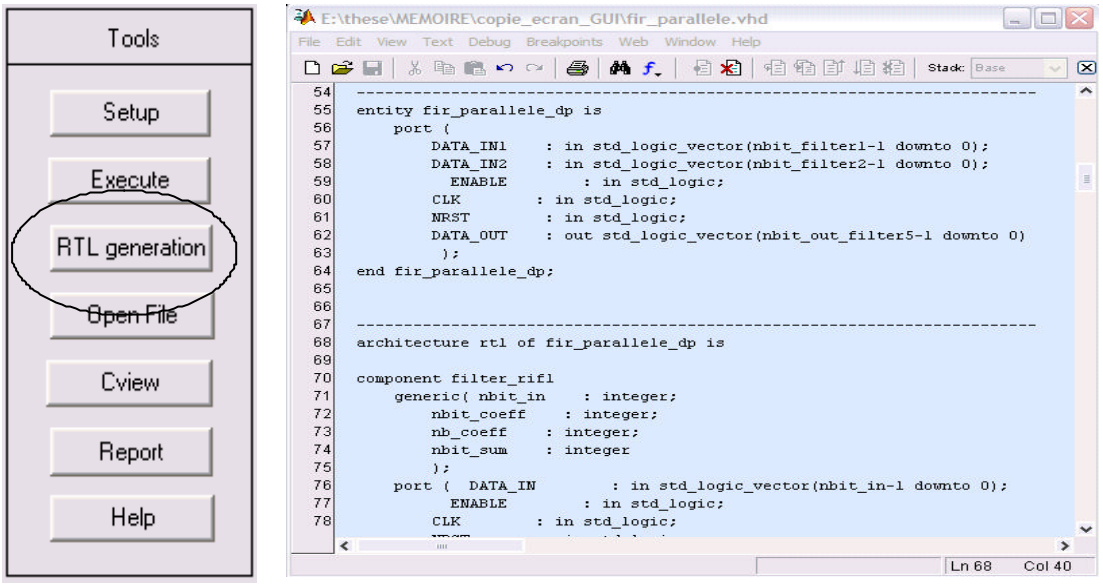


Figure 34. Exemple d'utilisation du bouton « RTL generation »

– *Open file* (figure 35) : ouverture des fichiers XML générés par l'outil.

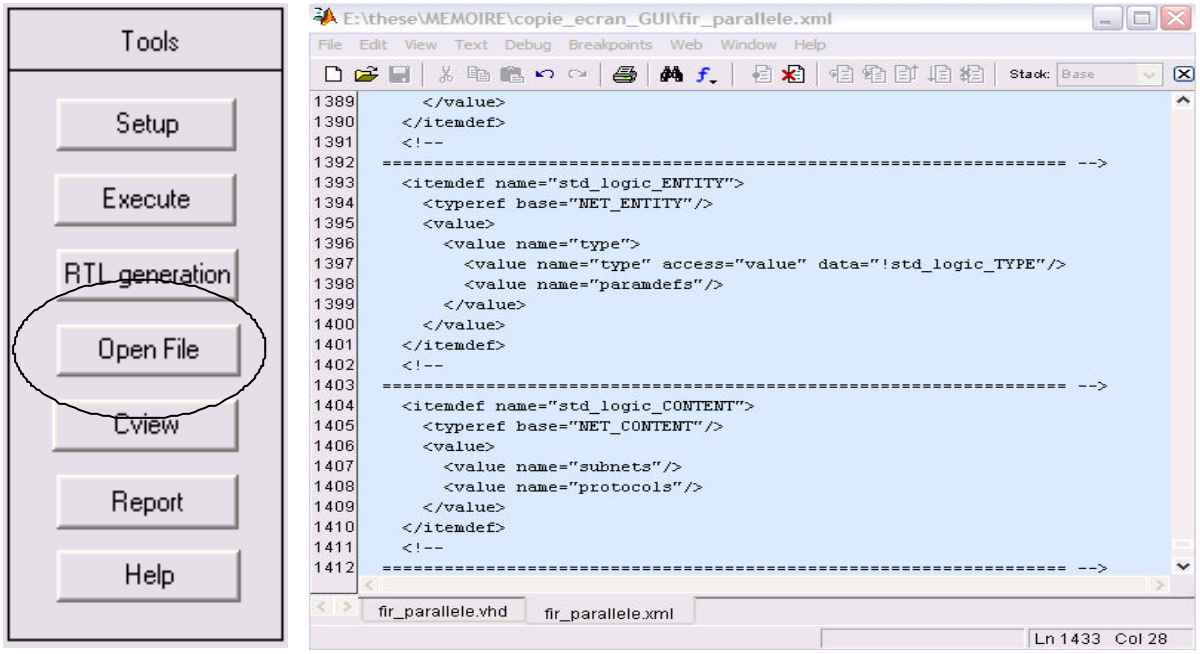


Figure 35. Exemple d'utilisation du bouton « open file »

– *Cview* (figure 36) : visualisation des structures COLIF par cview.

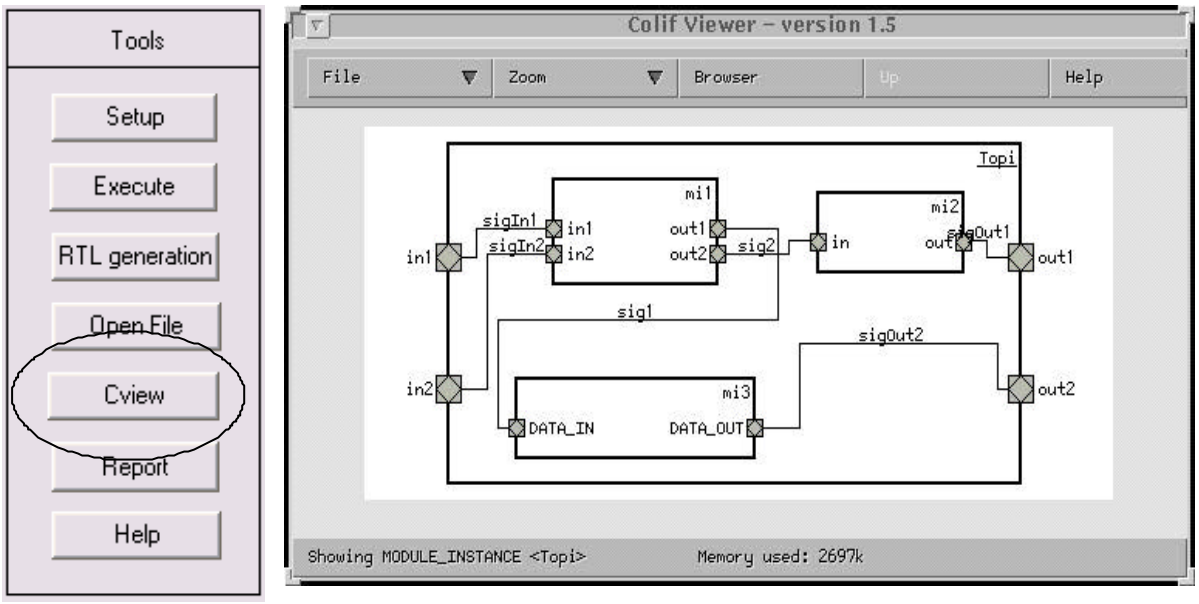


Figure 36. Exemple d'utilisation du bouton « cview »

– *Report* (figure 37) : ouverture des rapports d'exécution de l'outil.

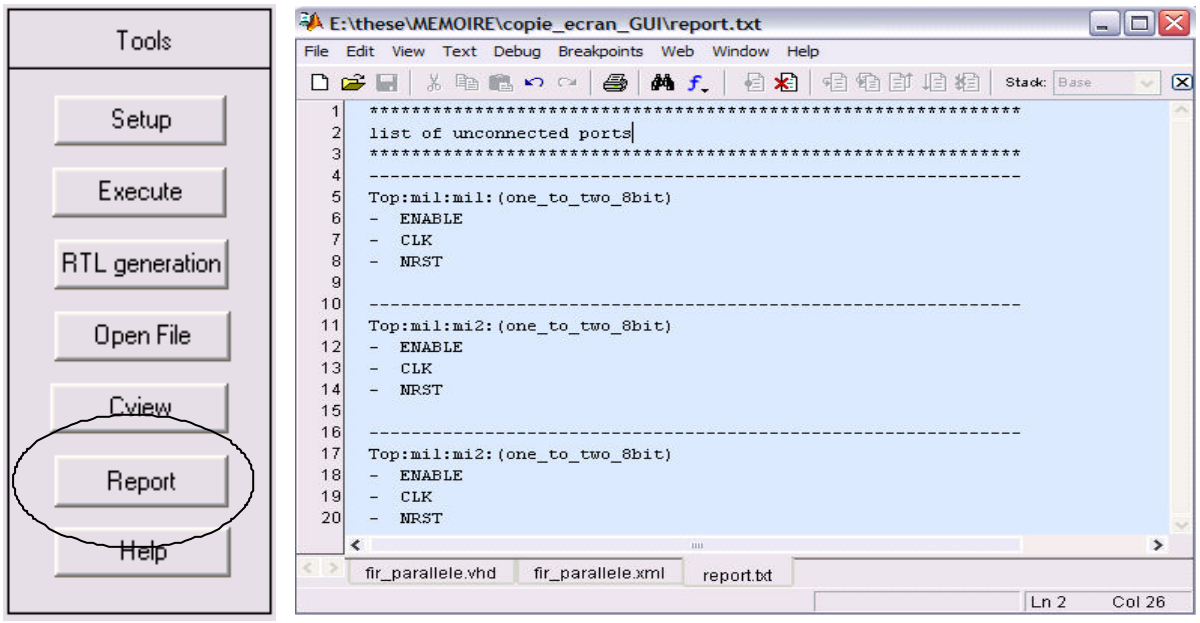


Figure 37. Exemple d'utilisation du bouton « report »

– *Help* (figure 38) : aide à l'utilisation de l'outil.

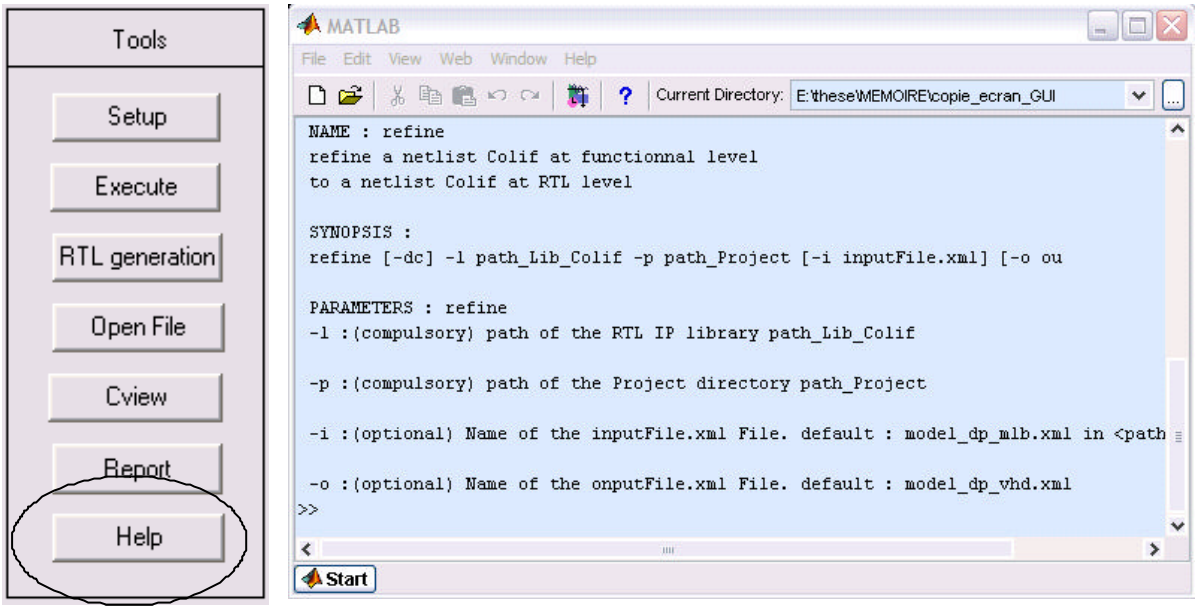


Figure 38. Exemple d'utilisation du bouton « help »

Ces actions sont contextuelles et dépendes de l'étape sélectionnée. Par exemple, la fenêtre de configuration est différente suivant l'étape en cours.

V.5. Conclusion

Dans ce chapitre, nous avons présenté l'implémentation de la méthodologie proposée (chapitre III) en un flot de conception semi-automatique complet. Ce flot utilise COLIF comme forme intermédiaire pour représenter l'application à chaque étape de la méthodologie. Chaque outil développé a été décrit. Nous avons également présenté l'interface graphique permettant de lier les différents outils sous un environnement homogène et convivial.

Actuellement, le développement n'est pas finalisé. Les principales étapes de l'assemblage automatique (i.e. parseur de netlist Matlab, raffinement et correction de retard) sont développées. Les étapes correspondant à la gestion de la librairie sont en cours de développement.

Chapitre VI. Conclusion

Sommaire :

VI.1. Récapitulatif des motivations, objectifs et contributions généraux de ce travail

L'incessante augmentation de la complexité des algorithmes DSP ainsi que du débit des données requièrent l'utilisation de macro-cellules ASIC (*Application Specific Integrated Circuit*) dédiées au traitement du signal. Ces macro-cellules deviennent de plus en plus coûteuses en temps et effort de conception pour : modéliser et valider la fonctionnalité de l'application ; explorer conjointement l'algorithme et l'architecture ; préserver la qualité de l'architecture ; concevoir des applications dérivées.

L'objectif général de cette thèse est de proposer une méthodologie et un flot de conception pour les macro-cellules ASIC dédiées au traitement numérique du signal. Cette méthodologie et ce flot doivent permettre : une modélisation et une validation rapide de la fonctionnalité de l'application traitée ; une exploration conjointe de l'algorithme et de l'architecture ; la préservation de la qualité de l'architecture comme si elle avait été manuellement faite par un concepteur matériel ; une conception rapide d'applications dérivées.

La contribution générale de cette thèse est la proposition d'une méthodologie de conception pour macro-cellule ASIC-DSP. Cette méthodologie part d'une description fonctionnelle. Elle est basée sur :

- l'utilisation de blocs de base (DSP-IP) de traitement du signal, préconçus, paramétrables, et décrits à la fois au niveau fonctionnel et RTL.
- l'assemblage automatique des IP-RTL suivant l'assemblage définit par le modèle fonctionnel d'entrée.

Les bénéfices de notre méthodologie sont les suivants :

- l'utilisation d'une description fonctionnelle et de DSP-IP fonctionnels préconçus permet une modélisation et une validation fonctionnelle rapide et efficace en adoptant une approche « diviser pour mieux régner ».
- l'utilisation de DSP-IP paramétrables permet une exploration conjointe algorithme/architecture car ces paramètres influencent à la fois la qualité du signal que la qualité de l'architecture.
- l'assemblage automatique et l'utilisation de DSP-IP RTL préconçus permet de préserver la qualité de l'architecture comme si elle avait été manuellement faite par un concepteur. En effet, le flot automatique ne prend pas de décision et reproduit au niveau RTL, les choix fait par le concepteur au niveau fonctionnel.
- l'utilisation de DSP-IP paramétrables permet également d'accroître sensiblement la conception d'applications dérivées. En effet, à partir du modèle fonctionnel, il suffit juste de modifier les valeurs des paramètres pour s'adapter aux nouvelles contraintes de fonctionnalités ou d'implémentations. Ces modifications sont alors automatiquement reportées au niveau RTL.

