Synthèse automatique d'interfaces de communication matérielles pour la conception d'applications du domaine du traitement du signal

Cyrille Chavet

N° d'ordre : 94

THESE

pour obtenir le grade de :

DOCTEUR DE L'UNIVERSITE DE BRETAGNE SUD

Spécialité : Sciences de l'ingénieur
Mention : Electronique et Informatique Industrielle

CYRILLE CHAVET

Synthèse automatique d'interfaces de communication matérielles pour la conception d'applications du domaine du traitement du signal

Soutenue publiquement le 26 octobre 2007

COMPOSITION DU JURY

Michel PAINDAVOINE	Le2i, Université de Bourgogne	Examinateur
Daniel ETIEMBLE	LRI, Université de Paris Sud	Rapporteur
Yves MATHIEU	ENST Paris	Rapporteur
Eric MARTIN	LESTER, Université de Bretagne Sud	Directeur de thèse
Philippe COUSSY	LESTER, Université de Bretagne Sud	Examinateur
Pascal URARD	STMicroelectronics	Examinateur
Thomas BOLLAERT	Mentor Graphics	Invité

Laboratoire d'Electronique des Systèmes Temps Réel (LESTER, FRE2734)
Université de Bretagne Sud

Avant-propos

Cette thèse rentre dans le cadre d'une convention CIFRE entre STMicroelectronics (site de Crolles, FTM) et le laboratoire LESTER (FRE2734, Université de Bretagne Sud).

Les travaux décrits dans ce manuscrit ont étés réalisés conjointement dans le groupe HLS (High Level Synthesis) dirigé par Pascal URARD de la division FTM de la société STMicroelectronics, et dans l'équipe IP/HLS, dirigée par Philippe COUSSY, du laboratoire LESTER, sous la direction de M le président de l'UBS Eric MARTIN, membre du LESTER.

Je tiens à adresser des remerciements tout particuliers à la société STMicroelectronics pour avoir permis la réalisation de ces travaux dans d'excellentes conditions qui ont permis de valoriser ces recherches à la fois en termes de publications scientifiques de premier plan, mais également par le biais d'une valorisation industrielle sous la forme d'un brevet.

Remerciements

Au moment d'amener les voiles, le marin sait qu'il ne doit son arrivée au port qu'aux vents qui ont su l'y mener.

Au vif Zéphyr (Eric Martin) qui, jubilant sur les kumquats du clown gracieux et non content d'avoir affrété le brigantin, a su gonfler ses voiles et le diriger sans abattre la mâture.

A l'Aquilon (Pascal Urard) qui, par les conseils qu'il a soufflé et les tempêtes qu'il a repoussé, a permis au bateau d'arriver à bon port.

A Balaguère (Philippe Coussy) qui m'a accueilli dans son aire et dont la force et la persévérance m'ont empêché d'affaler.

A Eole (Michel Paindavoine) et aux quatre vents (Daniel Etiemble, Yves Mathieu, Thomas Bollaert et Thierry Michel), qui ont su étudier les nœuds du mousse et y voir un travail de matelot.

A tous les vents de la rose, autan, mistral, khamsin, sirocco, harmattan, noroît, simoun, cers … (membres du LESTER, équipe HLS de STMicroelectronics, membres du comité d'organisation de MajecSTIC2006…) qui, par bâbord ou tribord, gonflant brigantine ou huniers, portant chant de sirènes ou arôme de terre, rafraîchissant cale et carré, réchauffant marins et matelots, ont aidé esquif et équipage à maintenir le cap vers de nouveaux rivages.

Je ne peux oublier les bricks (Jérémie Guillot et Caaliph Andriamisaina) qui à quelques encablures, cinglaient, tout comme moi, dans le sillage des goélettes (Bertrand Legal et Sylvain Huet) qui ont ouvert la marche.

Au Fœhn (Marie-Ange), qui a su prodiguer douceur et chaleur lorsque se profilaient sargasses et dérives dans le cœur du matelot.

A tous les vents que je ne peux citer ici, qu'ils me pardonnent et sachent que je n'oublie pas leur souffle enivrant.

A ma sœur, mes parents

Résumé

Les applications du traitement du signal (TDSI) sont maintenant largement utilisées dans des domaines variés allant de l'automobile aux communications sans fils, en passant par les applications multimédias et les télécommunications. La complexité croissante des algorithmes implémentés, et l'augmentation continue des volumes de données et des débits applicatifs, requièrent souvent la conception d'accélérateurs matériels dédiés. Typiquement l'architecture d'un composant complexe du TDSI utilise des éléments de calculs de plus en plus complexes, des mémoires et des modules de brassage de données (entrelaceur/désentrelaceur pour les Turbo-Codes, blocs de redondance spatio-temporelle dans les systèmes OFDM/MIMO, …), privilégie des connexions point à point pour la communication inter éléments de calcul et demande d'intégrer dans une même architecture plusieurs configurations et/ou algorithmes (systèmes (re)configurables). Aujourd'hui, le coût de ces systèmes en terme d'éléments mémorisant est très élevé; les concepteurs cherchent donc à minimiser la taille de ces tampons afin de réduire la consommation et la surface total du circuit, tout en cherchant à en optimiser les performances. Sur cette problématique globale, nous nous intéressons à l'optimisation des interfaces de communication entre composants. On peut voir ce problème comme la synthèse (1) d'interfaces pour l'intégration de composants virtuels (IP cores), (2) de composants de brassage de données (type entrelaceur) pouvant avoir plusieurs modes de fonctionnements, et (3) de chemins de données, potentiellement configurables, dans des flots de synthèse de haut niveau.

Nous proposons une méthodologie de conception permettant de générer automatiquement un adaptateur de communication (interface) nommé Space-Time AdapteR (STAR). Notre flot de conception prend en entrée (1) des diagrammes temporels (fichier de contraintes) ou (2) une description en langage C de la règle de brassage des données (par exemple une règle d'entrelacement pour Turbo-Codes) et des contraintes utilisateur (débit, latence, parallélisme…) ou (3) en ensemble de CDFGs ordonnés et assignés. Ce flot formalise ensuite ces contraintes de communication sous la forme d'un Graphe de Compatibilité des Ressources Multi-Modes (MMRCG) qui permet une exploration efficace de l'espace des solutions architecturales afin de générer un composant STAR en VHDL de niveau transfert de registre (RTL) utilisé pour la synthèse logique.
L'architecture STAR se compose d'un chemin de données (utilisant des FIFOs, des LIFOs et/ou des registres) et de machines d'état finis permettant de contrôler le système. L'adaptation spatiale (une donnée en peut être transmise de n'importe quel port d'entrée vers un ou plusieurs ports de sortie) est effectuée par un réseau d'interconnexion adapté et optimisé. L'adaptation temporelle est réalisée par les éléments de mémorisation, en exploitant leur sémantique de fonctionnement (FIFO, LIFO). Le composant STAR exploite une interface LIS (Latency Insensitive System) offrant un mécanisme de gel d'horloge qui permet l'asservissement par les données. Le flot de conception proposé génère des architectures pouvant intégrer plusieurs modes de fonctionnement (par exemple, plusieurs longueurs de trames pour un entrelaceur, ou bien plusieurs configurations dans une architecture multi-modes).

Le flot de conception est basé sur quatre outils :

- ***StarTor*** prend en entrée la description en langage C de l'algorithme d'entrelacement, et les contraintes de l'utilisateur (latence, débit, interface de communication, parallélisme d'entrée-sortie…). Il en extrait l'ordre des données d'entrée-sortie en produisant d'une trace à partir de la description fonctionnelle. Ensuite, l'outil génère le fichier de contraintes de communication qui sera utilisé par l'outil *STARGene*.
- ***StarDFG*** prend en entrée un ensemble de CDFGs générés par un outil de synthèse de haut niveau. Ces CDFGs doivent être ordonnancés et les éléments de calculs doivent avoir été assignés. L'outil en extrait ensuite l'ordre des échanges de données. Enfin, il génère le fichier de contraintes de communication qui sera utilisé par l'outil *STARGene*.
- ***STARGene***, basé sur un flot à cinq étapes, génère l'architecture STAR : (1) construction des graphes de compatibilité des ressources MMRCG, à partir du fichier de contraintes, correspondant à chacun des modes de fonctionnement du design, (2) fusion des modes de fonctionnement, (3) assignation des structures de mémorisation (FIFO, LIFO ou Registre) sur le MMRCG (4) optimisation de l'architecture et (5) génération du VHDL niveau transfert de registre (RTL) intégrant les différents modes de communication. Le fichier de contraintes utilisé dans la première étape peut provenir de l'outil *StarTor*, comme nous l'avons indiqué, ou peut être généré par un outil de synthèse de haut niveau tel que l'outil GAUT développé au laboratoire LESTER.
- ***StarBench*** génère un test-bench basé sur les contraintes de communication et permet de valider les architectures générées en comparant les résultats de simulation de l'architecture avec la spécification fonctionnelle.

Les expérimentations que nous présentons dans le manuscrit ont été réalisées pour trois cas d'utilisation du flot STAR. En premier lieu, nous avons utilisé l'approche STAR dans le cadre de l'intégration et l'interconnexion de blocs IPs au sein d'une même architecture. Cette première expérience pédagogique permet de démontrer la validité de l'approche retenue et de mettre en avant les possibilités offertes en terme d'exploration de l'espace des solutions architecturales.

Dans une seconde expérience, le flot STAR a été utilisé pour générer une architecture de type entrelaceur Ultra-Wide Band. Il s'agit là d'un cas d'étude industriel dans le cadre d'une collaboration avec la société STMicroelectronics. En utilisant notre flot, nous avons prouvé que nous pouvions réduire le nombre de points mémoires utilisés et diminuer la latence, par rapport aux approches classiques basées sur des bancs mémoires. De plus, lorsque nous utilisons notre flot, le nombre de structures à piloter est plus petit que dans l'architecture de référence, qui a été obtenue à l'aide d'un outil de synthèse de haut niveau du commerce. Actuellement, la surface totale de notre architecture d'entrelacement est environ 14% plus petite que l'architecture de référence STMicrolectronics.

Enfin, dans une troisième série d'expériences, nous avons utilisé le modèle STAR dans un flot de synthèse de haut niveau ciblant la génération d'architectures reconfigurables. Cette approche a été expérimentée pour générer des architectures multi-débits (FFT 64 à 8 points, FIR 64 à 16 points…) et multi-modes (FFT et IFFT, DCT et produit de matrices…). Ces expériences nous ont permis de

montrer la pertinence de l'association de l'approche STAR, pour l'optimisation et la génération de l'architecture de multiplexage et de mémorisation, à des algorithmes d'ordonnancement et d'assignation multi-configurations à l'étude dans GAUT (Thèse Caaliph Andriamissaina). Nous avons notamment obtenu des gains pouvant aller jusqu'à 75% en terme de surface par rapport à une architecture naïve et des gains pouvant aller jusqu'à 40% par rapport aux surfaces obtenues avec des méthodologies centrées sur la réutilisation d'opérateur (SPACT-MR).

Abstract

Digital Signal Processing (DSP) applications are know widely used from automotive to wireless communications. The ever growing design complexity, and the performance requirements, and constraints, on design costs and power consumption still require significant parts of a design to be implemented using a set of dedicated hardware accelerators. A classical complex DSP application architecture uses several complex processing elements, a lot of memories, data mixing modules (interleaver for TurboCodes, Spatial redundancy blocks for OFDM/MIMO systems…), and is based on a point to point communication network for inter processing element communications. Such a system may also require to include several applications in a single architecture ((re)configurable systems). Today, their cost in terms of memory elements is very expensive; that's why the designers try to reduce the size of the embedded buffers in order to reduce the overall design area and consumption, and to enhance design performances. In our work, we focus on the optimisation of component communication interfaces. This problem can be seen as the synthesis (1) of interfaces for IP cores integration, (2) of data mixing blocks (such as interleavers) with multi-modes architectures, and (3) of (re)configurable datapath synthesis in high level synthesis flows.

We propose a design methodology to automatically generate and optimize a communication adapter named Space-Time AdapteR (STAR). Our design flow inputs (1) a timing diagram (constraint file) or (2) a C description of I/O data scheduling (an interleaving formula), and user requirements (throughput, latency…), or (3) a set of scheduled and bound CDFGs, and formalizes communication constraints through a formal Multi-Modes Resource Constraints Graph (MMRCG). The MMRCG properties enable efficient architecture space exploration to generate a Register Transfert Level (RTL) STAR component.
The STAR architecture is composed of a datapath (using FIFOs, LIFOs and/or registers) and the associated control state machines. Spatial adaptation (a data can be send from any input port to any/several output ports) is performed by an interconnection logic. Timing adaptation (data reordering) is realized by the storage elements. The STAR component uses a LIS interface (Latency Insensitive System) that enables to implement a *gated clock* mechanism. The proposed design flow can generate multi-modes architectures.

The design flow is based on three tools:

- ***StarTor*** inputs a C level algorithmic description which specifies the interleaving scheme, and user requirements (latency, throughput, communication interface, I/O parallelism...). It extracts I/O data order by generating a trace from the C functional description. Next, it generates the constraints file.
- ***StarDFG*** inputs a set of CFDGs generated by a High Level Synthesis tool. These CDFGs are supposed to be scheduled and bound. This tool extracts data communication order. Then, it generates the constraints file.
- ***STARGene***, based on a five-step flow, generates the STAR architecture: (1) Muli-Modes Resource Compatibility Graph construction from constraint file, (2) Modes merging step, (3)

Storage resource binding on the MMRCG, (4) Architecture optimization and (5) VHDL RTL generation.

- ***StarBench*** generates a test bench based on constraints in order to validate the design by comparing simulation results.

The experiments presented in this thesis have been performed for three different applications of the STAR design flow. First, we use our approach for IP integration and communication adaptation. This educational example demonstrates the validity of our approach. It also exposes how to use our design flow to explore the design space.

In a second experience, our design flow has been used to generate an industrial Ultra Wide Band interleaver example. This is an industrial test case and these experiments have been performed in collaboration with STMicroelectronics. Using our flow, we show that we can save memory resources and decrease the latency in any case, compared to classical approach based on memory. Moreover the number of structure to be controlled is smaller, by using our model, than in the reference design from STMicroelectronics. Currently, the total area of the generated design is about 14% smaller than the reference design from STMicroelectronics (generated with a widespread commercial HLS tool).

In the last experiments, we use de STAR design flow in a HLS flow in order to generate a reconfigurable (muli-modes) datapath. These experiments have been performed to generate multi-throughputs (FFT 64 to 8, FIR 64 to 8…) and multi-configurations (FFT and IFFT, DCT and FIR…) architectures. These experiments show the efficiency of the combination of (1) our approach and (2) the multi-modes scheduling and binding algorithms developed for GAUT (PhD thesis of C. Andriamissaina), for the generation and the optimization of the memorising part and the steering logic of a datapath. We reduce the total area up to 75% compared to a cumulative architecture, and up to 40% compared to the systems generated by a dedicated multi-modes design flow (SPACT_MR).

Tables des matières

Table des figures

Chapitre 1

CONTEXTE

Pour répondre aux exigences des utilisateurs en termes de performances et de fonctionnalités, tout en maintenant les coûts de développement des circuits à des niveaux acceptables pour l'industrie, les concepteurs ont sans cesse besoin de nouvelles méthodologies de conception. Ainsi, ces dernières années ont vu entre autre, l'émergence des systèmes sur puce, des flots de synthèse de haut niveau et d'outils de vérification formelle.

Dans ce chapitre, nous resituons ces éléments dans le cadre de la problématique de cette thèse.

1. Introduction

En 1965 Gordon Moore, alors ingénieur de *Fairchild Semiconductor* et futur fondateur d'*Intel*, constatait que la complexité des semi-conducteurs proposés en entrée de gamme doublait tous les dix-huit mois à coût constant depuis 1959, année de leur invention. Bien qu'il ne s'agisse pas d'une véritable loi physique, cette prédiction qui s'est révélée étonnamment exacte, fut rapidement nommée **Loi de Moore** (cf. Figure.1.1).

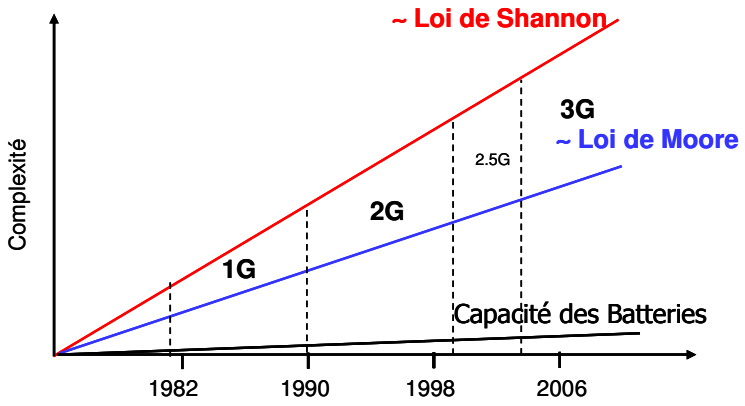


Figure.1.1. *Evolution de la complexité des applications Vs Evolution technologiques*

L'évolution technologique offre ainsi aux concepteurs les moyens d'implémenter des applications de plus en plus complexes et performantes. Toutefois, de nombreuses contraintes se posent concernant : le temps de mise sur le marché, la complexité des architectures ou les performances attendues (en termes de débit, de coût et de consommation). De fait, un fossé toujours plus grand se creuse entre les capacités nouvelles offertes par l'évolution technologique, et les niveaux de complexité que peuvent accepter et manipuler les concepteurs de circuits, grâce aux méthodologies et aux outils mis à leur disposition.

Evolutions technologiques

La Figure 1.2 (tirée de [URA07]) montre le besoin de nouvelles méthodologies de conception. En effet, on constate que l'évolution des finesses de gravure, et des technologies d'intégration, permet d'offrir de plus en plus de portes logiques sur un circuit comparable en coût et en dimension. La Figure 1.2 indique également qu'un concepteur peut produire environ 200 000 portes logiques par an. Ainsi, pour produire un circuit intégrant 15 millions de portes, il faut prévoir environ 75 Homme.Année pour son développement.

Toutefois, comme nous venons de l'évoquer, les évolutions technologiques vont, à court terme, permettre de doubler le nombre de portes logiques intégrées sur un circuit de dimension et de coût équivalents. Les systémiers auront alors la possibilité d'offrir beaucoup plus de fonctionnalités dans leur produit. Malheureusement, si de nouvelles méthodologies et de nouveaux outils de conception ne sont pas disponibles, il faudra alors également doubler le nombre de concepteurs. En effet, comme le

montre la Figure 1.2, si l'on considère qu'un développeur peut produire 200 000 portes logiques par an, pour développer un circuit intégrant 30 millions de portes, il faut alors compter 150 Homme.Année.