VI.2. Revue des travaux présentés

VI.2.1. Etude comparative des outils commerciaux actuels de synthèse d'architecture

Dans le **chapitre 2**, nous avons réalisé une étude comparative de trois outils de synthèse d'architecture. Le but de cette étude a été de comprendre pourquoi la synthèse d'architecture n'a pas encore été acceptée dans le monde industriel. Ces outils ont été évalués par expérimentation sur deux exemples industriels courants : un filtre interpolateur et un filtre SRC. Les résultats obtenus par synthèse d'architecture sont comparés à deux réalisations RTL conçues manuellement. Les résultats ont été les suivants : la synthèse d'architecture supporte

une description d'entrée bien plus efficace à modéliser qu'une description RTL conventionnelle ; mais, la qualité de l'architecture produite est bien inférieure à la qualité d'une architecture produite manuellement. Ce problème vient du fait que la synthèse d'architecture actuelle a du mal à tirer partie de la puissance de la synthèse RTL. Le second problème majeur est que la synthèse d'architecture modifie le comportement des entrées/sorties vis-à-vis de la description d'entrée. Ce constat complexifie à la fois la tâche de vérification de l'architecture RTL et l'intégration de cette architecture avec d'autres composants.

VI.2.2. Proposition d'une méthodologie et implémentation d'un flot semi-automatique de conception et de validation pour macro-cellules ASIC DSP

Dans le **chapitre 3**, nous avons présenté une méthodologie et un flot semi-automatique pour la conception et la validation de macro-cellules ASIC dédiées au traitement du signal. Cette méthodologie débute par une spécification de haut niveau et se basent sur l'assemblage automatique d'IP préconçus et paramétrables. Cela permet une modélisation et une validation fonctionnelle rapide, une exploration conjointe algorithme/architecture, la préservation de la qualité de l'architecture comme si fait main, et une conception rapide d'applications dérivées. Nous avons illustré l'efficacité de l'approche par l'expérimentation sur un exemple industriel : une chaîne de modulation numérique. Des outils (**chapitre 5**) ont été développés pour transformer cette méthodologie en un flot semi-automatique.

VI.2.3. Méthode de correction de retard lors du passage d'un assemblage d'IP fonctionnels vers un assemblage d'IP RTL

Dans le **chapitre 4**, nous avons vu que le passage d'un assemblage de DSP-IP fonctionnels vers un assemblage de DSP-IP RTL pouvait produire une différence de comportement entre les deux assemblages. Les causes de ce problème sont des retards présents dans les DSP-IP RTL et absents des DSP-IP fonctionnels. Un moyen de compenser ces retards additifs est

d'ajouter des registres dans le chemin de données entre les DSP-IP RTL. Nous avons défini une méthode automatique (appelée *méthode de correction de retard*) permettant la détermination du nombre et la localisation des registres de correction dans le cas des applications sans boucle. Le problème de différence de comportement a d'abord été formalisé comme un problème de théorie des graphes. A partir des graphes des modèles fonctionnel et RTL, deux algorithmes ont été développés. Le premier fournit une solution optimale en latence alors que le deuxième obtient une solution optimale en nombre de registres insérés. La validité de la méthode ainsi que l'optimalité des corrections ont été démontrées mathématiquement. La correction de retard par insertion de registres a été comparée expérimentalement à une correction de retard par modification de la FSM globale, autre moyen de correction possible. Nous avons constaté qu'apporter une correction par insertion de registres demande un coût de modification du modèle RTL initial beaucoup moins important qu'une correction par modification de la FSM. En contrepartie, le surcoût en surface ajoutée par les registres est plus grand que le surcoût d'une correction par modification de la FSM. Pour les applications contenant des boucles, nous montrons que l'insertion de registres ne peut être employée et discutons les alternatives d'implémentation : nous constatons alors que les alternatives donnent des résultats insuffisants pour être applicable. Le problème de différence de comportement est reformulé pour le cas des boucles. Un premier algorithme est fournit pour donner une solution permettant d'équilibrer le graphe différentiel. L'optimisation de cette solution dépend du type d'implémentation de la correction.

VI.3. Perspectives

Les perspectives à court terme de ce travail sont : de finaliser dans un premier temps, le travail de développement des outils composant le flot ; puis, dans un second temps, d'appliquer ce flot sur un exemple industriel concret. Actuellement, seule la méthodologie a

été validée par expérimentation. Nous cherchons maintenant à tester l'efficacité de nos outils sur une application concrète (e.g. codeur DIVX [DER 03]).

A moyen terme, une perspective est d'intégrer notre flot de conception dans un flot complet pour circuit intégré monopuce tel que le flot ROSES du groupe SLS. Un premier travail a été fait dans ce sens puisque notre flot utilise COLIF, langage de description développé par le groupe SLS, permettant de représenter un système complexe durant toutes les étapes de raffinement.

A long terme, la correction de retard offre des opportunités de recherche intéressantes. Actuellement, les travaux sont achevés pour le cas des assemblages sans boucle et ne contenant pas d'IP dépendant du temps. Il reste alors à traiter le cas des assemblages contenant des boucles ou contenant des IP dépendants du temps. Des travaux prospectifs ont déjà été faits dans ce sens. Dans le cas des IP dépendants du temps, le cas de la chaîne de modulation numérique à montrer qu'il était possible d'apporter une correction au niveau fonctionnel. Mais, un cas ne faisant pas une généralité, ce résultat n'a pas encore été démontré. Deux voies peuvent être explorées : la première consiste à apporter une correction en modifiant la FSM initiale (en retardant le signal de contrôle sur l'IP problématique) ; la seconde consiste à insérer des registres à l'intérieur de l'IP (méthode intrusive). Le choix de la solution apportée dépendra des preuves mathématiques apportées. D'autre part, nous avons vu (§IV.3) que le cas des assemblages avec boucle ne pouvait être corrigé par insertion de registres : d'autres voies ont été étudiées (retiming architectural) ou modification de la FSM. Ce point reste actuellement en suspens. Donc, une perspective au cas générale de la correction de retard serait la résolution du problème de différence de comportement par modification de la FSM. La difficulté est ici l'adaptation du formalisme employée à présent car orienté vers le flot de donnée plus que vers le contrôle

Le cas des applications multifréquences n'a pas été ciblé par la correction de retard. Ce cas peut être l'œuvre d'un sujet de recherche à part entière. En effet, la formalisation actuelle du problème de correction de retard portée au cas des applications multifréquences rentre dans le cadre de la résolution des équations diophantiennes (i.e. algèbre en nombre entier). Il est possible de formuler ce problème comme un problème de programmation linéaire en nombre entier. Néanmoins, si les solveurs sont actuellement capables de résoudre tout problème de programmation linéaire, l'analyse du problème dans le cadre plus particulier du problème de différence de comportement pourrait apporter des solutions plus efficaces en temps de résolution que celles fournit par les solveurs conçus dans un cadre général.

Annexe A. Démonstration des théorèmes

Résumé : dans cette annexe, nous démontrons les théorèmes et résultats donnés dans la section IV.

Sommaire :

A.1. Terminologie et notations relatives aux démonstrations

La terminologie et les notations relatives aux démonstrations sont les suivantes :

- un *graphe* est noté $G = (V, E)$ où V est l'ensemble de ses sommets et E est l'ensemble de ses arcs. Les graphes sur lesquels les démonstrations sont faites sont des *graphes orientés*.
- un *sommet* de G est noté par la lettre v.
- un *arc* d'extrémité initiale v_1 et d'extrémité finale v_2 est noté $e = [v_1, v_2]$.
- le *poids* de l'arc e est noté $d(e)$. Ici, un poids représente un retard ou une différence de retard.
- la *correction de retard* apportée à l'arc e est notée $c(e)$. L'ensemble des corrections portées sur tous les arcs est noté C.
- un *chemin* passant par les sommets $v_1, v_2,\ldots, v_n$ est noté $P=[v_1, v_2,\ldots, v_n]$. Le *plus long chemin* (resp. *court*) de G est noté P_{max}.
- rappel (définition 1) : un graphe est dit *équilibré* si et seulement si, pour tout sommet, la somme des retards est indépendante du chemin depuis une entrée jusqu'au sommet considéré.
- la *longueur* du chemin P est notée $L(P)$ et correspond à la somme des poids sur les arcs du chemin. Ici, cette longueur correspond à la somme des retards sur ce chemin. A noter que si le graphe G est équilibré, L est indépendant du chemin P mais uniquement dépendant du sommet terminal v_n de P. Dans ce cas, la longueur du chemin P est notée $L(v_n)$.
- $\Gamma^-(v)$ désigne l'ensemble des sommets prédécesseurs de v. De même, $\Gamma^+(v)$ désigne l'ensemble des sommets successeurs à v.
- l'extrémité initiale d'un arc e est notée I(e). Son extrémité finale est notée T(e).

- le *nombre d'éléments* d'un ensemble *å* est noté *card(å).*
- la *composition* de deux fonctions *g* et *h* sera notée $g \circ h$.
- *f* désigne *la fonction arithmétique* réalisée par un sommet. Comme expliqué dans le chapitre IV, cette fonction a la propriété d'être *une fonction indépendante du temps*.
- D_k est la *fonction retard* de *k* échantillons. Dans les démonstrations qui suivent, deux de ses propriétés seront exploitées : l'*additivité* ($D_{k1} \circ D_{k2} = D_{k1+k2}$) et la *commutation* avec une fonction invariante dans le temps ($D_k \circ f = f \circ D_k$).
- *min(g(x)/x Î A)* est la *minimum* des valeurs *g(x)* pour *x* appartenant à l'ensemble *A*. De même, *max(g(x)/x Î A)* est son *maximum*.
- la description des arcs d'un cycle $\Gamma = [e_1, e_2, \ldots, e_n]$ dans l'ordre de la séquence définit un *sens de parcours* sur Γ. Un arc du cycle est *orienté dans le sens du parcours* (resp. en *sens inverse du parcours*) s'il est adjacent à l'arc qui le précède dans la séquence par son extrémité initiale (resp. terminale) et à l'arc qui le suit par son extrémité terminale (resp. initiale). A un cycle Γ, on associe le vecteur g dit *vecteur représentatif* de Γ par $g(e) = 1$ lorsque l'arc *e* de Γ est orienté dans le sens du parcours et par $g(e) = -1$ lorsque l'arc *e* de Γ est orienté dans le sens inverse. On appelle *longueur algébrique* sur le chemin Γ, la somme des retards *d(e)* sur Γ compté suivant le sens du parcours : $\bar{L}(\Gamma) = \sum_{e \in \Gamma} g(e) \cdot d(e)$

La méthode de correction de retard manipule cinq graphes : le graphe fonctionnel d'évolution, le graphe RTL d'évolution, le graphe différentiel d'évolution, le graphe différentiel corrigé et le graphe RTL corrigé. On particularisera les notations ci-dessus à chacun des graphes en ajoutant la lettre :

- « F » pour le graphe fonctionnel.
- « R » pour le graphe RTL.

- « D » pour le graphe différentiel.
- « DC » pour le graphe différentiel corrigé.
- « RC » pour le graphe RTL corrigé
- S'il n'y a pas ambiguïté, ces extensions de nom seront omises.

A.2. Démonstration du théorème de conservation du comportement

Théorème 1. *Condition suffisante de conservation du comportement*

Enoncé : si un graphe différentiel d'évolution est équilibré alors le modèle fonctionnel et le modèle RTL ont le même comportement.

Démonstration :

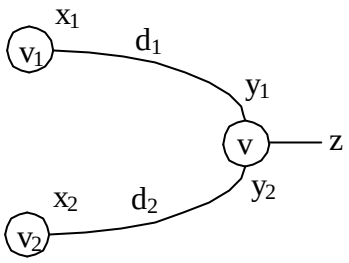
Soit G_D un graphe différentiel d'évolution équilibré. Montrer que le modèle fonctionnel et RTL ont un même comportement revient à montrer que localement, pour tout sommet v, le signal de sortie de v sur le graphe fonctionnel a le même comportement que le signal de sortie de v sur le graphe RTL.

Soit un sommet v de G_D. Pour faciliter la compréhension de la démonstration, on suppose qu'uniquement deux arcs e_1 et e_2 convergent sur ce sommet. La généralisation à plus de deux arcs peut être faite par récurrence en adoptant la même démarche. Sur l'arc e_1, on note (figure 39) :

- x_1, le signal de sortie du sommet v_1.
- y_1, le signal d'entrée du sommet v.
- d_1 le retard sur l'arc e_1.

On adoptera les mêmes notations pour l'arc e_2 en changeant l'indice «1» par «2». On note f, la fonction réalisée par le sommet v et z, le signal de sortie de v. Puisque le comportement

du modèle fonctionnel et RTL vont être comparés, on ajoutera les extensions de nom «R» et «F» pour distinguer ces deux modèles.



x_1, x_2, y_1, y_2 : signal au point considéré

d_1, d_2 : retard de l'arc

Figure 39. Notations de la démonstration du théorème de conservation du comportement

On suppose que le signal de sortie x_{F1} et x_{R1} ont le même comportement. De même, on suppose que x_{F2} et x_{R2} ont le même comportement. Si x_{F1} et x_{R1} ont le même comportement, alors ces signaux sont égaux à un retard prés : $x_{R1} = D_{k1}(x_{F1}) \text{ avec } k_1 = L_R(v_1) - L_F(v_1)$

Il est de même pour v_2 : $x_{R1} = D_{k1}(x_{F1}) \text{ avec } k_1 = L_R(v_1) - L_F(v_1)$

On exprime la sortie z_R en fonction de x_{F1} et x_{F2} :

$$y_{R1} = D_{dR1}(x_{R1}) = D_{dR1} \circ D_{k1}(x_{F1}) = D_{dR1+k1}(x_{F1})$$

$$y_{R2} = D_{dR2}(x_{R2}) = D_{dR2} \circ D_{k2}(x_{F2}) = D_{dR2+k2}(x_{F2})$$

$$z_R = f(y_{R1}, y_{R2}) = f(D_{dR1+k1}(x_{F1}), D_{dR2+k2}(x_{F2})) \quad [1]$$

De la même manière, on exprime la sortie z_F en fonction de x_{F1} et x_{F2} :

$$y_{F1} = D_{dF1}(x_{F1})$$

$$y_{F2} = D_{dF2}(x_{F2})$$

$$z_F = f(y_{F1}, y_{F2}) = f(D_{dF1}(x_{F1}), D_{dF2}(x_{F2})) \quad [2]$$

D'autre part, $k_1 = L_R(v_1) - L_F(v_1)$ [3]

Or, $L_F(v) = L_F(v_1) + d_{F1}$

Soit encore, $L_F(v_1) = L_F(v) - d_{F1}$ [4]

De même, $L_R(v_1) = L_R(v) - d_{R1}$ [5]

On remplace les expressions [4] et [5] dans [3] : $k_1 = L_R(v) - L_F(v) - d_{R1} + d_{F1}$ [6]

De même, $k_2 = L_R(v) - L_F(v) - d_{R2} + d_{F2}$ [7]

Les expressions [6] et [7] de k_1 et k_2 sont substituées dans l'expression [1] de z_R :

$$z_R = f(D_{dR1+LR(v)-LF(v)-dR1+dF1}(x_{F1}), D_{dR2+LR(v)-LF(v)-dR2+dF2}(x_{F2}))$$

$$z_R = f(D_{LR(v)-LF(v)+dF1}(x_{F1}), D_{LR(v)-LF(v)+dF2}(x_{F2}))$$

$$z_R = f(D_{LR(v)-LF(v)} \circ D_{dF1}(x_{F1}), D_{LR(v)-LF(v)} \circ D_{dF2}(x_{F2}))$$

Or, f est invariant dans le temps. Donc, f et D peuvent commuter. D'où,

$$z_R = D_{LR(v)-LF(v)} \circ f(D_{dF1}(x_{F1}), D_{dF2}(x_{F2})) \quad [8]$$

En comparant l'expression [8] et [2] de z_R et de z_F, on obtient : $z_R = D_{LR(v)-LF(v)}(z_F)$

z_R est donc une version retardée de z_F. D'où la conservation du comportement au niveau du sommet v. Si le graphe fonctionnel et RTL ont le même comportement pour chacun de leur sommet, ils ont globalement le même comportement.