1992	1994	1996	1998	2000	2002	2004	2006	2008	2010
Finesse de gravure									
0,6	0,5	0,35	0,25	0,18	0,13	90	65	45	32
# de portes / mm^2									
1k	5k	15k	30k	45k	80k	150k	300k	600k	1,2M
# de portes / die(50mm^2)									
50k	250k	750k	1,5M	2,25M	4M	7,5M	15M	30M	60M
# de portes / concepteur / an									
4k	6k	9k	40k	56k	91k	125k	200k	200k	200k
Homme.Année / die(50mm^2)									
~10	~40	~80	~40	~40	~43	~60	~75	~150	~300

Figure 1.2. *Evolution technologique Vs évolution méthodologique*

Bien évidement, cet accroissement exponentiel du nombre de concepteur n'est pas viable à terme pour l'Industrie de la microélectronique, c'est pourquoi de nouvelles méthodologies de conception sont nécessaires. La Recherche en C.A.O. (*Conception Assistée par Ordinateur*) est ainsi entraînée dans un mouvement perpétuel pour proposer de nouvelles solutions, de nouvelles approches, de nouvelles méthodologies aux concepteurs de circuits. Le but est de maintenir un niveau de productivité par concepteur suffisant pour permettre le développement de circuit à un coût raisonnable et dans un temps limité.

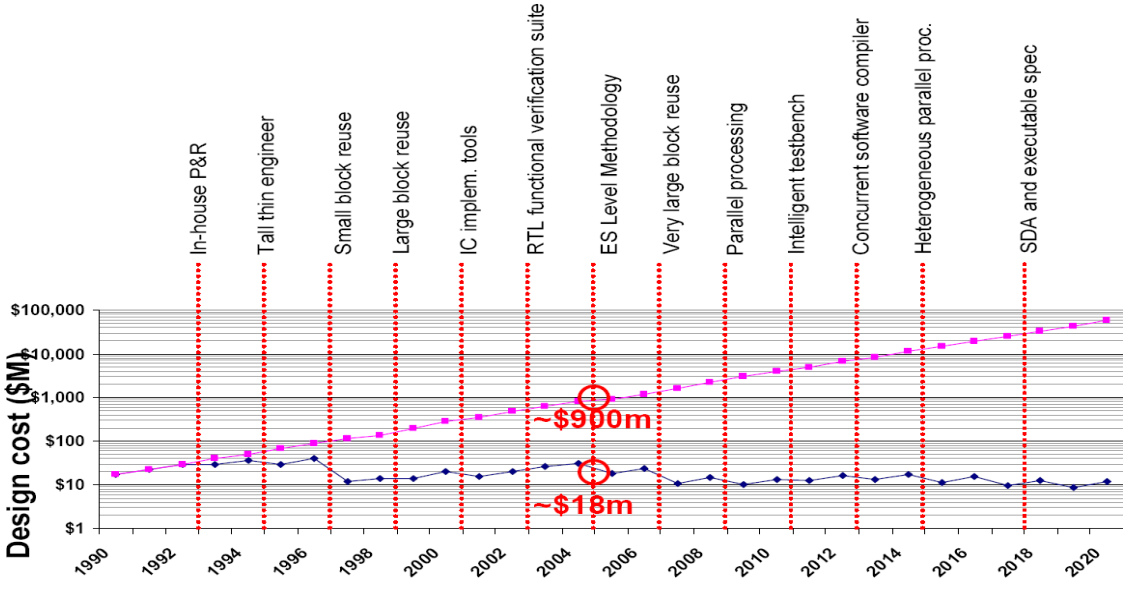


Figure 1.3. *Productivité des concepteurs (Nombre de Portes/Concepteur/An)*

Ainsi, sur la Figure 1.3 (tirée de l'International Technology Roadmap For Semiconductors-*ITRS*-Design, *2005*), les auteurs ont corrélé l'évolution des méthodologies de conception et leur impact sur

le coût de conception des circuits. On constate que les méthodologies dites E.S.L. (pour *Electronic System Level*) et les méthodologies de réutilisation de blocs matériels complexes constituent à l'heure actuelle les problématiques les plus importantes pour l'industrie.
Nos travaux trouvent place dans ces domaines et s'adressent également aux techniques de calcul parallèle (cf Figure 1.3).

Une problématique majeure : la mémoire
L'un des enjeux majeur des années à venir pour les outils de CAO est la réduction du coût de la mémoire dans les systèmes. En effet, si l'évolution technologique permet d'accroître de façon exponentielle les performances des processeurs, les performances des éléments mémorisants sont loin de croître au même rythme (cf. Figure 1.4).

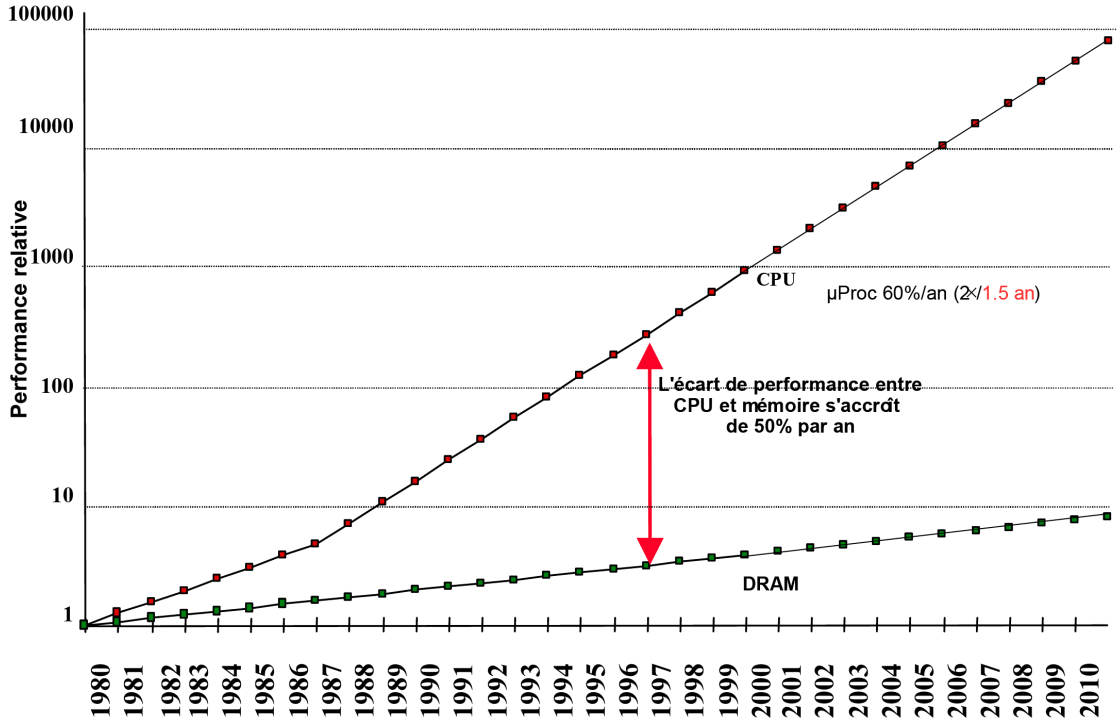


Figure 1.4. *Evolution de l'écart de performances (latence) entre CPU et mémoire*

Le goulet d'étranglement que représente la mémoire ne se limite pas aux performances. La prédominance des mémoires sur la surface et la consommation globale des systèmes va s'accentuer avec l'augmentation des quantités d'informations des nouvelles applications [HAN06]. Ainsi, l'ITRS prévoit que la mémoire occupera plus de 90% de la surface des systèmes sur puce, dans une dizaine d'années (cf. Figure 1.5). De même, la consommation en puissance des mémoires représente actuellement 10 à 15% de la consommation pour les architectures des processeurs et peut atteindre près de 50% de la consommation globale pour des applications de télécommunication ou multimédia [COR05].

Ce surcoût en termes d'éléments mémorisants se retrouve notamment au niveau des interfaces de communication, comme nous allons le voir par la suite. **Dans ce cadre, l'optimisation de ces interfaces peut se ramener à un problème d'optimisation de la mémoire et de l'architecture de contrôle et d'aiguillage associée.**

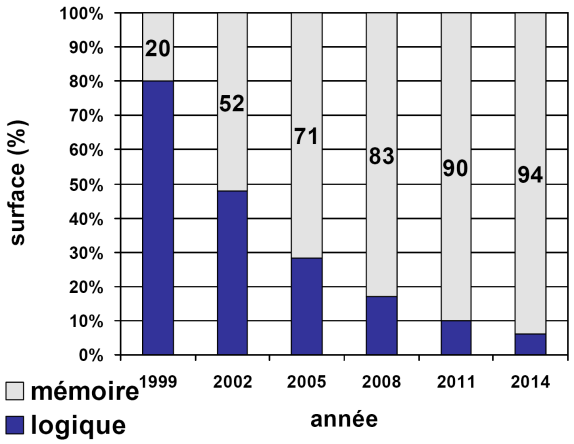


Figure 1.5. *Evolution de la surface des systèmes (ITRS)*

Pour répondre aux différents problèmes de performance, de surface et de consommation inhérents aux mémoires, une première approche peut consister à en pousser les performances : utiliser des mémoires de plus en plus denses, de plus en plus efficaces ou consommant moins. Les concepteurs se reposent alors sur les évolutions technologiques, pour réduire l'impact en termes de coût, de surface et de consommation de ces éléments (cf. Figure 1.6, tiré de l'ITRS 2005, Process Integration, Devices, And Structures).

Year in Production	*2014*	*2015*	*2016*	*2017*	*2018*	*2019*	*2020*
DRAM ½ Pitch (nm) [1]	*28*	*25*	*22*	*20*	*18*	*16*	*14*
DRAM cell size (μm^2) [2]	0.0048	0.0038	0.0030	0.0024	0.0019	0.0015	0.0012
DRAM storage node cell dielectric: equivalent physical thickness EOT (nm) [3]	0.45	0.4	0.4	0.3	0.25	0.2	0.15
DRAM storage node capacitor voltage (V) [4]	1	0.9	0.9	0.7	0.7	0.7	0.7
Electric field of capacitor dielectric, (MV/cm) [5]	22	23	23	23	28	35	47
DRAM cell FET dielectric: equivalent oxide thickness, EOT (nm) [6]	4	3.5	3.5	3.5	3	3	3
Maximum Wordline (WL) level (V)1 [7]	2.6	2.3	2.3	2.3	2	2	2
Electric field of cell FET device dielectric (MV/cm) [8]	6.5	6.6	6.6	6.6	6.7	6.7	6.7
Cell Size Factor: a [9]	6	6	6	6	6	6	6
Array Area Efficiency [10]	0.56	0.56	0.56	0.56	0.56	0.56	0.56
Minimum DRAM retention time (ms) [11]	64	64	64	64	64	64	64
DRAM soft error rate (fits) [12]	1000	1000	1000	1000	1000	1000	1000

Figure 1.6. *Evolution à long terme des DRAM (ITRS 2005)*

Toutefois, on notera qu'il ne s'agit là que de ***solutions technologiques***. Une analyse plus poussée des communications peut permettre de proposer des ***solutions architecturales*** et ***méthodologiques*** au problème de l'adaptation des communications.

Dans [CON06c], les auteurs s'inspirent des approches systoliques ([KUN82]) pour résoudre les problèmes de communication. Le principe est de régulariser l'architecture pour pouvoir ensuite exploiter des accès mémoires réguliers.
Nous nous proposons d'exploiter une ***analyse des communications à grain fin*** (données de niveau scalaire) afin d'extraire des schémas sémantiques connus des échanges de données et de les exploiter ([BAG98], [ZIS02], [CHE04]).

Evolution des applications
L'augmentation des densités d'intégration des transistors dans les circuits permet aux concepteurs d'intégrer des fonctionnalités de plus en plus complexes (cf Loi de Shannon, Figure.1.1). Ainsi, plus la technologie évolue, plus ils utilisent de transistors pour les nouveaux circuits. Inversement, plus les concepteurs ont besoin de transistors et plus les technologies sont appelées à évoluer.
Enfin, non seulement les applications sont de plus en plus complexes, mais les desideratas des utilisateurs sont tels que les concepteurs doivent intégrer sur une même puce des applications de plus en plus hétérogènes. On peut par exemple citer l'architecture de type *Set-Top Box* présentée dans [DUC07]. Ainsi un même circuit doit pouvoir gérer plusieurs protocoles de communication (Wifi, Edge, WCDMA, 3G…), plusieurs normes d'encodage/décodage audio et vidéo (MPEG, H264, JPEG2000, OGG, MP4, AAC…).

La difficulté pour les concepteurs de ces nouveaux circuits repose donc sur deux aspects : des applications à intégrer de plus en plus complexes et des applications de plus en plus hétérogènes (standards différents, normes d'encodage différentes…). Les réponses à ces évolutions se trouvent conjointement dans deux domaines distincts : (1) l'évolution des architectures ciblées vers des systèmes permettant de réutiliser rapidement des blocs préconçus autour de systèmes d'interconnexion fixés et (2) l'évolution des méthodologies et des flots de conception pour concevoir l'application à des niveaux d'abstraction de plus en plus élevés (cf. Figure 1.3).

2. Evolution des architectures ciblées – Notion de système sur puce

Le concept de système sur puce (SoC, pour *System On Chip*) découle des progrès technologiques des puces électroniques. La réduction de la taille des transistors permet en effet une concentration supérieure de transistors sur une même surface de silicium. Cette concentration autorise désormais la présence de plusieurs composants de natures différentes sur une unique puce.

Une carte électronique autrefois composée de plusieurs composants discrets (processeur, mémoires, ASIC...), se voit désormais dotée d'un composant unique intégrant tous les autres. C'est pourquoi le SoC est généralement présenté comme un ensemble de blocs hétérogènes présents sur la même surface de silicium [SOW04], [TAG00]. Ainsi les SoCs intègrent sur une même puce des processeurs (RISC,

DSP…) qui exécutent du code embarqué et qui offrent au système de la flexibilité, des blocs mémoires, des contrôleurs mémoires et des accélérateurs matériels dédiés.

D'autres architectures intègrent des composants analogiques, des capteurs, des microactionneurs, antennes, MEMs… : on parle alors de *System-in-Package* (SiP), [CAF05]. Enfin, on trouve également des FPGAs (pour Field Programmable Gate Arrays, [BRO96]) embarqués (eFPGA) dans ce type de systèmes sur puce [EFP07].

2.1. Les systèmes sur puce à base de bus

Pour qu'un tel système composé de différents blocs (cf. Figure 1.7) soit fonctionnel, il est nécessaire de disposer d'une architecture de communication adaptée et adaptable. Pour faciliter l'intégration des différents blocs de traitement, le monde de la microélectronique a vu apparaître des plateformes de communication de plus en plus élaborées. Les systèmes sur puce actuels sont principalement construits autour de bus de communication qui imposent leurs protocoles aux blocs qui leurs sont connectés.

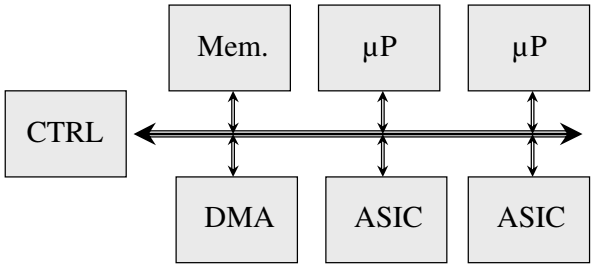


Figure 1.7. *Représentation schématique d'un Système mono-puce à base de bus*

Cette solution repose sur un mécanisme d'arbitrage qui malheureusement peut devenir un goulet d'étranglement, lorsque le nombre de composants accédant au média augmente. Un grand nombre de bus, dont les principales propriétés sont présentées dans [COU02b], est disponible sur le marché ; on peut ainsi citer CoreConnect [IBM07] d'IBM, STBus [WOD03] de STMicroelectronics ou AMBA [ARM07] de chez ARM.

La Figure 1.8 montre une architecture de type système sur puce à base de bus pour une *Set-Top Box* [DUC07]. Ce circuit intègre des applications très diverses (Décodeur vidéo multistandards, interfaces audio, décodeur audio multistandards, encryptage des données, ethernet…) autour d'un bus de communication *STBus*.

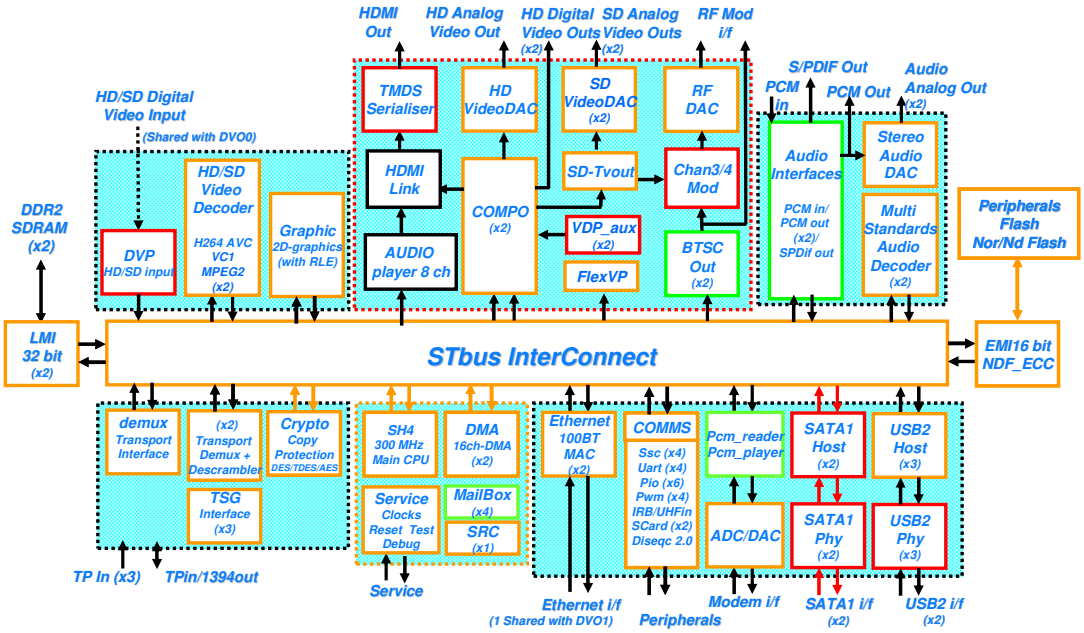


Figure 1.8. *Exemple d'architecture réelle de type systèmes sur puce à base de bus pour une Set Top Box*

Toutefois, l'inconvénient principal est qu'un système est construit autour d'un bus de communication centralisateur. Ainsi, lorsqu'un bloc utilise le bus pour transmettre des données, les autres ressources n'y ont plus accès et doivent donc attendre leur tour. Le concepteur doit alors assurer les contraintes de débit en utilisant des politiques d'accès au bus clairement définies.

2.2. Les systèmes dit de réseau sur puce

Ces dernières années, un nouveau type d'architecture d'interconnexion est apparu, visant à passer outre les limitations des systèmes à base de bus. Cette solution exploite une architecture de communication construite autour d'un réseau sur puce ou NoC (pour *Network-on-Chip*) [ART07], [JAN03], [CAR02]. Leur modèle de communications distribuées tente de répondre aux contraintes de débit, de latence, de flexibilité et de réactivité qui devront être mises en œuvre dans les futures applications.

Ce type d'architecture permet de réaliser plusieurs interconnexions distinctes de différents blocs de calcul en même temps. Dans [GRE06] les auteurs proposent une architecture basée sur un NoC interconnectant des îlots de calcul, composés eux même de système sur puce à base de bus (cf. Figure 1.9). Le réseau est constitué de routeurs [CHA04], [EVA04], qui transfèrent les données vers les ressources de calcul ou bien vers d'autres routeurs.