A.3. Démonstration de l'obtention d'une solution par l'algorithme de correction pour l'optimisation en latence

Nous montrons dans cette section que l'algorithme de correction de retard pour l'optimisation en latence (algorithme 1 rappelé ci-dessous) converge vers une solution équilibrant le graphe différentiel d'évolution. Cette démonstration nécessite de démontrer trois points :

(1) à chaque itération, il est toujours possible de trouver un nouveau sommet non traité par l'algorithme dont tous les sommets prédécesseurs sont déjà traités. Dans le cas contraire, la boucle de l'algorithme ne peut s'achever.

(2) à chaque itération, la correction apportée est une valeur entière positive (puisque les registres insérés ne peuvent que retarder le signal et non l'avancer).

(3) le graphe final est un graphe équilibré (pour pouvoir satisfaire la condition du théorème de conservation du comportement).

1 : $V_1 = \{v \in V / \Gamma_G^- = \varnothing\}$ et $\forall v \in V_1, L(v) = 0$
2 : **Tant que** V_1 V **faire :**
3 : Soit un sommet v tel que $v \notin V_1$ et $\Gamma_G^-(v) \subset V_1$
4 : Soit E^- l'ensemble des arcs arrivants en v
5 : $\forall e \in E^-, p(e) = L(I(e)) + d(e)$
6 : $L(v) = \max(p(e) / e \in E^-)$
7 : $\forall e \in E^-, c(e) = L(v) - p(e)$
8 : Incrémenter V_1 en faisant $V_1 = V_1 \cup \{v\}$
9 : **Fin tant que**

Algorithme 1. Algorithme de correction de retard pour l'optimisation en latence

Démonstration de (1) : existence d'un nouveau sommet

Soit G = (V,E) le graphe différentiel traité par l'algorithme. A la n^{ième} itération de l'algorithme, on note V_1 l'ensemble des sommets de V déjà traités par l'algorithme et V_2 l'ensemble des sommets de V qui n'ont pas encore été traités. L'existence d'un sommet v de V_2 ayant tous ses prédécesseurs dans V_1 sera démontrée par l'absurde. On suppose alors l'hpothèse (H) :

(H) - tout sommet v de V_2 a au moins un prédécesseur qui n'est pas dans V_1 (donc, qui est dans V_2 puisque que $V = V_1 \cup V_2$).

Considérons la suite récurrente (v_n) composée d'éléments de V_2 : v_0 est un sommet quelconque de V_2 et pour tout entier n, v_{n+1} est tel que il existe un arc reliant v_n à v_{n+1}. D'après l'hypothèse précédente, pour tout n, un nouvel élément v_{n+1} de la suite peut être construit puisque tout sommet de V_2 a au moins un prédécesseur dans V_2. Cette suite est tout simplement un chemin construit de proche en proche dans V_2.

Le graphe G est acyclique, donc tous les éléments de la suite sont différents. A chaque étape de la construction de la suite, un nouveau sommet est ajouté. Donc, pour tout entier n, $card(\{v_0,\ldots,v_n\}) = n$. Donc, $card(\{v_0,\ldots,v_n\})$ est une suite strictement croissante non bornée [9].

Or, pour tout entier n, l'ensemble $\{v_0,\ldots,v_n\}$ est inclu dans V_2. Ce qui implique que la suite $card(\{v_0,\ldots,v_n\})$ est bornée par $card(V_2)$. Cela est contradictoire avec [9] : l'hypothèse (H) est absurde. D'où l'existence d'un sommet v de V_2 ayant tous ses prédécesseurs dans V_1.

Démonstration de (2) : la correction est une valeur positive

Pour chaque sommet v, la correction apportée sur tout arc e arrivant en v est égale à $c(e) = \max(\pi(e_i)) - \pi(e)$ (où π est une variable intermédiaire introduite par l'algorithme). Ce maximum est réalisé sur l'ensemble E^- des arcs e_i arrivants en v. Puisque e est inclus dans E^-, on a donc $\max(\pi(e_i)) \geq \pi(e)$. Donc, $c(e) \quad 0$.

Démonstration de (3) : le graphe final est un graphe équilibré

La démonstration est faite par récurrence sur le nombre d'itération de l'algorithme. On note $G_1^{(n)}$, le sous-graphe de G composé des sommets déjà traités par l'algorithme à la $n^{ième}$ itération.

Pour n=0, $G_1^{(0)}$ n'est composé que des sommets sans arc incident.

Donc, il n'existe pas de chemin arrivant jusqu'à ces sommets. Tous les chemins arrivants à ces sommets sont alors égaux puisqu'il n'y a pas de chemin. Le graphe $G_1^{(0)}$ est donc un graphe équilibré.

Supposons qu'à la n^{ième} itération, $G_1^{(n)}$ soit un graphe équilibré et montrons que l'algorithme équilibre $G_1^{(n+1)}$.

Soit *v*, le nouveau sommet introduit par l'algorithme à l'itération n+1. Les corrections apportées par l'algorithme sont réalisées sur les arcs incidents à *v*. Donc, cette itération conserve $G_1^{(n)}$ comme graphe équilibré. Reste à montrer que, pour le nouveau sommet *v*, la somme des retards est indépendante du chemin depuis une entrée jusqu'à *v*. Pour cela, on considère un chemin *P* quelconque depuis une entrée jusqu'à *v*. On note $\tilde{v}$ le sommet prédécesseur de *v* sur le chemin *P* et *e* l'arc partant de $\tilde{v}$ et arrivant en *v* (i.e. $e = [\tilde{v}, v]$).

Par définition de *v*, tous les sommets prédécesseurs de *v* sont des sommets de $G_1^{(n)}$. Donc, $\tilde{v}$ est un sommet de $G_1^{(n)}$. Or, $G_1^{(n)}$ est un graphe équilibré par hypothèse de récurrence. Donc, la somme des retards jusqu'à $\tilde{v}$ est indépendante du chemin et vaut $L(\tilde{v})$.

Après correction par l'algorithme, la somme des retards sur *P* vaut :

$$L(P) = L(\tilde{v}) + d(e) + c(e) \qquad [10]$$

Cela signifie que le retard cumulé sur le chemin *P* est égal au retard jusqu'en $\tilde{v}$ additionné du retard induit par le dernier arc *e*. Ce dernier retard se compose du retard initial *d(e)* de l'arc et d'une éventuelle correction de retard *c(e)*.

Or, $c(e) = \max(p(\tilde{e}) / \tilde{e} \in E^-) - p(e)$ (où E^- est ensemble des arcs arrivants en *v*)

Où $p(e) = L(\tilde{v}) + d(e)$

Donc en remplaçant ces deux dernières expressions dans celle de [10], on obtient :

$$L(P) = L(\tilde{v}) + d(e) + \max(p(\tilde{e}) / \tilde{e} \in E^-) - L(\tilde{v}) - d(e)$$

Soit encore, $L(P) = \max(p(\tilde{e}) / \tilde{e} \in E^-)$

Or, $\tilde{e} = [I(\tilde{e}), T(\tilde{e})] = [I(\tilde{e}), v]$

Donc, $L(P) = \max(p([I(\tilde{e}), v] / \tilde{e} \in E^-))$ [11]

Le maximum d'un ensemble ne peut être fonction de la variable sur lequel il est fait. Donc, l'expression [11] montre que *L(P)* n'est fonction que du sommet *v*. Cette propriété est vraie pour tout chemin *P*, donc, la somme des retards est indépendante du chemin depuis une entrée jusqu'à *v*. D'où le résultat pour n+1.

Le graphe final obtenu par correction de retard est un graphe équilibré.

A.4. Démonstration de l'optimalité en latence de la solution obtenue

La solution obtenue par l'algorithme 1 est optimale en latence aussi bien pour la latence minimale que la latence maximale. La latence minimale d'un graphe *G* correspond au niveau du graphe fonctionnel, à la somme des retards sur le plus court chemin P_{Gmin} (il est donc noté $L_G(P_{Gmin})$). Sa latence maximale correspond à la somme des retards sur le plus long chemin P_{Gmax} (i.e. $L_G(P_{Gmax})$). Nous démontrons ici l'optimalité en latence. La démonstration fait intervenir deux lemmes préliminaires.

Lemme 1. *Conservation du plus long chemin après correction de retard*

Enoncé : après une correction de retard, le plus long chemin du graphe fonctionnel d'évolution est le même que celui du graphe RTL corrigé d'évolution.

Démonstration :

Soit un chemin *P* entre une entrée et une sortie v_o. La somme des retards sur *P* sur le graphe différentiel corrigé est égale à : $L_{DC}(P) = \sum_{\text{arc } e \text{ sur } P} d_{DC}(e)$

Or, par définition, $\forall e \in E_{DC}, d_{DC}(e) = d_{RC}(e) - d_F(e)$

Donc, $L_{DC}(P) = \sum_{\text{arc } e \text{ sur } P} (d_{RC}(e) - d_F(e))$

Soit encore, $L_{DC}(P) = \sum_{\text{arc } e \text{ sur } P} d_{RC}(e) - \sum_{\text{arc } e \text{ sur } P} d_F(e)$

D'où, $L_{DC}(P) = L_{RC}(P) - L_F(P)$ [12]

Après correction de retard, le graphe différentiel d'évolution est équilibré. Donc, tout chemin arrivant en v_o, est constant. Aussi bien le chemin P que le plus long chemin P_{Fmax} du graphe fonctionnel se terminent en v_o.

Donc, $L_{DC}(P) = L_{DC}(P_{F\max})$

En utilisant le résultat [12], on obtient : $L_{RC}(P) - L_F(P) = L_{RC}(P_{F\max}) - L_F(P_{F\max})$

$L_F(P) - L_F(P_{F\max}) = L_{RC}(P) - L_{RC}(P_{F\max})$ [13]

Puisque P_{Fmax} est le plus long chemin, on a donc $L_F(P_{F\max}) \geq L_F(P)$.

Soit encore, $L_F(P_{F\max}) - L_F(P) \geq 0$

Donc, [13] devient alors $L_{RC}(P_{F\max}) - L_{RC}(P) \geq 0$

Soit encore $L_{RC}(P_{F\max}) \geq L_{RC}(P)$ [14]

Puisque [14] est vrai quelque soit le chemin P, le chemin P_{Fmax} est le plus long chemin du graphe RTL corrigé.

Lemme 2. *Conservation de l'écart entre la latence maximale et minimale après correction de retard*

Enoncé : après correction de retard, l'écart entre la latence maximale et la latence minimale du graphe RTL corrigé est le même que celui du graphe fonctionnel d'évolution. I.e :

$$L_{RC}(P_{RC\max}) - L_{RC}(P_{RC\min}) = L_F(P_{F\max}) - L_F(P_{F\min})$$

Démonstration :

Posons $X = (L_{RC}(P_{RC\max}) - L_{RC}(P_{RC\min})) - (L_F(P_{F\max}) - L_F(P_{F\min}))$.

Le lemme sera alors prouvé en montrant que $X = 0$.

$$X = \sum_{e \in P_{RC\max}} d_{RC}(e) - \sum_{e \in P_{RC\min}} d_{RC}(e) - \sum_{e \in P_{RC\max}} d_F(e) + \sum_{e \in P_{RC\min}} d_F(e)$$

D'après le lemme 1, une correction de retard conserve les plus long et plus court chemins.

Donc, $P_{Fmin} = P_{RCmin}$ (noté simplement P_{min} par la suite) et $P_{Fmax} = P_{RCmax}$ (noté P_{max}).

Donc, $X = \sum_{e \in P_{\max}} d_{RC}(e) - \sum_{e \in P_{\min}} d_{RC}(e) - \sum_{e \in P_{\max}} d_F(e) + \sum_{e \in P_{\min}} d_F(e)$

$$X = \sum_{e \in P_{\max}} (d_{RC}(e) - d_F(e)) - \sum_{e \in P_{\min}} (d_{RC}(e) - d_F(e))$$

$$X = L_{DC}(P_{\max}) - L_{DC}(P_{\min})$$

Or, le graphe différentiel est équilibré. P_{min} et P_{max} se termine par le même sommet.

Donc, $L_{DC}(P_{\max}) = L_{DC}(P_{\min})$. D'où $X = 0$.

Théorème 3. *Optimalité de la solution vis-à-vis de la latence maximale*

Enoncé : l'algorithme de correction de retard fournit une solution optimale vis-à-vis de la latence maximale.

Démonstration :

Pour montrer que la solution *Calgo* obtenue par l'algorithme 1 est une solution optimale en latence maximale, on est amené à considérer l'ensemble des solutions permettant d'équilibrer le graphe différentiel d'évolution. Nous montrons alors l'optimalité en montrant la longueur sur le chemin le plus long du graphe RTL corrigé par l'algorithme est plus petite que celle obtenue par n'importe quelle solution *C*.

On notera P_{RCmax} le chemin le plus long (appelé aussi *chemin critique*) du graphe RTL corrigé. Puisque les longueurs des chemins sur les graphes corrigés dépendantes de la correction *C* envisagée, il est alors nécessaire d'introduire *C* comme paramètre dans les notations : $L_{RC}(P,C)$ et $L_{DC}(P,C)$. Bien entendu, ce paramètre n'intervient pas dans les graphes non corrigés. On gardera alors les notations initiales : $L_F(P)$ et $L_G(P)$.