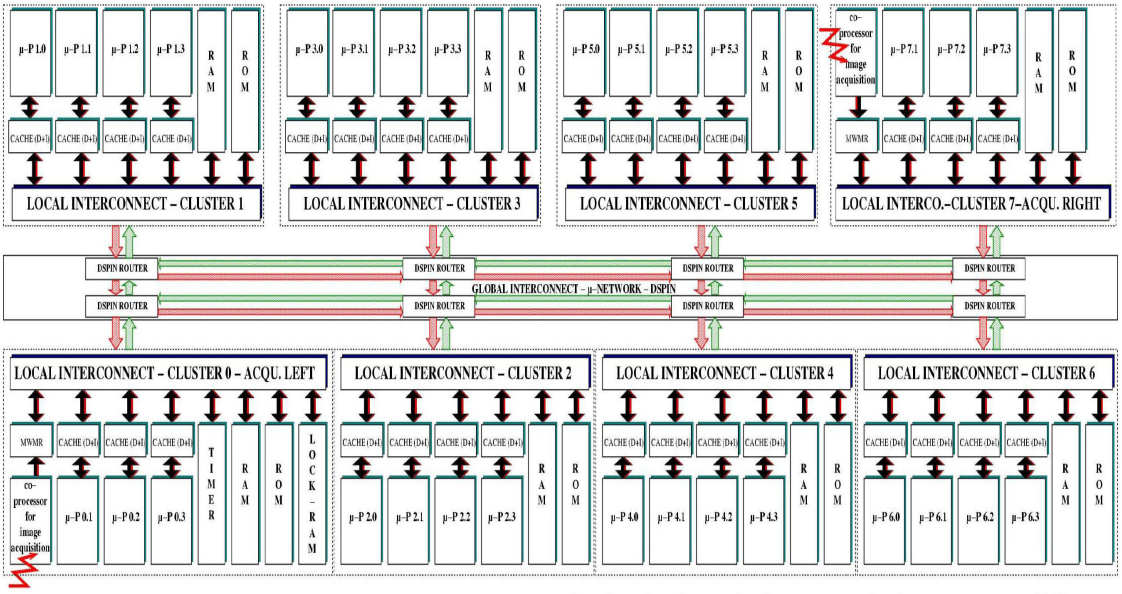


Figure 1.9. *Architecture NoC à base de routeurs DSPIN et de 8 îlots de calcul (clusters) [GRE06]*

Ce type d'architecture peut se généraliser comme indiqué Figure 1.10. L'application est ainsi construite autour d'un réseau sur puce interconnectant entre eux des îlots, au sein desquels on peut trouver des éléments de calculs de type accélérateurs matériels (architectures dédiées) ou bien des systèmes plus complexes exploitant des architectures de type "système construit autour d'un bus".

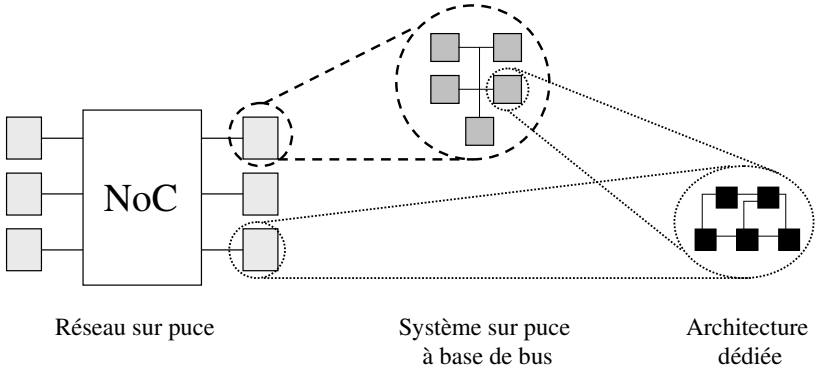


Figure 1.10. *Décomposition en niveau d'un système mono-puce complexe.*

Toutefois, le coût d'un tel système est très élevé et ne peut se concevoir que pour des architectures très complexes. Ainsi si l'on s'intéresse à une architecture comme celle présentée dans [GRE06], son coût en termes d'éléments mémorisants est très important, notamment à cause des nombreuses FIFOs à écriture/lecture multiple *MWMR* (pour *Multiple Write / Multiple Read*). Bien que l'utilisation de ce type de solutions commence à apparaître dans le monde industriel, elles restent principalement utilisées dans le domaine de la recherche.

Enfin, la complexité croissante des algorithmes implémentés, et l'augmentation continue des volumes de données et des débits applicatifs, requièrent souvent la conception d'accélérateurs matériels dédiés.

Typiquement l'architecture d'un tel composant requiert des éléments de calcul de plus en plus complexes, des mémoires et des modules de brassage de données (entrelaceur/désentrelaceur pour les TurboCodes, blocs de redondance spatio-temporelle dans les systèmes OFDM/MIMO…) et privilégie des connexions points à points (cf. Architecture dédiée de la Figure 1.10) pour la communication inter éléments de calcul. La synchronisation et la communication de données entre ces composants requièrent souvent la conception d'interfaces spécifiques (adaptation des contraintes temporelles, des séquences de transfert, de la synchronisation …) car ces composants n'ont pas nécessairement été pensés et conçus pour communiquer et échanger des données entre eux.

La temporisation, le réordonnancement et l'aiguillage des échanges de données constituent ainsi une problématique non négligeable pour la minimisation du coût du système en termes d'éléments mémorisants et de contrôle [CHE06], [HAN06].

2.3. Flexibilité et logiciel embarqué

L'utilisation des architectures de type SoC permet au concepteur de réaliser une partie de son application en logiciel, qui sera alors exécuté par les processeurs embarqués. Les temps de conception (développement, validation) sont alors grandement réduits. De plus, la possibilité de changer le logiciel embarqué permet d'envisager de réutiliser les blocs matériels pour d'autres applications, simplement en modifiant le code embarqué. La possibilité d'exécuter du code embarqué offre donc une plus grande flexibilité pour développer une application, par rapport aux systèmes tout silicium.

Toutefois, comme le montre [PAU07], l'essentiel des performances d'un système provient des accélérateurs matériels qui le composent (cf. Figure 1.11.a). En effet, si près de 95% d'une application peut-être implémenté en logiciel, l'essentiel des performances du circuit (80%) provient des accélérateurs matériels qui le composent. De même, ces accélérateurs sont beaucoup plus efficaces si l'on compare le rapport entre la surface qu'ils occupent et la fréquence à laquelle ils doivent fonctionner (cf. Figure 1.11.b).

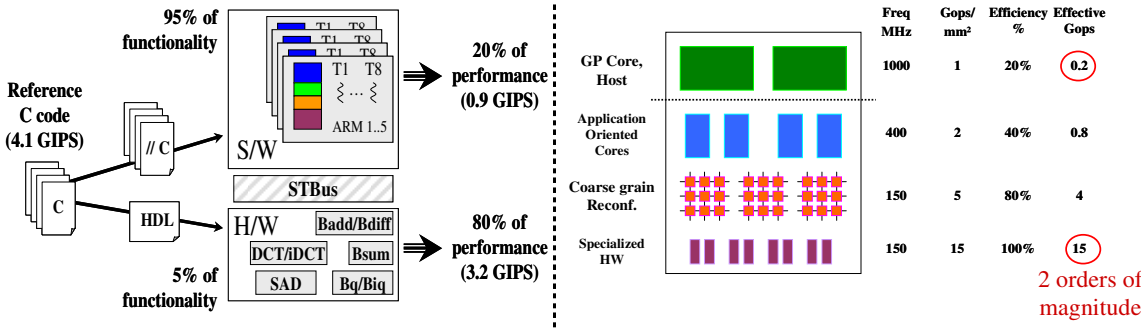


Figure 1.11. *Comparaison de performances pour un Codec MPEG4 (VGA, 30fps)*

Le concepteur doit donc étudier avec le plus grand soin le partitionnement de l'application en deux classes de fonctions :

- Les fonctions qui peuvent être implémentées en logiciel sans dégrader les performances (débit, consommation…) du circuit.
- Les fonctions qui doivent être implémentées en matériel pour garantir des performances en termes de débit, de consommation…

Ainsi, le concepteur a besoin de méthodologies de conception et d'outils pour le guider dans les choix qu'il doit faire durant cette phase d'exploration architecturale consacrée au partitionnement logiciel/matériel.

3. Evolution des méthodologies de conception

L'évolution des méthodologies de conception a principalement consisté en une élévation du niveau d'abstraction des descriptions. Ceci a permit aux concepteurs d'exploiter leurs modèles de haut niveau, alors utilisés pour valider des principes algorithmiques, directement en entrée des outils de conception. L'objectif est de pouvoir mettre au point les applications plus rapidement à l'aide de spécifications, de modélisations et de langages de haut niveau. Une fois le système validé, le concepteur a alors besoin de méthodologies et d'outils lui permettant de raffiner cette description abstraite en une implémentation. La Figure 1.12 présente un flot de conception classique.