Puisque le graphe différentiel est obtenu par soustraction des poids des arcs du graphe RTL par ceux du graphe fonctionnel. Après correction de retard, le graphe différentiel corrigé est aussi le graphe obtenu par soustraction des poids du graphe RTL corrigé par ceux du graphe fonctionnel. Donc, on peut écrire que la somme des retards sur P_{RCmax} vaut :

$$L_{DC}(P_{RC\max}, C) = \sum_{e \in P_{RC\max}} (d_{RC}(e) - d_F(e))$$

$$L_{DC}(P_{RC\max}, C) = \sum_{e \in P_{RC\max}} d_{RC}(e) - \sum_{e \in P_{RC\max}} d_F(e)$$

$$L_{DC}(P_{RC\max}, C) = L_{RC}(P_{RC\max}, C) - L_F(P_{RC\max}) \qquad [15]$$

Considérons maintenant P_{Dmax}, le chemin critique du graphe différentiel avant correction. Le chemin P_{Dmax} et P_{RCmax} se terminent par le même sommet (i.e. la sortie). Après correction, le graphe différentiel est équilibré. Donc, la somme des retards sur ces deux chemins est égale :

$$L_{DC}(P_{D\max}, C) = L_{DC}(P_{RC\max}, C)$$

La relation [15] devient alors :

$$L_{DC}(P_{D\max}, C) = L_{RC}(P_{RC\max}, C) - L_F(P_{RC\max})$$

Soit encore, $L_{RC}(P_{RC\max}, C) = L_{DC}(P_{D\max}, C) + L_F(P_{RC\max})$ [16]

Une solution quelconque C ajoute des poids positifs sur les arcs du graphe différentiel. Donc, le chemin critique P_{Dmax} peut être augmenté lorsque C ajoute des poids sur ce chemin critique. Donc, la longueur de P_{Dmax} est plus grande ou égale après correction qu'avant :

$$L_{DC}(P_{D\max}, C) \geq L_D(P_{D\max})$$

En reportant cette inégalité dans [16], on obtient une inégalité vérifiée pour n'importe quelle correction C :

$$L_{RC}(P_{RC\max}, C) \geq L_D(P_{D\max}) + L_{GF}(P_{RC\max}) \quad [17]$$

[16] est vrai pour toute solution, donc en particulier pour celle fournit par l'algorithme 1. Donc, on peut écrire :

$$L_{RC}(P_{RC\max}, C_{algo}) = L_{DC}(P_{D\max}, C_{algo}) + L_F(P_{RC\max}) \quad [18]$$

L'algorithme proposé est analogue à un algorithme de Bellman de recherche de plus long chemin. Il apporte des corrections sur les chemins les plus courts sans changer le chemin critique.

Donc, $L_{DC}(P_{D\max}, C) = L_D(P_{D\max})$

En reportant cette égalité dans [18], on obtient :

$$L_{RC}(P_{RC\max}, C) = L_D(P_{D\max}) + L_{GF}(P_{RC\max}) \quad [19]$$

Dans les expressions [19] et [17], les membres de droites sont identiques. On obtient alors la relation [20] valable pour tout solution C équilibrant le graphe différentiel :

$$L_{RC}(P_{RC\max}, C) \geq L_{RC}(P_{RC\max}, C_{algo}) \quad [20]$$

L'inégalité [20] montre que la solution de l'algorithme 1 minore toutes autres solutions. D'où l'optimalité vis-à-vis de la latence maximale.

Théorème 4. *Optimalité de la solution vis-à-vis de la latence minimale*

Enoncé : l'algorithme de correction de retard fournit une solution optimale vis-à-vis de la latence minimale.

Démonstration :

L'optimalité est démontrée en montrant que la solution *Calgo* est un minorant de l'ensemble des corrections permettant d'équilibrer le graphe différentiel. Comme précédemment, il est nécessaire d'introduire une solution notée C comme paramètre dans les notations : $L_{GRC}(P,C)$ et $L_{GC}(P,C)$. Bien entendu, ce paramètre n'intervient pas dans les graphes non corrigés. On gardera alors les notations initiales : $L_{GF}(P)$ et $L_G(P)$.

Pour toute solution C, le lemme 2 montre qu'après correction, on a :

$$L_{RC}(P_{RC\max}, C) - L_{RC}(P_{RC\min}, C) = L_F(P_{F\max}) - L_F(P_{F\min})$$

$$L_{RC}(P_{RC\min}, C) = L_{RC}(P_{RC\max}, C) - L_F(P_{F\max}) + L_F(P_{F\min}) \quad [21]$$

On minore l'égalité [21] sur l'ensemble des solutions C :

$$\min(L_{RC}(P_{RC\min}, C)) = \min(L_{RC}(P_{RC\max}, C) - L_F(P_{F\max}) + L_F(P_{F\min})) \quad [22]$$

Puisque L_F est indépendant de C, les constantes $L_F(P_{Fmin})$ et $L_F(P_{Fmax})$ peuvent sortir de la fonction *min* de [22] :

$$\min(L_{RC}(P_{RC\min}, C)) = \min(L_{RC}(P_{RC\max}, C)) - L_F(P_{F\max}) + L_F(P_{F\min}) \quad [23]$$

L'expression [23] montre que si une correction C minore la longueur du plus long chemin sur le graphe RTL corrigé, elle minore également la longueur du plus court chemin. C'est le cas pour la solution C_{algo} fournie par l'algorithme 1 d'après le théorème 3. Donc, C_{algo} est une solution minorant la longueur du plus court chemin sur le graphe RTL corrigé. D'où le résultat.

A.5. Démonstration du théorème de factorisation de retard

Théorème 5. *Factorisation de retard*

Enoncé : soit un graphe différentiel corrigé d'évolution. Soit v un sommet du graphe et $k \in Z$ une constante. Alors, le comportement du sommet v est inchangé si on supprime k corrections aux poids des arcs d'extrémités initiales v et si on ajoute ces k corrections aux poids des arcs d'extrémités terminales.

Démonstration :

Soit v un sommet du graphe différentiel corrigé. Soit $k \in Z$. Puisque la démonstration ne concerne que le graphe différentiel corrigé, l'extension « DC » sera omise pour simplifier la démonstration.

Pour chaque arc e_i arrivant en v, on note (figure 40) :

- d_i, le retard sur l'arc e_i.
- w_i, le signal de sortie du sommet prédécesseur de v sur l'arc e_i.

– x_i, le signal d'entrée du sommet v sur l'arc e_i.

Pour chaque arc e'_j sortant de v, on note :

– d'_i, le retard sur l'arc e'_j.

– y_j, le signal de sortie du sommet v sur l'arc e'_j.

– z_j, le signal de sortie du sommet successeur de v sur l'arc e'_j.

On note également f, le traitement arithmétique réalisé par v.

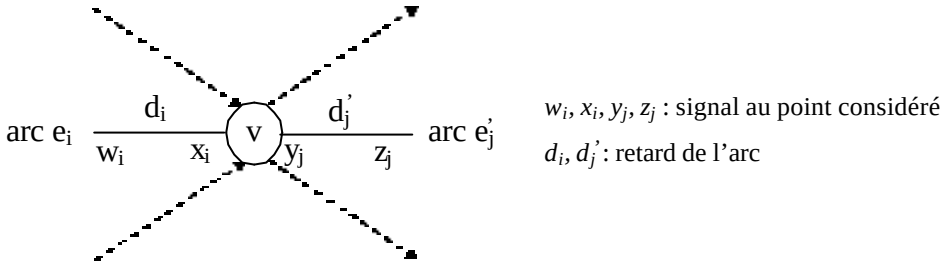


Figure 40. Notations de la démonstration du théorème de factorisation de retard

Pour chaque arc e'_j sortant de v, on exprime le signal de sortie z_j en fonction des signaux d'entrée w_i :

$$x_i = D_{di}(w_i)$$

$y_j = f(x_1,...,x_i,...,x_n)$ (où n est le nombre d'arc arrivant en v)

$$y_j = f(D_{d1}(w_1),...,D_{di}(w_i),...,D_{dn}(w_n))$$

$$z_j = D_{d'j}(y_j) = D_{d'j} \circ f(D_{d1}(w_1),...,D_{di}(w_i),...,D_{dn}(w_n))$$

Or, f est invariant dans le temps. Donc, f et D peuvent commuter. D'où :

$$z_j = f(D_{d'j} \circ D_{d1}(w_1),...,D_{d'j} \circ D_{di}(w_i),...,D_{d'j} \circ D_{dn}(w_n))$$

$$z_j = f(D_{d1+d'j}(w_1),...,D_{di+d'j}(w_i),...,D_{dn+d'j}(w_n)) \quad [24]$$

Si on supprime k corrections aux poids des arcs d'extrémités initiales v et si on ajoute k corrections aux poids des arcs d'extrémités terminales v, le nouveau graphe est similaire à celui de la figure 41, mais en remplaçant d_i par $d_i + k$ et en remplaçant d'_j par $d'_j - k$. On obtient la nouvelle expression de z_j (appelée z'_j) en effectuant ces remplacements :

$$z'_j = f(D_{d1+k+d'j-k}(w_1),...,D_{di+k+d'j-k}(w_i),...,D_{dn+k+d'j-k}(w_n))$$

$$z'_j = f(D_{d1+d'j}(w_1),...,D_{di+d'j}(w_i),...,D_{dn+d'j}(w_n)) \qquad [25]$$

L'expression [25] de z'_j est identique à l'expression [24] de z_j.

Donc, $\forall j \in [1, M], z'_j = z_j$. La factorisation de retard conserve le comportement.

A.6. Relation entre graphe équilibré et cycle

Théorème 6. *Relation entre graphe équilibré et cycle*

Enoncé : un graphe G est équilibré $\Leftrightarrow \forall \Gamma$ cycle de $G, \overline{L}(\Gamma) = 0$

Démonstration :

1) Démontrons : un graphe G est équilibré $\Rightarrow \forall \Gamma$ cycle de $G, \overline{L}(\Gamma) = 0$.

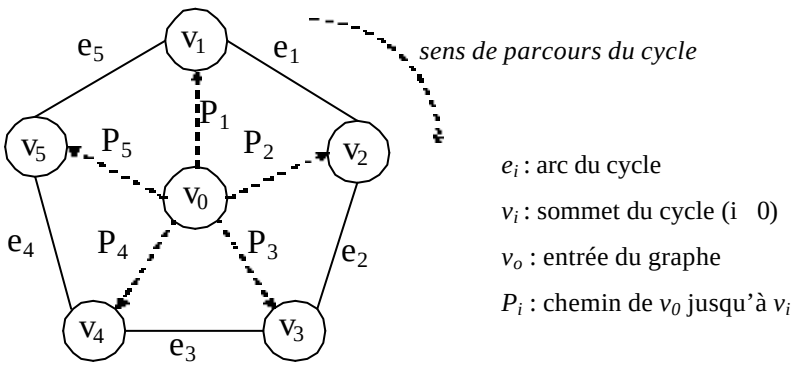


Figure 41. Notations de la démonstration de relation entre graphe équilibré et cycle

On suppose alors que G est un graphe équilibré. Soit $\Gamma = (e_1, e_2, \ldots, e_n)$ un cycle de G. Montrons que $\overline{L}(\Gamma) = 0$. Pour cela, on note (figure 41), v_i, les sommets rencontrés par le cycle Γ. Puisque tous les chemins sont atteignables, on note P_i un chemin depuis l'entrée v_0 jusqu'au sommet v_i.

Par définition, $\overline{L}(\Gamma) = \sum_{i \in [1,n]} g(e_i) \cdot d(e_i)$. [26]

Pour $i \in [1,n]$, deux cas sont possibles suivant le sens de e_i :

– si e_i est dans le sens du parcours de Γ, avec les notations adoptées, e_i a pour extrémité initiale v_i et pour extrémité terminale v_{i+1}. Donc, $P_i \cup e_i$ est un chemin allant de l'entrée jusqu'à v_{i+1}. Par définition, P_{i+1} est également un chemin allant de l'entrée jusqu'à v_{i+1}. Or, le graphe G est équilibré, donc la somme des retards sur les chemins $P_i \cup e_i$ et Pi+1 sont égaux :

$$L(P_i \cup e_i) = L(P_{i+1})$$

$$L(P_i) + d(e_i) = L(P_{i+1})$$

$$d(e_i) = L(P_{i+1}) - L(P_i)$$

Or, ei est dans le sens du parcours. Donc, $g(e_i) = 1$ et :

$$g(e_i) \cdot d(e_i) = L(P_{i+1}) - L(P_i) \quad [27]$$

– si e_i est dans le sens inverse du parcours de Γ, e_i a pour extrémité initiale v_{i+1} et pour extrémité terminale v_i. Donc, $P_{i+1} \cup e_i$ est un chemin allant de l'entrée jusqu'à v_i. Par définition, P_i est également un chemin allant de l'entrée jusqu'à v. Or, le graphe G est équilibré, donc la somme des retards sur les chemins $P_{i+1} \cup e_i$ et P_i sont égaux :

$$L(P_{i+1} \cup e_i) = L(P_i)$$

$$L(P_{i+1}) + d(e_i) = L(P_i)$$

$$d(e_i) = L(P_i) - L(P_{i+1})$$

Or, e_i est dans le sens inverse du parcours. Donc, $g(e_i) = -1$ et :

$$g(e_i) \cdot d(e_i) = L(P_{i+1}) - L(P_i) \qquad [28]$$

Par conséquent, les expressions [27] et [28] de $g(e_i) \cdot d(e_i)$ sont identiques quelques soient le sens de parcours de e_i. On remplace [28] dans [26] :

$$\bar{L}(\Gamma) = \sum_{i \in [1,n]} (L(P_{i+1}) - L(P_i))$$

$$\bar{L}(\Gamma) = \sum_{i \in [1,n]} L(P_{i+1}) - \sum_{i \in [1,n]} L(P_i)$$

En changeant d'indice dans la première somme, on obtient :

$$\bar{L}(\Gamma) = \sum_{i \in [2,n+1]} L(P_i) - \sum_{i \in [1,n]} L(P_i)$$

Comme Γ est un cycle alors $P_{n+1} = P_1$. Donc,

$$\bar{L}(\Gamma) = \sum_{i \in [1,n]} L(P_i) - \sum_{i \in [1,n]} L(P_i)$$

$\bar{L}(\Gamma) = 0$. D'où le résultat.

2) Démontrons : $\forall\, \Gamma \text{ cycle de } G, \bar{L}(\Gamma) = 0 \Rightarrow \text{le graphe } G \text{ est équilibré}$.

On suppose alors que $\forall\, \Gamma \text{ cycle de } G, \bar{L}(\Gamma) = 0$. Montrer que G est un graphe équilibré revient à montrer que pour tout sommet v, le retard cumulé sur tout chemin depuis l'entrée jusqu'à v est constant.

Soit v un sommet de G. Soit P_1 et P_2 deux chemins depuis l'entrée jusqu'à v. Montrons que $L(P_1) = L(P_2)$. On note P_2' l'ensemble des arcs de P_2 mais pris dans l'ordre inverse (i.e. du

sommet *v* vers l'entrée). Puisque P_1 est un chemin de l'entrée jusqu'à *v* et que P_2' est un chemin de *v* jusqu'à l'entrée, alors $\Gamma = P_1 \cup P_2'$ est un cycle.

$$\overline{L}(\Gamma) = \sum_{e \in \Gamma} g(e) \cdot d(e)$$

Cette somme est décomposé pour distinguer les arcs de P_1 et P_2'.

$$\overline{L}(\Gamma) = \sum_{e \in P_1} g(e) \cdot d(e) + \sum_{e \in P_2'} g(e) \cdot d(e)$$

Les arcs de P_1 sont dans le sens du parcours alors que les arcs de P_2' sont dans le sens inverse. Donc, $g(e) = 1$ si $e \in P_1$ et $g(e) = -1$ si $e \in P_2'$. Donc,

$$\overline{L}(\Gamma) = \sum_{e \in P_1} d(e) - \sum_{e \in P_2'} d(e)$$

$\overline{L}(\Gamma) = L(P_1) - L(P_2)$ par définition de $L(P_1)$ et $L(P_2)$.

Or, par hypothèse, pour tout cycle, $\overline{L}(\Gamma) = 0$. Donc,

$$L(P_1) - L(P_2) = 0$$

$$L(P_1) = L(P_2)$$

Le graphe *G* est un graphe équilibré.

A.7. Extension de la solution à Z dans le cas des boucles

Théorème 7. *Possibilité d'une solution dans Z dans le cas des boucles*

Enoncé : il est toujours possible d'équilibrer un graphe différentiel lorsque les corrections apportées sont dans Z.

Démonstration :

Pour démontrer ce résultat, il suffit de poser $c(e) = -d_D(e)$ pour tous les arcs *e* du graphe différentiel considéré.

Après correction, les poids du graphe différentiel valent alors :

$d_{DC}(e) = d_D(e) + c(e) = d_D(e) - d_D(e) = 0$.

Donc, pour tout cycle du graphe, la longueur algébrique est nulle puisque tous les arcs du graphe différentiel corrigé sont nuls. Le graphe différentiel est bien un graphe équilibré (théorème 6). Les corrections choisies sont donc bien une solution permettant d'équilibrer le graphe différentiel. D'où l'existence.

A.8. Démonstration de l'obtention d'une solution par l'algorithme de correction dans le cas des boucles

Démonstration : après application de l'algorithme, on démontre que le graphe différentiel d'évolution est équilibré. Pour cela, on montre que pour tout cycle Γ, $\bar{L}(\Gamma) = 0$. On reprend les notations de la figure 41.

Après correction par l'algorithme, $\bar{L}_{DC}(\Gamma) = \sum_{i \in [1,n]} g(e_i) \cdot d_{DC}(e_i)$ [29]

Pour $i \in [1,n]$, deux cas sont possibles pour les arcs e_i :

– si $e_i \in \Psi$, aucune correction n'est apportée sur l'arc e_i. Donc, $c(e_i) = 0$.

Donc, $d_{DC}(e_i) = d_D(e_i) + c(e_i) = d_D(e_i) + 0 = d_D(e_i)$.

Si $e_i \in \Psi$, L_{min} est telle que $L_{\min}(T(e_i)) = L_{\min}(I(e_i)) + d_D(e_i)$ (ligne 3 de l'algorithme 3).

Soit encore, $d_D(e_i) = L_{\min}(T(e_i)) - L_{\min}(I(e_i))$ [30]

– si $e_i \notin \Psi$, la correction apportée vaut : $c(e_i) = L(T(e_i)) - L(I(e_i)) - d_D(e_i)$

Le poids de l'arc sur le graphe différentiel vaut alors :

$d_{DC}(e_i) = d_D(e_i) + c(e_i)$

$d_{DC}(e_i) = d_D(e_i) + L(T(e_i)) - L(I(e_i)) - d_D(e_i)$

$$d_{DC}(e_i) = L(T(e_i)) - L(I(e_i)) \quad [31]$$

Nous constatons que les expressions [30] et [31] de $d_{DC}(e_i)$ sont communes pour $e_i \in \Psi$ et $e_i \notin \Psi$. Donc : $\forall e_i \in E, g(e_i) \cdot d_{DC}(e_i) = g(e_i) \cdot [L(T(e_i)) - L(I(e_i))]$ [32]

Comme pour la démonstration de la section A.6., deux cas sont envisageables :

– si e_i est dans le sens du parcours, alors $T(e_i) = v_{i+1}$, $I(e_i) = v_i$, $g(e_i) = 1$

– si e_i est dans le sens inverse, alors $T(e_i) = v_i$, $I(e_i) = v_{i+1}$, $g(e_i) = -1$

Dans les deux cas, [8] devient : $\forall e_i \in E, g(e_i) \cdot d_{DC}(e_i) = L(v_{i+1}) - L(v_i)$ [33]

On remplace l'expression [33] dans la définition de la longueur algébrique [29] :

$$\overline{L}_{DC}(\Gamma) = \sum_{i \in [1,n]} g(e_i) \cdot d_{DC}(e_i) = \sum_{i \in [1,n]} (L(v_{i+1}) - L(v_i))$$

$$\overline{L}(\Gamma) = \sum_{i \in [1,n]} L(v_{i+1}) - \sum_{i \in [1,n]} L(v_i)$$

En changeant d'indice dans la première somme, on obtient : $\overline{L}(\Gamma) = \sum_{i \in [2,n+1]} L(v_i) - \sum_{i \in [1,n]} L(v_i)$

Comme Γ est un cycle alors $v_{n+1} = v_1$. Donc, $\overline{L}(\Gamma) = \sum_{i \in [1,n]} L(v_i) - \sum_{i \in [1,n]} L(v_i)$

$\overline{L}(\Gamma) = 0$. D'où le résultat.

Annexe B. An Efficient Methodology and Semi-automated Flow for Design and Validation of Complex Digital Signal Processing ASIC Macro-Cells

Résumé : cette annexe est un papier publié à RSP [TAM 03]. Il développe les idées résumées au chapitre III.

Sommaire :

B.1. Introduction

Digital signal processing (DSP) is used for automotive, telecommunications, wireless, multimedia, networking aerospace applications and consumer products. The increasing complexity and data rates of DSP algorithms demand Application Specific Integrated Circuits (henceforth called DSP-ASIC macro-cells). These DSP-ASIC macro-cells will be assembled with other components (e.g. processor and domain-specific cores, peripherals etc.) to build the required Multiprocessor System-On-Chip (MP-SoC).

Today, the design of DSP-ASIC macro-cell requires much time and money. It is often too rigid to follow the many modifications that occur. Indeed, the algorithm should be thoroughly tested and optimized at all design stages before final design. To verify and optimize the signal processing algorithms, they must first be specified (*floating-point specification*) and their functionality tested completely (*functional validation*). In the next step, the algorithm is transformed into macro-architecture. The optimum signal widths are determined (*fixed-point description*), balancing cost with noise and disturbances (*word-length validation*). During this step, the macro-architecture's compatibility with the original algorithm is verified (*macro-architectural verification*). Finally, the macro-architecture is transformed into Register Transfer Level (RTL) code, i.e. micro-architecture. In this final step, the designer must verify that the code matches the specified functionality and architectural constraints. This flow requires three translations of the design, expressing the functionality as gradually less sequential and more structural with requirements for re-verification at each stage. Gaps exist between languages, tools or data used at each design step. Translations are error prone and time consuming in resolving portability troubles. Additionally, opportunities for algorithmic modifications to reduce power and area are often lost due to the separation of engineering decisions. As the complexity of the DSP-ASIC macro-cells under design increases, the

development efforts increase dramatically. To keep these efforts in check and at the same time meet the design time requirements, a design flow and methodology for DSP-ASIC macro-cells that favor reuse and early error detection is essential [15,16].

B.1.1. Related work

The productivity offered by the expressive power of RTL languages is way below critical. This level is clearly too low for complex design. Researchers have worked on developing efficient flows for mastering the DSP design by raising the abstraction level of the input system description. The flows can be classified according to the abstraction level of the input description.

The high-level synthesis (HLS) [3-9] generates RTL system description starting from a behavioral system specification. However, these tools are targeted mostly for hardware designers and are unattractive to algorithm designers. The main problem with this flow is that it attempts to avoid feeding back information to algorithm designers.

Some attempt to close the gap between algorithm and hardware designers by basing synthesis tools on languages derived from general programming language such C/C++ [10-14] (e.g. HardwareC [10], Handel-C, SpecC [11], SystemC [12], etc). These languages provide concepts similar to HDLs. The modeling and the validation using these languages are faster than HDL description. However, the synthesis tools starting from these languages obscure mostly the information about the algorithm and architecture through the code generation process. The generated architectures are often incompatible with industrial requirements.

The authors in [1, 2] propose a design flow based on Simulink, a high level description environment. The flow converts a Simulink block diagram (i.e. structural description) into a layout. The flow uses an efficient DSP validation environment (i.e. Simulink/Matlab

environment [18]). The generated architecture is very close to the input description. In spite of using the high level environment, the flow starts from a structural description similar to RTL code. The productivity offered by the expressive power of structural description is way below critical. The use of floor-plan description involves that the technology migration is very expensive in time.

However, we believe that in all above works, design methodologies tackle some issues of DSP design but they have yet to encompass the entire problem. In fact, most of above-mentioned approaches cannot satisfy a tradeoff between architecture quality, rapid algorithm/architecture exploration, and fast modeling and validation.

B.1.2. Contribution

The main contribution in this paper is the definition of new methodology and semi-automated flow which tackle the design and validation of DSP-ASIC macro-cells in an efficient way. The flow generates RTL architecture starting from a functional architecture model of digital system. The flow is based on the automatic assembly of pre-designed generic DSP basic blocks, this allows increasing reuse factor. The flow includes a fast verification platform that drives both algorithm and architecture validation within functional validation environment. The key characteristics of the proposed methodology (that we will justify along this paper) are:

— Mastering the increasing design complexity and reducing strongly the design effort;

— Mastering the architecture generation as if it had been designed manually;

— Allowing efficient design space exploration and rapid prototyping;

— Allowing the macro-cell generation that will be easily integrated within using an existing design flow, in order to produce a complete MP-SoC.

The rest of the paper is organized as follows. Section II presents some basic concepts adopted and introduces our methodology. Section III details the design and validation steps of the methodology and the associated design flow. Section IV presents the experimental results. Finally, section V provides our conclusions.

B.2. Preliminaries

In our methodology a single model is defined to represent a DSP system in all phases during the design and validation process. The model consists basically of two levels: the functional level and RT level. At functional level, the system is represented as Data Flow Graph (DFG). The edges represent the flow of data and control signals between nodes. The nodes represent the pre-designed functional DSP basic blocks (F-DSP-IP, *Functional DSP Intellectual Properties*). The DFG represents the functional description written in high level language (Matlab) of the DSP system (Figure 42).

```
% DSP Functional Architecture Model
% Name of DSP Application : DSP_Appli_Example
% Description :

function [out] = DSP_Appli_Example (in)

% feedback signal
  persistent r;
  if (isempty(r))
    r = 0;
  end

% parameters of functional-IP are now known and fixed
  [p1, p2, p3, p4] = read_parameter ('param_file.txt');

  x = Func_DSP_IP1(in, p1);
  y = Func_DSP_IP2(x, r, p2);
  out = Func_DSP_IP3(y, p3);

% update 'r' for the next execution
  r = Func_DSP_IP4(y, p4);
```

Figure 42. Functional architecture model

At the RT level, the system is represented by two graphs written in hardware description language (VHDL): a Data-Path Graph (DPG) and State Transition Graph (STG). The DPG is analogous to a schematic diagram; it describes the design in terms of pre-designed RTL DSP basic blocks (RT-DSP-IP, *Register Transfer level DSP Intellectual Property*), and their interconnections. The STG is used to model a global finite state machine (FSM). Figure 43 shows the synoptic structure of RT level model. In our methodology, functional modeling and RT architecture generation are performed using libraries of generic pre-designed basic blocks and macros. Three libraries are supported: generic F-DSP-IP library, generic RT-DSP-IP library and state-machine macros library.

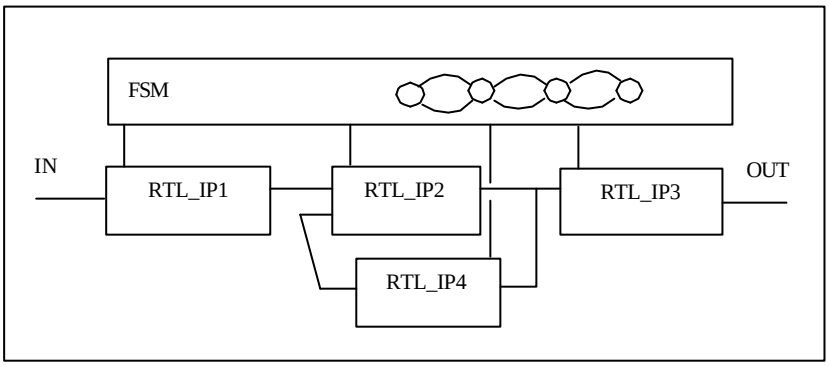


Figure 43. RTL model synoptic structure

GF-DSP-IP Block: The functional basic block is the function within the meaning of the sequential languages such as C. It describes in algorithmic form, the mathematical relation between input and output signals. In our approach, we define the GF-DSP-IP (Generic Functional DSP IP) as "*template*" described in Matlab hybrid representation; many details are left open, only some signals which are relevant for the quantification are implemented in quantified integer. The GF-DSP-IP blocks are ranged in library by complexity and application kind. An example of GF-DSP-IP description is shown in Figure 44.

```
% DSP Library : Functional Template
% Filters library Box
% Name IP : FILTER_FIR
% Description : Finite Impulse Response Filter

function [out] = FILTER_FIR (in, nbit_in, coeff)

% compute nbit_sum and nbit_round
  nbit_sum = nbit_in + log2(sum(abs(coeff))) + 2;
  nbit_sum = ceil(nbit_sum);
  nbit_round = nbit_sum - nbit_in - 1;

% Init persistent variable : delays line
  persistent dl;
  if (isempty(dl))
    dl = zeros(1,length(coeff));
  end

% Update delay line
  dl(2:end) = dl(1:end-1); dl(1) = in;

% Compute current output
  out_filter = dl*coeff;

% quantify the output
  out_rnd = round(out_filter/(2^nbit_round));
  out = satur(out_rnd, -2^(nbit_in-1), (2^(nbit_in-1))-1);
```

Figure 44. GF-DSP-IP description of FIR filter

GRT-DSP-IP: We define the Generic Register Transfer level IP block (GRT-IP) as the architecture template describes the structural implementation of DSP basic block. Each GF-IP is mapped on one or more GRT-IPs according to the desired implementation specificity, but stilling generic like the GF-IP. Some implementation details are left open at the GRT-IP and have to be filled in by the final design step in our flow. One kind of architecture template structure (i.e. GRT-IP) is shown in Figure 45 (where the generic parameters are in italic).