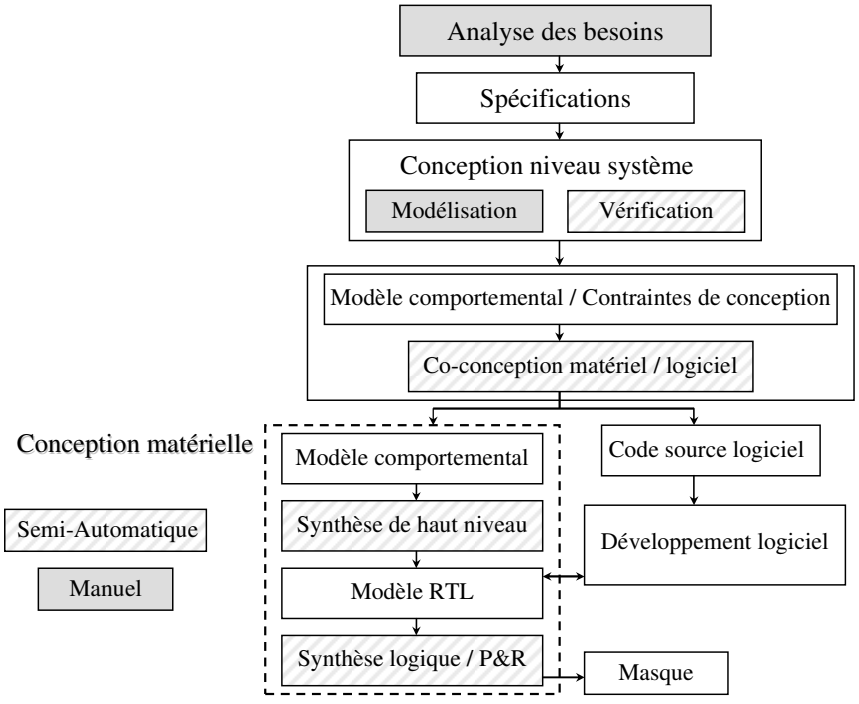


Figure 1.12. *Flot de conception*

Les flots en question doivent permettre d'explorer l'espace des solutions architecturales, de réaliser le partitionnement, de réutiliser des blocs de plus en plus complexes et enfin ils doivent également permettre la synthèse des parties logicielles et matérielles et la validation formelle des architectures obtenues.

3.1. Spécification et raffinement

L'application fait en premier lieu l'objet d'une modélisation à haut niveau (MatLab, C, C++, SystemC, Esterel….), appelée spécification fonctionnelle, avant d'être raffinée. Grâce aux vitesses de simulation importantes qu'ils offrent, ces premiers modèles sont utilisés pour déterminer rapidement un découpage fonctionnel du système et pour tester les algorithmes retenus.
Des outils comme [COF07], [COW07] ont étés proposés pour permettre à l'utilisateur de réaliser un premier découpage fonctionnel de l'application. Une fois le modèle de haut niveau validé, celui-ci pourra alors être raffiné en descriptions de plus en plus proches de l'implémentation finale.

Récemment, des flots exploitant des modèles de raffinement de type TLM (pour *Transaction Level Model*) ont étés proposés, [CAI03], [ZHE03]. Cette approche permet par exemple de proposer une modélisation des parties matérielles du système très en avance de phase, pour pouvoir démarrer le plus tôt possible le développement des parties logicielles embarquées dans le système [CLO03].
D'autres travaux proposent d'utiliser les modèles TLM pour explorer et raffiner l'architecture des coprocesseurs à générer. C'est notamment le cas de [THA06], dont les auteurs proposent une méthodologie de raffinement des communications en se basant sur des modèles abstraits originaux.
Dans [VIA06] les auteurs présentent un modèle TLM avec du temps (TLM/T, pour *TLM with Time*) qui permet de déterminer si une architecture respecte, ou non, des contraintes temps réel. Ceci suppose une prise en compte du comportement dynamique de l'application, des interconnexions et des accès mémoire. Ce type de modèle est utilisé dans le cadre du projet SoCLib [SOC07], qui a pour but de créer une librairie libre de modèles de simulation inter-opérables de coeur d'IP à destination des industriels et des laboratoires de recherche.
Un aperçu des flots et des approches basées sur les TLM est présenté dans [DON04].

3.2. Conception conjointe

Le concept de Conception Conjointe Matériel/Logiciel (ou Hw/Sw Co-Design) propose de développer en parallèle les parties matérielles et logicielles d'une application [DEM02], [SAN04], [DUM02], [MOU05].

Les méthodologies et outils de co-design logiciel/matériel réalisent en premier lieu une étape d'exploration architecturale afin de guider le concepteur vers une implantation efficace de chacune des fonctions composant l'application considérée (cf. Figure 1.13). La phase d'exploration architecturale est basée sur le calcul de fonctions de coût (parallélisme, taux d'accès à la mémoire, contraintes de communication…). En fonction de ces métriques et de bibliothèques d'estimation, l'outil va générer ou proposer un ensemble de solutions d'implémentation pour les différents blocs fonctionnels.

Les différents modules, ainsi que leurs interfaces de communication, sont ensuite implémentés en fonction des décisions prises durant l'étape de partitionnement. L'implémentation des parties logicielles et matérielles peut être réalisée manuellement (développement de code ASM ou RTL) ou à l'aide d'outils adaptés (compilateurs ou outils de synthèse).

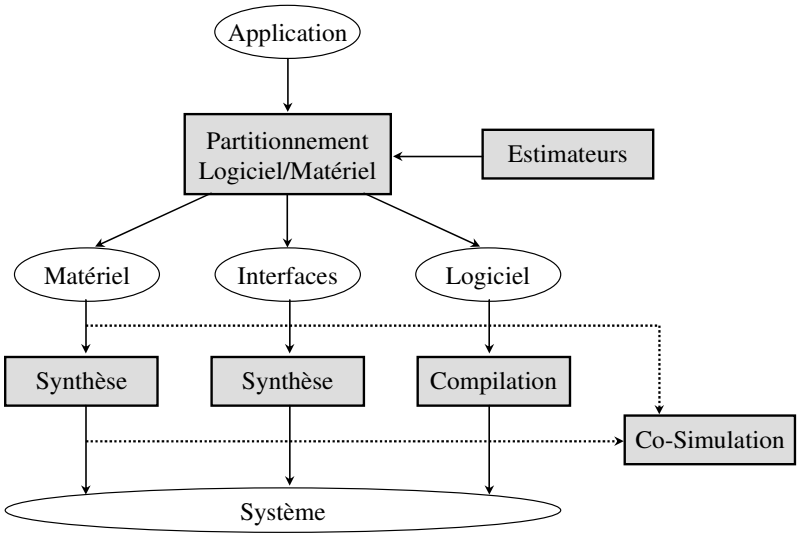


Figure 1.13. *Flot de conception conjointe logiciel/matériel*

Concernant l'implémentation des parties matérielles, la réutilisation de blocs matériels préconçus (ou IP, pour *Intellectual Property*) permet d'accélérer les temps de développement. Différentes solutions pour réaliser les communications entre parties logicielles et matérielles ont étés proposées, on peut ainsi citer les travaux de [HOM01a].

Dans [GRE06], les auteurs utilisent un ensemble de FIFO (*First In First Out*) acceptant des lectures/écritures multiples pour temporiser les données en provenance du réseau, avant leur utilisation dans les îlots de calculs. Dans [OBE99], l'auteur propose une grammaire permettant de modéliser les comportements aux interfaces de communication des composants, dans le but de proposer par la suite des outils logiciels capables de synthétiser automatiquement les interfaces de communication. Dans [CES01] ou encore [FRA04] les auteurs proposent des solutions permettant la génération d'adaptateurs pour l'intégration de composant IP.

La dernière étape consiste à valider fonctionnellement le système obtenu, même si des validations fonctionnelles par (co-)simulation sont pratiquées durant l'ensemble du processus de raffinement du système. En effet, le temps de simulation d'un circuit numérique dépend principalement du niveau d'abstraction de sa description. S'il faut quelques secondes pour simuler la description d'un système avant synthèse/compilation des composants, plusieurs jours peuvent être nécessaires lorsque le système complet est décrit à bas niveau (modèles RTL et ASM).

3.3. Concept de composants virtuels

Cette section présente brièvement le concept de composant virtuel aussi nommé IP pour *"Intellectual Property"*.

Notion de composants virtuels

La réutilisation de composants préconçus est un concept bien connu dans le domaine de la conception de logiciels. Cette technique s'applique depuis quelques années à la conception des systèmes sur silicium : plutôt que de concevoir intégralement toutes les fonctions d'un système, il peut s'avérer intéressant d'utiliser (ou plutôt de réutiliser) des fonctions déjà développées en interne ou par une tierce partie. Ainsi à l'origine, la réutilisation a débuté au niveau portes logiques (assemblage de transistors), puis a évolué vers l'utilisation de macro-fonctions (assemblage de portes logiques) pour finalement aboutir à la réutilisation des composants RTL pouvant aller jusqu'à plusieurs centaines de milliers de transistors. Actuellement, les composants réutilisés au cœur des systèmes sont majoritairement les processeurs généraux (CPU) et spécialisés (DSP).

Les règles de standardisation classent les composants virtuels matériels en trois familles suivant leur niveau d'abstraction:

- *HARD IP (ou IP matériel)* : le composant est délivré au niveau physique (Netlist, Layout personnalisé). L'inconvénient de ce type de composant très optimisé est qu'il n'est pas flexible, ni portable.
- *SOFT IP (ou IP logiciel)* : le composant est livré sous sa forme HDL synthétisable. Ce type de composant est très flexible, mais il ne peut être très prédictif (superficie, puissance et temps).
- *FIRM IP (ou IP flexible)* : le composant est délivré sous forme de Netlist qu'il est possible d'optimiser. Le composant virtuel FIRM est un compromis entre les composants SOFT et HARD.

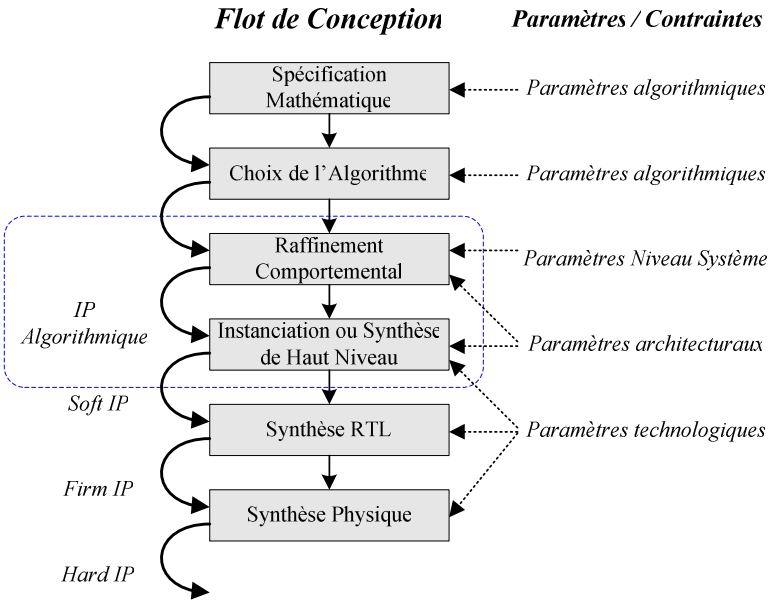


Figure 1.14. *Flot de synthèse d'une IP algorithmique*

A cette brève description, nous pouvons également adjoindre la notion de composant virtuel algorithmique (cf. Figure 1.14). Il s'agit dans ce cas d'une description à très haut niveau (C, SystemC) de l'IP. Ce type de composant peut ensuite être utilisé dans des flots de synthèse de haut niveau. Pour plus de détails sur les IPs, le lecteur intéressé pourra se reporter à [SAV02].

Afin de limiter les problèmes d'interconnexion/d'intégration des composants virtuels, le consortium VSIA (pour Virtual Socket Interface Alliance) a dans un premier temps proposé de définir une interface standard : cette solution s'est avérée trop complexe et n'a pas été retenue [CAT97]. Le groupe de travail s'est donc orienté vers la spécification de protocoles standards facilitant la réutilisation et l'intégration des composants virtuels. Ainsi, s'appuyant sur le principe de séparation du traitement et de la communication, une interface pour composant virtuel (VCI) [OCB00] a été définie. Les composants virtuels ainsi fournis s'adaptent à tous types de média (e.g. à l'aide d'un adaptateur dans le cas des bus partagés).

Intégration des composants virtuels

On notera également que l'utilisation de protocoles de communication standardisés n'exempte pas le concepteur de devoir réaliser une adaptation supplémentaire de la communication, avant de pouvoir traiter les données. En effet, si l'IP attend les données dans un ordre différent que celui imposé par les protocoles de communication, alors avant de pouvoir traiter les données, il faut procéder à une étape de réordonnancement de celles-ci.

Il n'existe pas à l'heure actuelle de système totalement *plug-and-play*, car il n'existe pas pour l'instant de protocole de communication standard accepté et acceptable par tous, [SPI]. L'intégration de blocs de calcul suppose donc de pouvoir analyser leurs protocoles de communication, de vérifier leur compatibilité, d'adapter les interfaces de communication et l'ordre de transfert des données…

Dans l'optique d'une intégration au sein d'une architecture de type architecture dédiée (cf. Figure 1.10), l'interconnexion d'IPs n'étant pas originellement prévue pour fonctionner ensemble peut poser problème. Dans ce cas le concepteur se retrouve dans une situation où il doit mettre au point une interface de communication permettant de réaliser l'adaptation des communications. Ainsi, les méthodologies partant du principe qu'un composant virtuel peut être réutilisé sans ajout ou modification ne sont plus valides, en particulier dans les applications dominées par le traitement des données. En effet, de tels systèmes requièrent une évaluation et une prise en compte de l'organisation des échanges (ordre des échanges de données, parallélisme…) et du stockage des données, afin d'obtenir une solution d'implémentation viable. La réutilisation de composants incompatibles en termes d'entrée/sortie aboutit à une adaptation coûteuse en termes de latence, débit et surface (mémoire + contrôle) des transferts de données.

Nous illustrons ce problème par un exemple pédagogique décrit dans la Figure 1.15. Supposons que nous souhaitions réutiliser un élément de calcul conçu pour acquérir ses entrées ligne par ligne. Supposons que le contexte dans lequel ce composant doit être intégré n'est pas celui pour lequel il a été conçu : dans la nouvelle application, le système fournit les données au bloc de calcul en zig-zag.

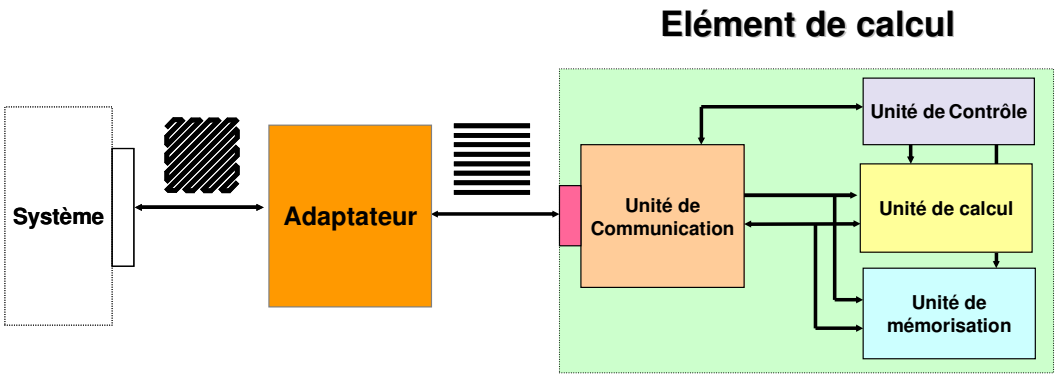


Figure 1.15. *Adaptation de la communication*

Ce cas de figure impose l'utilisation d'un ensemble de tampons adaptant les séquences d'émissions des données par le système vers l'élément de calcul, et d'un ensemble de tampons adaptant les séquences d'émissions de données depuis l'élément de calcul vers le système. L'adaptateur se compose donc d'éléments de mémorisation, de logique d'aiguillage et d'un contrôleur.

Classiquement, dans le cas où l'ordre d'arrivée des données dans le bloc généré est différent de celui dans lequel celles-ci sont sensées être traitées (cf. Figure 1.15), l'interface sera généralement composée de un ou plusieurs blocs mémoires RAM (en fonctionnement ping-pong) qui stockeront l'ensemble des données avant de lancer les traitements [MAL06], [KAT02]. Le lecteur intéressé trouvera une description de ces outils en annexe A.
L'adaptation des communications introduit donc un surcoût pouvant aboutir à la violation des contraintes imposées pour la conception du système. De plus, le coût de la mémoire devenant de plus en plus critique dans les systèmes actuels, l'optimisation de ces composants est un problème central non trivial. Cette problématique de réduction du coût des mémoires (surface, consommation…) a également été abordée dans [CAT99]. Les auteurs estiment que le coût en terme de consommation serait dû pour 50 à 80 %, au stockage et aux transferts de données.

Dans [BAG98] ou [ZIS02] les auteurs proposent des approches permettant de générer des interfaces de communication optimisées. C'est travaux seront développés dans le chapitre suivant car l'approche que nous proposons peut être utilisée pour répondre à cette classe de problème.
Une autre solution peut être de modifier directement le bloc de calcul, comme proposé dans [VER02]. Toutefois cette méthodologie, bien qu'efficace, oblige le concepteur à réécrire en partie le code de la spécification initiale. De plus, cette approche ne permet pas l'intégration des composants virtuels dans le cadre général de l'adaptation de débits ou de protocoles, et ne fait aucune référence aux contraintes de communication telles que les modes de synchronisation. Enfin, une fois la spécification fonctionnelle modifiée, il faut encore utiliser un flot de synthèse de haut niveau pour générer l'architecture finale.
Dans [COU05] les auteurs proposent de synthétiser les IPs sous contraintes de communication. Ici, comme dans [VER02], l'idée est de rapprocher le comportement de l'IP, du comportement des entrées/sorties du système (i.e. de l'ordre dans lequel les données lui sont transmises et l'ordre dans

lequel les composants qui lui sont connectés en sortie les attendent). Les auteurs n'ont toutefois pas présenté de méthodologie permettant de générer l'unité de communication.

3.4. Synthèse de haut niveau

La problématique consiste ici à générer automatiquement des architectures dédiées aux applications, par exemple de traitement du signal et de l'image. Les architectures ciblées peuvent être distribuées et hétérogènes. Un flot de synthèse de haut niveau classique est présenté Figure 1.16.

Flot de synthèse de haut niveau

La première étape d'un flot de synthèse de haut niveau est l'étape de compilation. Cette étape réalise la vérification syntaxique et sémantique de la description algorithmique et la traduit en un format intermédiaire propre à l'environnement de synthèse. La phase de compilation réalise de plus les opérations telles que : l'élimination du code mort, la propagation des expressions constantes, le déroulage de boucles, la mise en ligne des fonctions [BAC94]… Deux types de modèles sont couramment employés dans les outils de synthèse pour la représentation intermédiaire : les graphes flot de données *(Data Flow Graph DFG)* [GAJ92] et les graphes flot de données et de contrôle *(Control and Data Flow Graph CDFG)* [ORA96]. Ces deux modèles seront présentés dans le chapitre III.