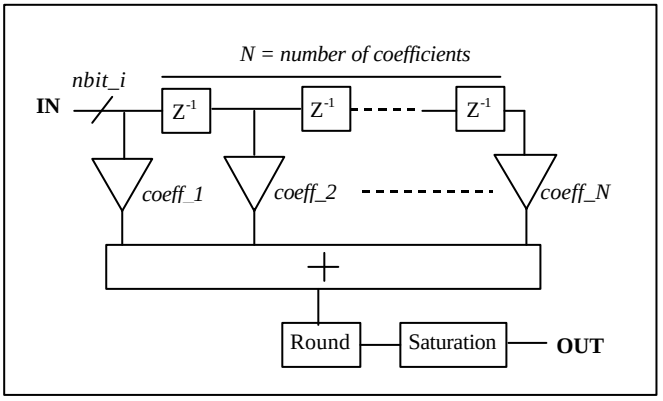


Figure 45. GRT-DSP-IP description of FIR filter

Interface Models: In order to draw an analogy and establish a relationship (in terms of interconnection issues) between RTL and functional IP blocks, we needed to study the interface models of generic IP blocks. Figure 46 draws the analogy between GF-IP and GRT-IP interfaces. The external concepts (e.g. external ports-structure, functional, and timing details, generic parameters …) of the interface model are provided to show how the generic IP component exchanges information with its environment. The GF-IP interface defines the component name, I/O data-stream names, and parameters names. The external interface concepts of GRT-IP block are described by the component definition (including component name, generic parameters names, ports names, ports directions, and ports data types). The ports can be data, clock, reset, control and test ports.

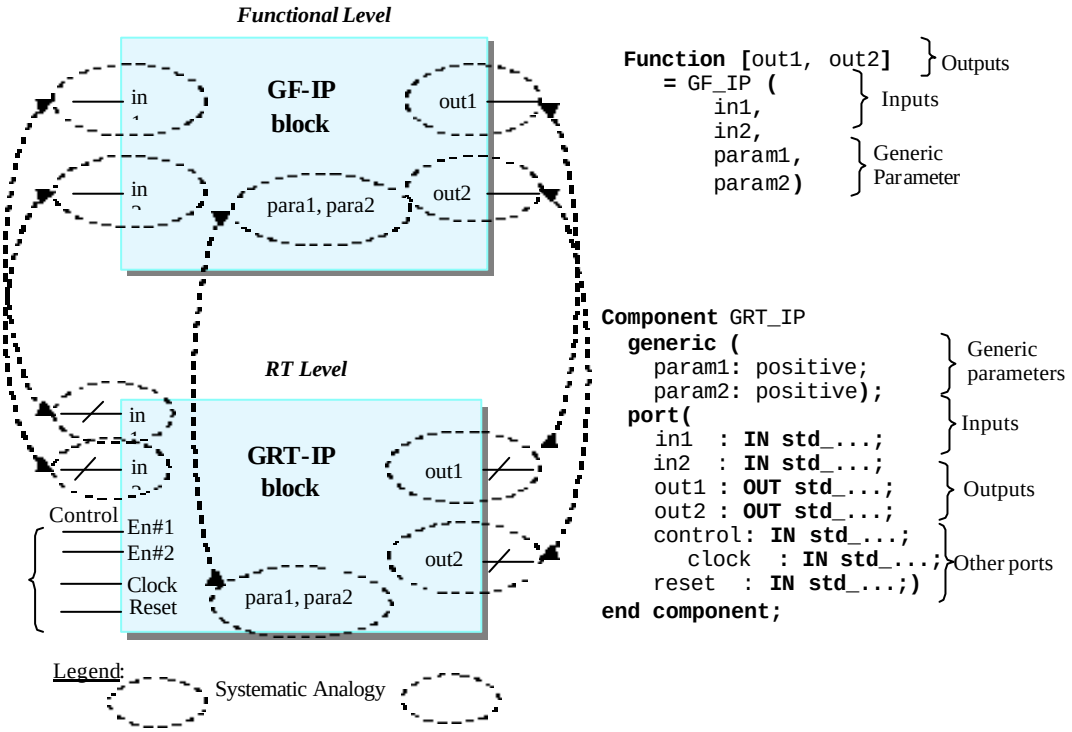


Figure 46. Interface concepts analogy

B.3. Design and validation methodology

An overview of our design flow is depicted in Figure 47. The input of the system is functional description in a high level language (Matlab). The output is RTL architecture (VHDL code).

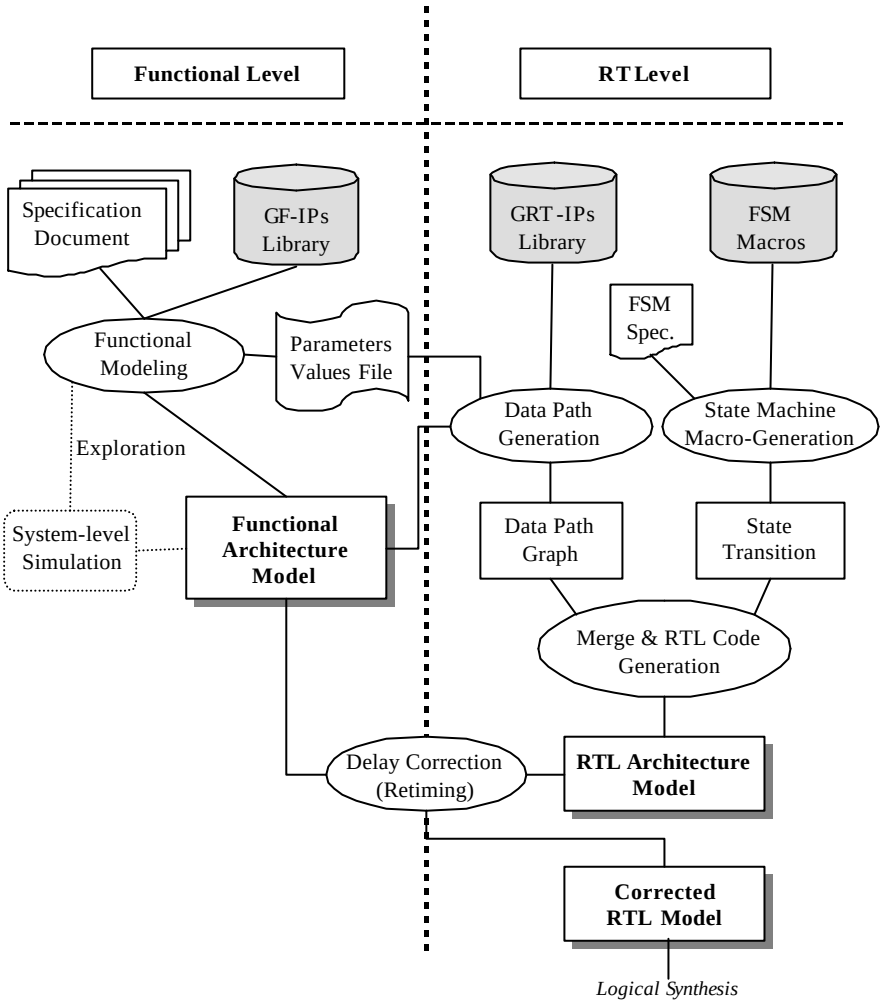


Figure 47. Proposed semi-automated design flow for DSP application

The functional architecture model is performed by assembly a generic GF-IPs. The choices of IPs, IP parameters values and assembly topology, are carried out by designer to satisfy a trade-off between signal quality and implementation constraints. To generate the architecture (i.e. Data Path Graph), IP parameters values are first extracted from functional architecture

model, and then used to instantiate the generic RTL-DSP basic blocks. Architecture is finally built by automatic assembly of pre-designed RTL-IP blocks with same assembly topology as functional architecture model. According to implementation and functional constraints, the hardware designer can choose from different kinds of GRT-IPs by loading the appropriate GRT-IP. A global finite state machine (i.e. State Transition Graph) is generated (i.e. macro-generation-based) from a high-level FSM specification written by the designer. The composition of DPG and STG gives the final RTL architectural model. In some cases, we have identified a behavior difference between functional and RTL models. For the same test bench, both the RTL and the functional models of the whole application, produce different numerical values. This problem is generally known as retiming issue. To solve this issue, we have developed a systematic method called "delay correction method" (cf. §3.3), in order to make up for the delay and to close the gap between RTL model and functional model. The design flow includes a unified verification platform used to verify both RTL and functional model. The platform exploits directly the high-level environment used for functional validation. The results of the methodology are a safety functional and RTL models of the whole DSP application. The functional model can be used such as an executable reference for the next generation of design. The RTL model is suitable for logical synthesis.

The next sections detail the main design and validation steps of the methodology and the associated design flow.

B.3.1. Functional Modeling

For modeling and validation of the whole DSP application, a fast functional design and validation environment is available. It includes high-level language, system-level simulator, and GF-IP library. Thanks to the modularity of DSP algorithms, the generic modeling, and the assembly technique, the methodology acts at the high abstraction level and adopts the "divide-

and-conquer strategy". Starting from the specification document, the system designer decomposes the DSP system into set of subsystems. Each subsystem must be decomposed into simpler generic DSP functions of same type (i.e. DSP basic block). Once, all GF-IPs were selected and the assembly topology was defined, the system designer can easily describe the whole functionality of DSP system in functional structure form by using high level language (Matlab). In this preliminary specification, some functional constraints can be distributed on each GF-IP (e.g. noise budgeting, I/O frequencies management). This specification results in a hybrid description (*Hybrid Functional Model* [17]) of the algorithm since only some of the parameters are fixed, but the others remain generic. The next step is to determine the architectural parameters values which satisfy a trade-off between signal quality and implementation constraints. DSP algorithms are sensitive to finite word length effects. The signal-to-noise ratio is an integral part of functional modeling of DSP system. As noise is an essential aspect of the behavior of digital system, noise is the responsibility of the system designer. DSP architecture can be optimized towards area by minimizing the word size under the constraint of avoiding limit cycles and round-off noise. Parameters values exploration is performed by using functional simulation and validation platform (cf. §3.4).

The final output result of this step is functional architecture model, i.e., fixed-point functional model (including architectural parameters values file) of DSP system.

B.3.2. Datapath and FSM Generation Phases

Three major phases can be distinguished in RTL code generation of the target architecture.

The first phase is the analyzing and graph transformation phase. It consists of parsing the source program of functional model (DFG). This phase transforms the DFG graph into structural graph, identifies the GF-DSP-IPs and replaces them by their homologous RT-DSP-IPs according to systematic analogy and relationship between their interfaces. We adopt the

same convention to name the interface elements. In some cases, this mapping is not a one-to-one relationship. In these cases, a mapping file is added to establish a link between both abstraction levels. One more issue will be addressed is the *unconnected ports* such control, clock, reset and test ports… These ports are lacking in functional model but must be available in RT level. The results of this phase are a *high-level structural model* of a system and an *unconnected ports list (i.e. template mapping file*). This structural model describes system in terms of GRT-IP instantiations and their topological interconnections following the physical hierarchy of the system (notice only the data-stream ports are connected, the others ports are not assigned). The unconnected ports list is a template file written in comprehensive pseudo-code format will be easily completed by designer according to the desired port connections and control constraints. Designer specifies the link between the unconnected ports and in the same time introduces the control constraints, which are added in form of sequences edges, and resolved through RTL architecture generation.

The second phase is datapath architecture generation. A sequence of generation steps transforms the high-level structural model into register-transfer description. The high-level structural model is first transformed into a new complete structural netlist; all ports are now assigned and/or connected according to the mapping file (the connection and assignment are disseminated across the hierarchy). The next step assigns to each GRT-DSP–IP one particular instance of the target architecture, according to the desired architectural constraint. The final step is the RTL code generation of the datapath architecture in accordance with coding rules for logical synthesis.

The last phase is the finite state machine generation. This phase starts from high-level FSM specification file, uses a FSM macros library and produces RTL code of global FSM of the target architecture. The designer specifies the control steps in form of transition behavior with

action statements annotated as labels and added to a mapping file elaborated during the first phase. This specification is translated into synthesizable VHDL code.

B.3.3. Delays Correction Method : Retiming

As discussed earlier, in some situations we have identified a behavior difference between functional and RTL models during validation phase. In this section, we first study and classify the reasons causing the behavior difference. At last, we propose an overall solution so that the behavior difference can be systematically identified and corrected in a methodical way. We define two models as functionally equivalent when they produce the same sequence of digital values in the same order (no redundancy and no loss of value but a delay between the two sequences is allowed). Although the GF-DSP-IP and GRT-DSP-IP blocks are functionally equivalent (to the nearest delay), it happens sometimes that both models RTL and functional of the whole application (obtained by assembling the pre-designed IPs) are not functionally equivalent. Usually, this problem occurs in following cases: the model contains two convergent parallel paths, feedback-loops or time-dependent DSP-IP. In these cases, there are two reasons causing the behavior difference between RTL and functional models which are straight linked to the implementation constraints. The first one is local to each pre-designed GRT-DSP-IP block and arises from delay in the GRT-DSP-IPs. In fact the blocks are functionally equivalent to the nearest delay. The delay is due to additional clock cycles required for the implementation constraints. The second reason is global to the RTL architectural model (usually occurred in the multi-rates DSP applications) and arises from phase-difference between FSM signals.

To achieve a functional equivalence, compensation delays are cleverly added to the functional model and/or the RTL model following the case. The correction is performed in the functional model when the application contains at least one time-dependent IP block; it

consists to put a delay block in front of each time-dependent IP block. These blocks are parameterized by the number of inserted delays. In the other cases, the delay correction is performed in the RTL model by inserting registers between IP blocks, in an efficient way. Conceptually, there are various steps will be performed in the delay correction method such verification of the functional equivalency by simulation, problem identification (time-dependent IP blocks, parallel IP-blocks or/and feedback loop IP-blocks), determination of the delays number and applied the correction to functional model, computation of registers number, registers partitioning/allocation and applied the correction to RTL model. The adopted method uses graph theory formalization as basis and combines several complementary algorithms (Bellman algorithm and integer linear programming) to produce the optimal solution in terms of area overhead cost and latency.

B.3.4. Verification Platform

We propose a fast verification platform which takes advantages of the signal processing capabilities of the functional validation environment, in order to perform the RTL model verification. The use of a functional validation environment allows a better data analyze and diagnostic than the exploitation of a typical waveform viewer. The platform allows checking automatically that the RTL model products the same output values than the functional model. In our verification platform, the functional model is executed using data input stimuli produced from a pattern generator. The results can be analyzed using post-processing algorithm and save in a *result data file*. The used input stimuli are saved in a *stimuli data file* to drive the RTL simulation. The RTL compilation, simulation process, analysis and comparison between the RTL and functional results are started automatically from functional validation environment. The system produces a diagnostic file, which indicates whether the behavior difference and signals phase differences.

B.4. Design Industrial Example

In this section, we present the results that we have obtained from the design of a complex industrial DSP system of ST Microelectronics (i.e. TV digital transmission satellite application). The methodology and the design flow have been applied on the digital modulation chain of this application. The both functional and RTL models were validated. The analysis of the result shows the effectiveness of our approach. The modulation chain is typically used in a large field of telecommunication applications (cellular phone, digital radio, modem…). Hence it must strongly be reusable. It can be dedicated to be inserted with other components in a complete SOC [19].

B.4.1. Digital Modulation Chain

To verify the feasibility and effectiveness of our approach, we keep the similar specifications (i.e. functional and implementation constraints) as the real case. As shown in figure 48, the real and imaginary parts of the input signal are first separated by a demultiplexer. Then, each signal component is processing to be adapted at the output frequency. The frequency adaptation is a chain composed of a *SRC filter* and *6 interpolator filters by 2*. Finally, the real and imaginary parts are modulated by a modulator. The input frequency is 2.048 MHz. The output frequency is 200 MHz. The carrier frequency is between 5 MHz and 54 MHz. The area of the complete chip is also a constraint to be optimized. The design difficulties of this application are: the multi-frequency (I/O frequencies ratio is irrational), the high output frequency and the strict quality requirements of output signal.

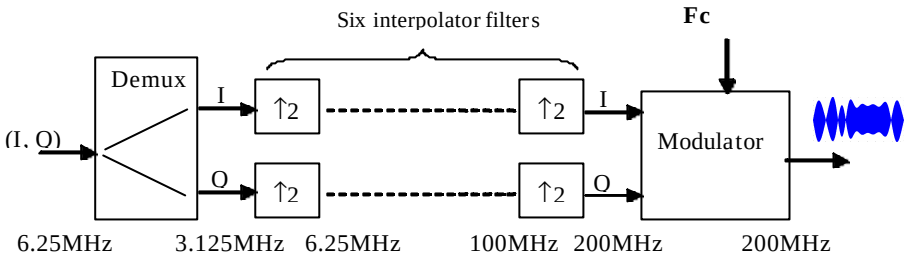


Figure 48. Digital Modulation Chain Subsystem

B.4.2. Design the Digital Modulation Chain

From the system architecture, a functional application has been modeled and validated in Matlab [18]. Among all the architectural parameters values of the design space, we found the parameters values which allow to respect functional and implementation constraints. The parameters values have been obtained by simulation from the functional model following the proposed exploration approach. As instance, the table 7 shows the selected parameters of the 6 interpolator filters. The explored parameters are the number of coefficients and the number of bits to quantify the coefficients. From these parameters, the coefficients values can be calculated mathematically choosing an appropriate integration window. The figure 49 shows the frequency responses of each interpolator designed with the 80 dB constraint requirement.

InterpolatorFilter	Coef. Nb	Bits Nb/ Coef.	Windows	Coefficients values
Filter 1	19	18	Chebyshev80	49 0 –813 0 4712 0 -17984 0 79571 79571 0 -17984 0 4712 0 -813 0 49
Filter 2	15	16	Chebyshev80	-24 0 529 0 -3487 0 19367 32768 19367 0 -3487 0 529 0 -24
Filter 3	15	16	Chebyshev80	-24 0 529 0 -3487 0 19367 32768 19367 0 -3487 0 529 0 -24
Filter 4	7	5	Hanning	-1 0 9 16 9 0 -1
Filter 5	7	5	Hanning	-1 0 9 16 9 0 -1
Filter 6	3	2	Hanning	1 2 1

Tableau 7. Extracted parameters values of 6 interpolator filters

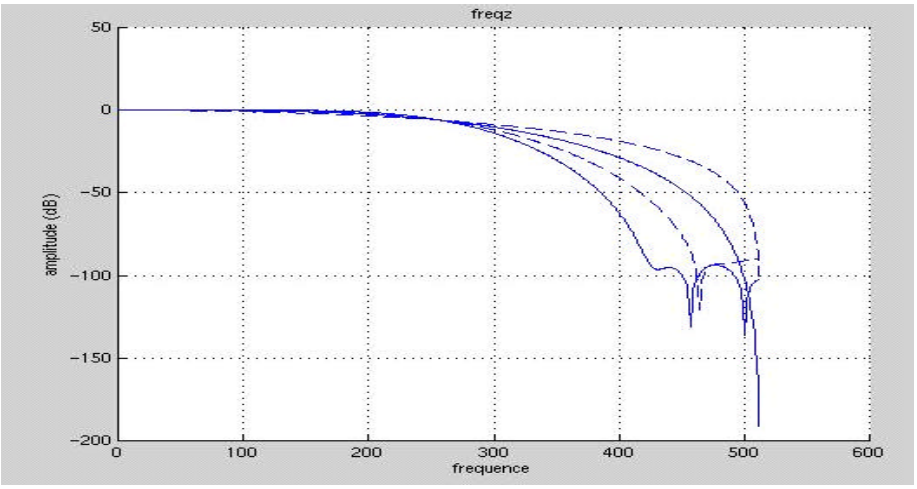
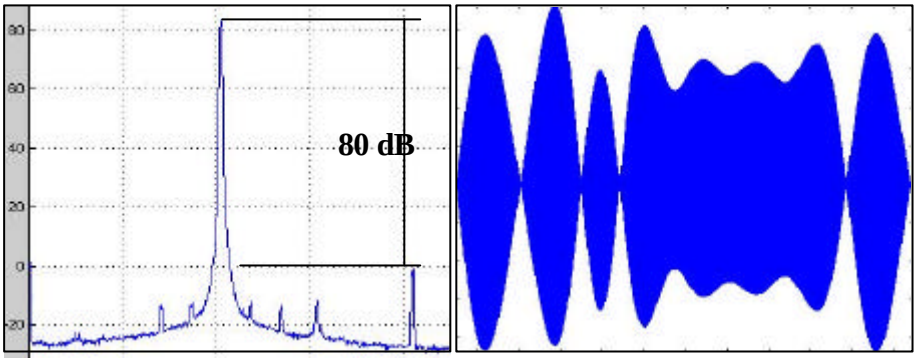


Figure 49. Frequencies responses of interpolator filters by 2

The final output result of this phase is complete functional architecture model of digital modulation chain in fixed-point. The obtained model perfectly respects the required constraints. In the RTL model generation phase, we applied the various steps of our methodology to design the datapath and FSM. The result of this phase is the RTL model of the digital modulation chain. The next phase is the analysis and comparison between the RTL and functional results which performed from the functional validation environment. During this phase, the system detected a behavior difference between the both models. The diagnostic file indicated that the behavior difference comes from counters in the SRC filter and the modulator (i.e. time-dependent block) as well as signals phase differences. A delays correction has been performed on the functional model to exactly obtain the same digital values. This correction is performed in adding delay blocks between the functional DSP-IPs blocks. The number of compensatory delays has been determined by system-level simulation approach. Finally a RTL model of digital modulation chain (e.g. 600 MGates) satisfying both implementation and functional constraints has been obtained as shown in Figure 50.



Frequency response Temporal response

Figure 50. Digital modulation chain response

B.4.3. Result Analysis

Table 8 shows the code lines number of each pre-designed generic DSP-IP block. In our approach, we assumed that all IP blocks are available. In this case, the number of code lines written manually to assemble the IP blocks, in order to design functional model, is of same magnitude as the number of the calls of GF-DSP-IP blocks (i.e. a few Matlab lines). The VHDL model is automatically assembled, only some lines (pseudo-code) must be written to allow the generation of the FSM.

Generic DSP-IP	Functional DSP-IP (Number of Matlab lines)	RTL DSP-IP (Number of VHDL code lines)
Demultiplexer	6	78
InterpolatorFilter	15	379
SRC_Filter	134	711
Modulator	84	450

Tableau 8. Matlab versus VHDL code line numbers

Table 9 shows the cost in time to explore and to fix the all parameters values of the generic IPs. Six hours are necessary to fix the all parameters that keep to the required constraints. When the 80 dB constraint is changed, the new parameters values can immediately be reported into the RTL model. Different derivative architectures can be quickly explored

changing the system architecture in the functional model. For instance, a chain of interpolator filters by 4 or by 8 can be tried. Also, the methodology allows to quickly trying different IP architectures when they are available in the library.

Generic DSP-IP	Exploration Time (parameters extraction)
Demultiplexer	0
Six InterpolatorFilter(s)	~ 4 hours
SRC_Filter	~ 1 hours
Modulator	~ 1 hours

Tableau 9. Cost in time for parameters exploration of each IP

Considering the simulation speed, the simulation time of functional model is 40 times faster than the RTL model. The system-level simulation of digital modulation chain takes approximately 90 seconds, whereas the cycle accurate simulation takes more than 1 hour. Matlab model has been translated into C code via Matlab Compiler for fast simulation. The parameters values have been obtained by simulation from the functional model. The implementation constraints are directly specified in the functional model during the choice of the system architecture, the parameters values and IPs architectures. This allows a fast both algorithm and architecture explorations as well as to sensibly increase the reuse factor. The factor reuse is estimated at more 90% compared to a manual design. For instance, both the functional and RTL models of the chain of modulation can be obtained and validated in less than 24 hours, according to designer expertise and the availability of IPs blocks.

B.5. Conclusion

In this paper, we presented a design and verification methodology for DSP ASIC macro-cells. The design and verification methodology is based on the assembly of pre-designed generic DSP-IPs. In our methodology, the designer is in center of the design process and masters the resulting architecture as a manual design. The RTL verification is performed from

the environment used for functional validation. This allows reusing the validation environment for more efficient RTL verification. The proposed methodology allows the master of the resulting architecture, important reuse factor, quick architecture exploration and efficient verification. It has been applied on a concrete industrial.

References

[1] Da Silva, L, et al.: *Design methodology for Pico Radio networks.* Proc. IEEE/ACM Design, Automation and Test in Europe, March 2001, pp. 314–323.

[2] Davis, W. R, et al.: *A Design Environment for High Throughput Low-Power Dedicated Signal Processing Systems.* IEEE Journal of Solid-State-Circuits Vo. 37, No 3 March 2002.

[3] Jerraya. A.A, et al.: *Behavioral Synthesis and Component Reuse with VHDL.* Kluwer Academic Publishers, 1996.

[4] Hurk, J. Dilling, E.: *System Level Design, a VHDL Based Approach*. Proceedings of Euro-DAC 1995.

[5] Elliott, J.P.: *Understanding Behavioral Synthesis: A Practical Guide to High-Level Design*. Kluwer Academic Publishers, 1999.

[6] Gajski, D., Ramacahndran, L.: *Introduction to high level synthesis*. IEEE Design and Test Computer, October 1994.

[7] Genoe, M., et al.: *On the use of VHDL-based behavioral synthesis for telecom ASIC design*. In the Proceedings of the International Symposium on System Synthesis ISSS'95, February 1995.

[8] Lauwereins R., Engels M., Ade M.: *GRAPE-lI: A System-Level Prototyping Environment for DSP Applications*. IEEE Computer, February, 1995, PP 35-43.

[9] Lee M.T., et al.: *Domain-specific high-level modeling and synthesis for ATM switch design using VHDL*. DAC'96.

[10] Ku, D.C. and De Micheli, D*: HardwareC – A Language for Hardware Design*. Stanford University, Technical Report CSL-TR-90-419, 1988.

[11] SpecC. Available at http://www.ics.uci.edu/~specc/

[12] SystemC. Available at http://www.systemc.org/

[13] CoWare N2C. Available at http://www.coware.com

[14] De Micheli, G.: *Hardware Synthesis from C/C++ Models*. ACM/IEEE DATE'99, pp.382-383, March 99.

[15] Martin, G., Salefski, B.: *Methodology and Technology for Design of Communication and Multimedia Products via System-Level IP Integration*. ACM/IEEE DATE 1998.

[16] Koegst, M. and al.: *A Systematic Analysis of Reuse Strategies for Design of Electronic Circuits.* DATE 1998.

[17] Lee E.A., and Sangiovanni-Vincentelli, A.:"*A framework for comparing models of computation,*" IEEE Trans. CAD, vol. 17, pp. 1217–1229, Dec. 1998.

[18] The MathWorks, Inc. Matlab, Simulink and stateflow. Available at http://www.mathworks.com.

[19] Zergainoh N., et Al.: *Framework for System Design, Validation and Fast Prototyping of Multiprocessor SoCs.* Chapter in book, Edited by B. Kleinjohann, Kluwer Academic Publishers, April 2001.

Bibliographie

[AHO 86] Aho A., Sethi R., Ullmann J., « Compilers, Principles », *Addison-Wesley Publishing*, 1986.

[AID 03] Aidi H., Berhouma R., « Flot semi-automatique de conception et de validation pour macro-cellules ASIC dédiées au traitement du signal », rapport de stage, ENSERG, 2003.

[AOY 85] Aoyagi T., Fujita, Tanaka H., « Temporal Programming Language Tokio », in *Logic Programming Conf.'85*. Berlin: Springer-Verlag, pp. 128-137, 1985.

[ART 01] « ART Designer, Reference Manual, Version 2.2 rev 25 », Frontier Design, 2001.

[AUG 97] Augé I., Bawa R.K., Guerrier P., Greiner A., Jacomme L., Pétrot F., "User Guided High Level Synthesis" in proceedings of the International Conference on Very Large Scale Integration (VLSI'97), Gramado Brasil, August 1997, pp. 464-475.

[BAR 73] Barbacci M., « Automatic Exploration of the DesignSpace for Register Transfer (RT) Systems », Thèse de doctorat, Dept. Of CS, Carnegie-Mellin University, 1973.

[BER 97] Berrebi E., « Méthodologie pour l'application industrielle de la synthèse comportementale », Thèse de doctorat, Institut National Polytechnique de Grenoble, 1997.

[BER 99] Bergamaschi R.A. , « Behavioral Network Graph: Unifying the Domains of High-Level and Logic Synthesis », *36th ACM/IEEE DAC*, June 1999.

[BIE 92] Biesenack J., Koster M., « The Siemens High Level Synthesis System, CALLAS », in proc. of *High-Level SYnthesis Workshop*, 1992.

[CAM 89a] Camposano R., Rosenstiel W., « Synthesizing Circuits From Behavioral Descriptions », *IEEE Trans. on CAD*, vol. 8, no. 2, pp. 171-180, Feb. 1989.

[CAM 89b] Camposano R., Tabet R.M.., « Design Representation for the Synthesis of Behavioral VHDL Models », in *Proc. Of the 9th Int'I Conf on CHDL*, Juin 1989.

[CAM 91] Camposano R., Bergamaschi R.A., Haynes C.E., Payer M., Wu S., « High Level VLSI Synthesis », *volume The IBM high-level Synthesis System, Kluwer Academic Publishers*, 1991.

[CAR 03] Caron S., « Génération automatique de macro-cellules DSP ASIC à partir d'un modèle fonctionnel », rapport de stage, Polytech'Grenoble, 2003.

[CES 00a] Cesario W., Gauthier L., Lyonnard D., Nicolescu G., Jerraya A.A., « An XML-based meta-model for the design of multiprocessor embedded systems », in VHDL International User's Forum (VIUF) Fall Workshop, Orlando, FL, USA, 2000.

[CES 00b] Cesario W.O., Sugar Z., Moussa I., Jerraya A.A. « Rethinking Behavioral Synthesis for a Better Integration within Existing Design Flows », *International Conferenece on Computer Design (ICCD)*, 2000.

[CES 01a] Cesario W., Nicolescu G., Gauthier L., Lyonnard D., Jerraya A.A. « Colif : a multilevel design representation for application-specific multiprocessor system-on-chip design », 12th IEEE International Workshop on Rapid System Prototyping (RSP 2001), Monterey, California, USA, 2001.

[CES 01b] Cesario W., Nicolescu G., Gauthier L., Lyonnard D., Jerraya A.A. « Colif : a design representation for application-specific multiprocessor SOCs », in IEEE Design & Test of Computers, Vol. 18, No. 5, September/October, 2001.

[CES 02] Cesario W., Jerraya A.A., "La conception comportementale", chapitre dans "Conception de haut niveau des systèmes monopuces", pp 65-106 traité EGEM Electronique - Génie Electrique - Microsystèmes, Série : Electronique et Micro-électronique, HERMES Science Publications, Paris, 2002.

[COC 00] « CoCentric SystemC Compiler, User Guide », Version 2000.11-SCC1, Synopsys, 2000.

[COT 00] Cottet F., « Traitement du signal, aide-mémoire », Edition DUNOD, 2000.

[DAS 01] Da Silva L., et al., « Design Methodology for Pico Radio Networks », Proc. IEEE/ACM Design, Automation and Test in Europe (DATE), pp. 314-323, 2001.

[DAV 02] Davis W.R., et al., « A Design Environment for High Throughput Low-Power Dedicated Signal Processing Systems », IEEE Journal of Solid-State-Circuits Vo. 37, No. 3, mars 2002.

[DER 03] Dérossi Q., Hadriani A., « Conception d'un codeur DIVX en Matlab », rapport de stage, ENSERG, 2003.

[DSP 01] DSP Builder, Quartus II and Matlab/Simulink Interface, User Guide v1.0, Altera, october 2001.

[DZI 01] Dziri A., Samet F., Wagner FL., Cesario W., Jerraya A.A. « Combining architecture exploration and a path to implementation to build a complete SoC design flow from system specification to RTL », in Asia South Pacific Design Automation Conference (ASP-DAC'03), Kitalyushu, Japan, 2003.

[ELL 99] Elliott J.P., « *Understanding Behavioral Synthesis: A Pratical Guide to High-Level Design* », Kluwer Academic Publishers, 1999.

[GAS 94] Gasjki D., Ramachandran L., « Introduction to High-Level Synthesis », *IEEE Design and Test of Computers*, 1994.

[GIR 84] Girczcy E.F., « Automatic Generation of Microsequenced Data Paths to Realize ADA Circuit Descriptions », These de doctorat, Université de Carleton, Ottawa, Canada, Juillet 1984.

[GOO 87] Goosens G., Rabaey J., Vandewalle J., De Man H., « An Efficient Micro-Code Compiler for Custom DSP-processors » , *Digest of Technical Papers of the IEEE Int. Conference on Computer-Aided Design ICCAD'87*, pp. 24-27, Santa Clara, California, November 1987.

[GOO 89] Goossens G., Vandewalle J., De Man H. « Loop Optimization in Register-Transfer Scheduling for DSP-systems », *Proceedings 26th IEEE/ACM Design Automation Conference*, Las Vegas NV, June 1989.

[GRA 85] Granacki J., Knapp D., Parker A., « THE (ADAM) Advanced Design Automation System: Overview Planner and Natural Language Interface », in proc. of the *22nd ACM/IEEE Design Automation Conference*, Los Alamitos, Ca. USA, 1985.

[GUI 03] « Matlab, the language of technical computing, creating graphical user interfaces, version 6 », Mathworks inc., 2003.

[GUO 03] Guo Y., Xu G., McCain D., Cavallaro J., « Rapid Scheduling of Efficient FPGA Architectures for Next Generation CDMA Wireless Communication Systems Using Tsunami PrecisionC Synthesizer», in 14th IEEE international workshop on Rapid System Prototyping (RSP), 2003.

[HAS 95] Hassoun S., Ebeling C., "Architectural Retiming: An Overview", TAU95, Seattle, WA, November, 1995.

[HAS 96] Hassoun S., Ebeling C., "Architectural Retiming: Pipelining Latency-Constrained Circuits", Design Automation Conference, Las Vegas, 1996.

[HAS 97a] Hassoun S., Ebeling C., "Sequential Circuit Optimizations Using Precomputation", International Workshop on Logic Synthesis (IWLS-97), June 1997.

[HAS 97b] Hassoun S. and Ebeling C., "An Overview of Prediction-Based Architectural Retiming", International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems, Dec, 1997.

[HAS 98] Hassoun S., « Fine Grain Incremental Rescheduling via Architectural Retiming », ISSS, 1998.

[HIL 85] Hilfinger P.N., « A High Level Language and Silicon Compilation for Digital Signal Processing », in proc. of *IEEE Custum integrated Circuits Conference*, pages 213-216, 1985.

[JER 96] Jerraya A.A., Ding H., Kission P., Rahmouni M., « *Behavioral Synthesis and Component Reuse with VHDL* », Kluwer Academic Publishers, 1996.

[JOH 84] Johnson S.D.,« Synthesis of Digital Designs from Recursion Equations », Thèse de doctorat, Université d'Indiana, 1984.

[KED 98] Keding H., Willems M., Coors M., Meyr H., « FRIDGE: A Fixed-Point Design and Simulation Environment », *Proceeding of DATE*, (Paris), pp. 429-435, Feb. 1998.

[KNA 96] Knapp D.W., « Behavioral Synthesis: Digital System Design Using the Synopsys Behavioral Compiler », *Prentice Hall,* 1996.

[KU 88] Ku D., De Micheli, D*.: HardwareC – A Language for Hardware Design*. Stanford University, Technical Report CSL-TR-90-419, 1988.

[KU 92] Ku D. and De Micheli, G.: *High-level Synthesis of ASICs under Timing and Synchronization Constraints*. Kluwer Academic Publishers, 1992.

[LAG 90] Lagnese T., W. Nestor, R. Blackburn, « Algorithmic and Register Transfer Level Synthesis », *The System Architect's Workbench, Kluwer*, 1990.

[LEE 96] Lee M.T., et al.: Domain-specific high-level modeling and synthesis for ATM switch design using VHDL. DAC'96.

[MAG 03] Magarshack P., «MPSOC'2003 », 3rd International Seminar on Application-Specific Multi-processor SoC, Chamonix, France, 2003.

[MAH 99] Maheshwari N., Sapatnekar S.S., *Timing, Analysis and Optimization of Sequential Circuits,* Kluwer Academic Publishers, 1999.

[MAR 01] Martin E., «Adéquation algorithme – architecture pour les communications numériques », GDR ISIS – GT7, 2001.

[MAR 02] Martin E., " Les outils de CAO de circuits et de systèmes", chapitre dans "Conception de haut niveau des systèmes monopuces", pp 203-204 traité EGEM Electronique - Génie Electrique - Microsystèmes, Série : Electronique et Micro-électronique, HERMES Science Publications, Paris, 2002.

[MAR 79] Marwedel P., « The MIMOLA Design System: Detailed Description of the Software System », in the *16th Design Automation Conference* Proceedings, pages 59-63, New York, USA, 1979.

[MAR 86] Marwedel P., « A New Synthesis Algorithm for the MIMOLA Software System », in proc. of the *Design Automation Conference*, 1986.

[MAR 93] Martin E., Sentieys O., Dubois H., Philippe J.L., « GAUT: An Architectural Synthesis Tool for Dedicated Signal Processors », In proc. of the *Europeen Conference on Design Automation*, Paris, France, 1993.

[MAT 03] Matlab. Available at http://www.mathworks.com

[MIC 88] De Micheli G., Ku D.C., « HERCULES – a system for High-Level Synthesis », in proc. of *Design Automation Conference*, 1988.

[NOT 91] Note S., « CATHEDRAL III: Architecture-Driven High-Level Synthesis for High throughput DSP Applications », in proc. *Design Automation Conference*, 1991.

[PAU 86] Paulin P.G., Knight J.P., Girzyc E.F., « HAL: A Multi-paradigm Approach to Automatic Data-Path Synthesis », in proc. of the Design *Automation Conference*, 1986.

[PAU 87] Paulin P.G., Knight J.P., « Force-Direct Scheduling in Automated Data Path synthesis », *Proceedings 24th IEEE/ACM Design Automation Conference*, Miami FL, pp. 195-202, July 1987.

[RAH 97] Rahmouni M., « Ordonnacement pour la synthèse de haut niveau », Thèse de doctorat, Institut National Polytechnique de Grenoble, 1997.

[REN 03] http://noemed.univ-rennes1.fr/sisrai/dico/21024.html

[ROM 92] Van Rompaey K., Bolsens I., De Man H., « Just in Time Scheduling », *Proceedings of the ICCD'92*, Massachusetts, October 1992.

[SAK 84a] Sakarovitch M., Optimisation combinatoire, programmation discrète, Editions Hermann, 1984.

[SAK 84b] Sakarovitch M., Optimisation combinatoire, graphe et programmation linéaire, Editions Hermann, 1984.

[SCH 01] Schneider T., *VHDL : méthodologie de design et techniques avancées*, Edition Dunod, 2001.

[SIM 00] Simulink, User's Guide, Dynamic System Simulation for Matlab, version 4, Mathworks inc., 2000.

[SOU 83] Southard J.R., « MacPitts: an approach to Silicon Compilation », *IEEE Computer Magazine*, 16(12):74-82, 1983.

[SPE 03] SpecC. Available at http://www.ics.uci.edu/~specc/

[STO 91] Stok L., « Architectural Synthesis and Optimization of Digital Systems », These de doctorat, Eindhoven University of Technology, 1991.

[STO 94] Stok L., « A Decade of High Level Synthesis: Fundementals and Applications », in proc. IX *Congress of Brazilian Microelectronics Society (SBMICRO)*, Rio de Janeiro, 1994.

[STO 98] Stocklein T., Basig J., Handel-C, an Efficient Method for Designing FPGAs (and ASICs), 1998.

[SUG 00] Sugar Z., « Synthèse comportementale basée sur l'ordonnancement », Thèse de doctorat, Institut National Polytechnique de Grenoble, 2000.

[SYS 00] « SystemC Version 1.0, User Guide », Synopsys Inc, CoWare Inc, Frontier Design Inc, 2000.

[SYS 03] SystemC. Available at http://www.systemc.org

[TAM 02] **Tambour L.**, Zergainoh N.E., Urard P., Valentin T., Ghénassia F., Jerraya A.A., "Utilisation d'une méthode de correction de retards pour la vérification d'un assemblage de fonctions RTL par rapport à un assemblage de fonctions au niveau fonctionnel », colloque GDR-CAO, Paris, 2002.

[TAM 03] **Tambour L.**, Zergainoh N., Jerraya A., Urard P., Michel H., «An Efficient Methodology and Semi-automated Flow for Design and Validation of Complex Digital Signal Processing ASIC macro-cells», 14th IEEE international workshop on Rapid System Prototyping (RSP), 2003.

[TRI 88] Trickey H., « Flamel : A High-Level Hardware Compiler », *IEEE Transaction on CAD*, vol. CAD-6, no. 2, pp. 259-269, Mars 1987.

[VSI 98] « VSI System Level Design Model Taxonomy, VSI Reference Document », Version 1.0, VSI Alliance, 1998.

[WAN 92] Wanhammar L., « *DSP integrated Circuits* », Department of Electrical Engineering, Linkoping University, Sweden, 1992.

[WAS 03] Wassfi A., « Environnement de création et de validation de librairies d'IP », rapport de stage, Polytech'Grenoble, 2003.

[WHE 94] When N., « Scheduling of Behavioral VHDL by Retiming Techniques », in proc. of *European Conference on Design Automation*, 1994.

[ZEM 86] Zemen J., Moschyts G.S., « Systematic Design and Programming of Signal Processors, using Project Management Techniques », *IEEE Transactions on Acoustics, Speech and Signal Processing*, Vol. ASSP-31, No.6, pp. 1536-1549, December 1986.

[ZER 00] Zergainoh N.E., Baghdadi A., **Tambour L**, Lyonnard D., Gauthier L., Jerraya A.A., "Framework for System Design, Validation and Fast Prototyping of Multiprocessor System-on-Chip: Applied to Telecommunication Systems" in International Workshop on Distributed and Parallel Embedded Systems, DIPES-2000 IFIP, Paderborn, Germany, 2000.

[ZER 01] Zergainoh N.E., Baghdadi A., **Tambour L**, Lyonnard D., Gauthier L., Jerraya A.A., "Framework for System Design, Validation and Fast Prototyping of Multiprocessor

SoCs" in Architecture and Design of Distributed Embedded Systems, edited by B. Kleinjohann, Kluwer, 2001.

[ZER 02] Zergainoh N.E., "Méthodologie et modèles pour la conception digitale", chapitre dans "Conception de haut niveau des systèmes monopuces", pp 28-29-33-34 traité EGEM Electronique - Génie Electrique - Microsystèmes, Série : Electronique et Micro-électronique, HERMES Science Publications, Paris, 2002.

[ZIM 80] Zimmermann G., «MDS – The Mimola Design Method », *J. Design Systems,* vol. 4, no. 3, pp. 337-369, 1980.

RESUME

Aujourd'hui, les macro-cellules ASIC dédiées au traitement du signal deviennent de plus en plus complexes, coûteuses en temps et efforts de conception. Ces macro-cellules seront ensuite assemblées avec d'autres composants IPs (e.g. processeurs, mémoires, média de communication, etc.) pour être intégrées dans un Système-sur-Puce (SoC, pour System-On-Chip). Les procédés de conception deviennent insuffisants pour conserver la maîtrise de la complexité (d'un point de vue aussi bien algorithmique qu'architectural) des nouvelles applications tout en respectant le temps de mise sur le marché.

Cette thèse est consacrée au problème de conception et de validation des macro-cellules ASIC dédiées au traitement du signal. Nous étudions et nous illustrons les possibilités d'une nouvelle méthodologie comme une alternative à la synthèse de haut niveau. Cette méthodologie se base sur l'assemblage de composants élémentaires (IPs) paramétrables et préconçus. Elle part d'une description fonctionnelle de l'application et produit le modèle RTL de l'architecture finale. Le principal problème d'une méthodologie de conception basée sur l'assemblage de composants IPs préconçus et pré-validés est que le modèle RTL de l'architecture finale peut avoir un comportement défectueux. Cela est dû à des retards induits par des contraintes d'implémentation. Nous présentons la formalisation de ce problème et proposons une méthode automatique de correction (dite correction de retard) pour le résoudre. Nous proposons deux algorithmes originaux qui garantissent des solutions optimales en latence et en surface. La faisabilité de l'approche et l'optimalité des solutions proposées sont démontrées mathématiquement. Des outils ont été développés pour transformer cette méthodologie en un flot semi-automatique. Nous illustrons l'efficacité de l'approche par l'expérimentation sur un exemple industriel : une chaîne de modulation numérique.

ABSTRACT

Today, ASIC macro-cells dedicated to signal digital processing (DSP-ASIC macro-cells in follow) become more and more complex. Their design requires much time and money. These DSP-ASIC macro-cells will be assembled with other components to build the required Multiprocessor System-On-Chip (MP-SoC). The current design processes become insufficient to master the increasing complexity (in term of algorithm and architecture) while meeting the design time requirements.

This Ph-D thesis targets the problem of the design and the validation of DSP-ASIC macro-cells. We study the possibilities of a new methodology that could be used as an alternative to high level synthesis. This methodology is based on the assembly of basic generic predesigned components (IPs). It starts from a functional description of the application and produces an RTL model of the final architecture. The main problem of a component based assembly methodology is that the RTL model can have dysfunctions. This is due to delays introduced by implementation constraints. We present the formalization of this problem and propose an automatic correction method (called delay correction method) to resolve it. The viability of the method and the optimality of the solution are mathematically proved. Tools have been implemented to transform the methodology into a semi-automatic flow. We illustrate the approach efficiency by experiment on an industrial example: a digital modulation chain.

ISBNE 2-84813-025-3 (electronic format) **ISBN 2-84813-024-5 (paperback)**