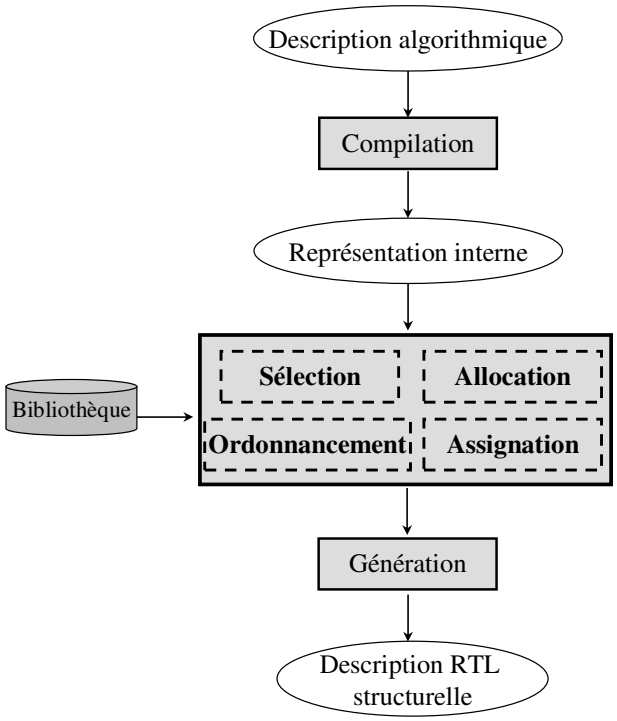


Figure 1.16. *Flot de conception type en synthèse de haut niveau*

Les différentes étapes effectuées par la suite opèrent sur le modèle de représentation interne :

- L'étape de ***sélection*** consiste à choisir la nature des ressources matérielles (opérateurs) qui réaliseront les opérations présentes dans l'application. Le choix des composants se fait sur des critères tels que leur surface, leur vitesse ou leur consommation.
- L'étape d'***allocation*** détermine, pour chaque type d'opérateur ou d'opération, le nombre de ressources à utiliser dans l'architecture finale.
- L'étape d'***ordonnancement*** a pour rôle d'affecter une date d'exécution à chacune des opérations en tenant compte d'une part des dépendances de données et d'autre part des contraintes imposées par le concepteur. Ainsi, l'ordonnancement peut chercher à : minimiser le nombre d'étapes de contrôle en fonction d'une quantité de ressources, minimiser le nombre de ressources en fonctions d'un nombre de cycles d'horloge...
- L'étape d'***assignation*** associe à chaque opération un opérateur matériel dans l'architecture.

De l'ordre dans lequel ces opérations sont effectuées, on peut déduire comment l'outil de synthèse de haut niveau fonctionne. Par exemple, si la synthèse est réalisée sous contraintes de latence, l'allocation sera alors réalisée après l'étape d'ordonnancement. Inversement, si la synthèse est réalisée sous contrainte de ressources, alors c'est l'ordonnancement qui est réalisé après l'étape d'allocation.

Les outils de synthèse de haut niveau

Les travaux menés autour de l'outil Syndex [GRA00] [KAO03] permettent de choisir une implantation respectant les contraintes temps réel et minimisant les ressources matérielles utilisées (Architectures multi-processeurs). L'implémentation sélectionnée est ensuite transformée en un exécutable temps réel distribué.

D'autres approches ciblent uniquement la génération d'accélérateurs matériels dédiés à la synthèse d'une application sous contraintes [GAJ91], [GAU07] ou [GUP04]. Ces méthodologies, basées sur l'utilisation de méthodes de raffinement automatique, produisent des solutions plus coûteuses que des architectures développées "*à la main*", mais les temps de conception s'en trouvent grandement réduits. De plus, ces techniques permettent le partage temporel et spatial des différentes ressources de calcul entre les opérations à effectuer, lorsque les contraintes temporelles le permettent. Ces flots de conception sont regroupés sous la dénomination d'outils de synthèse de haut niveau, ou HLS (*High Level Synthesis*).

D'autres travaux universitaires sont à mentionner dans ce domaine, ainsi les travaux concernant les outils Streamroller [MAL06] ou xPilot [CON06b] peuvent être cité. Ces travaux sont à l'origine de nombreuses solutions proposées par les fournisseurs d'outils logiciel, on peut ainsi citer les outils CatapultC [MEN07], Pico [KAT02], Cynthesizer [FOR07] ou encore Bluespec [BLU07] (cf. Annexe A).

Toutefois la plupart des outils disponibles ne proposent pas de méthodologie permettant d'optimiser les interfaces de communication des blocs de calcul qu'ils génèrent (à la différence de [COU05], [CHA07e] ou encore [ZIS02]).

3.5. Vérification Formelle

Les techniques de vérification formelle connaissent un engouement très important depuis quelques années dans l'industrie de la microélectronique. En effet, elles permettent de certifier rapidement et de façon certaine, qu'un circuit respecte un ensemble de propriétés. Les gains qu'elles apportent sont considérables si l'on considère que les erreurs sont détectées beaucoup plus tôt au cours du cycle de développement d'un circuit. Les allers-retours entre les équipes de conception et les équipes de tests ainsi que le nombre d'erreurs à tracer, sont fortement réduits. Les gains en terme de temps de conception sont donc conséquents.

Modélisation des propriétés

L'une des avancées majeures dans ce domaine a été la définition d'un langage permettant d'exprimer formellement les propriétés à vérifier : PSL (pour *Property Specification Language*) [PSL07]. Ce langage est basé sur les logiques temporelles (CTL, LTL) [QUE82],[CLA86], et s'inspire du langage Sugar d'IBM. Il a été approuvé et standardisé par le "*Formal Checking Tool Commitee"* d'Accelera en 2003 et par l'IEEE en 2004.

Les méthodologies de vérification formelle

Les outils de vérification formelle à proprement parler se classent en deux grandes catégories. On trouve en premier lieux des outils dits d'*Equivalence Checking* qui vont chercher à vérifier que deux descriptions ont bien le même comportement. C'est le cas par exemple des outils SLEC [CAL07] ou Formality [SYN07].

L'autre grande catégorie regroupe des outils qui vont chercher à valider (*Model Checking* : RuleBase [RBF03], FormalCheck [FCU99]) ou à prouver (*Theorem Proving* : ACL2 [KAU00], HOL [SLI00]) des propriétés formelles, qui peuvent être exprimées à l'aide du langage PSL, sur un design.

4. Problématique

Les méthodologies de conception font appel à des compétences dans des domaines très variés comme la vérification formelle, les méthodologies d'exploration et de raffinement ou encore la synthèse de haut niveau. Nos travaux trouvent leur place dans le cadre de flots de synthèse de haut niveau dédiés à la génération d'accélérateurs matériels. Nous avons constaté que la réduction du coût (surface, consommation) des éléments mémoires dans un circuit est un enjeu majeur et une problématique centrale pour les concepteurs.

Dans cette optique, nous proposons une approche permettant d'optimiser le coût de ces éléments mémorisants dans le cadre de l'adaptation de la communication entre éléments de calcul. Toutefois, comme nous allons le voir, cette problématique se retrouve dans plusieurs domaines.

4.1. Intégration de composants IP

La problématique de l'adaptation des communications pour l'intégration d'IPs revient à proposer une méthodologie et une architecture permettant de temporiser et de réordonner les données qui sont échangées, [BAG98], [ZIS02].

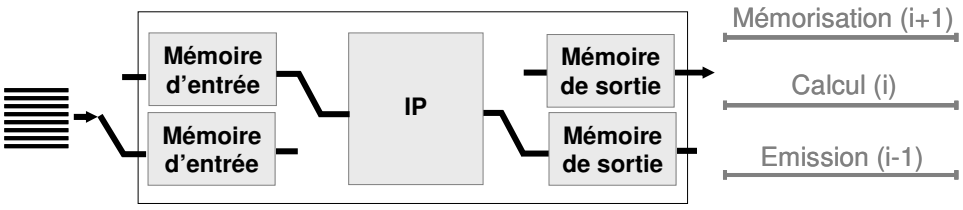


Figure 1.17. *Intégration d'IP : adaptation des communications à gros grain*

Les solutions classiques consistent à utiliser une architecture mémoire (RAM) reposant sur une analyse des communications à gros grain (e.g. matrice). Le principe est de mémoriser l'ensemble des données dans un bloc mémoire, dans l'ordre où elles arrivent. Puis, l'IP accèdera aux données dans l'ordre pour lequel il a été conçu. Pour assurer des performances élevées en termes de débit, la solution consiste alors à utiliser plusieurs blocs mémoire en mode «ping-pong» [XIL07], [MAL06] (cf. Figure 1.17).

Toutefois, il est possible d'optimiser l'architecture de mémorisation en analysant les communications à un grain de donnée plus fin, comme nous l'exposerons par la suite.

4.2. Synthèse de chemins de données

Les flots de synthèse de haut niveau reposent sur un ensemble d'étapes d'analyse et d'exploration (sélection, ordonnancement, assignation…) permettant de déterminer les différents éléments constitutifs sur un chemin de données (opérateurs, éléments mémorisants, multiplexeurs…). Ces éléments constitueront l'architecture finale de l'application (cf. Figure 1.18).

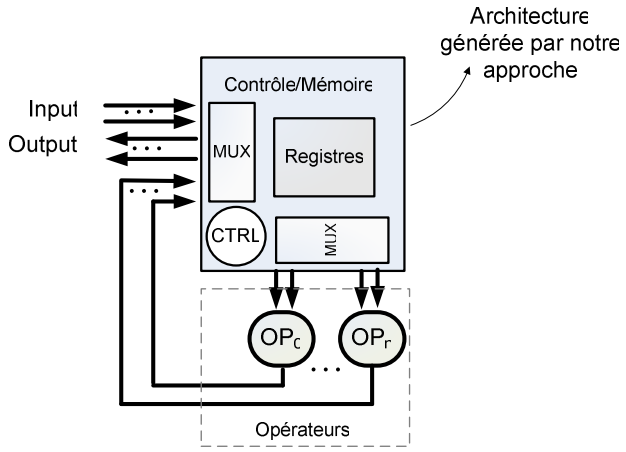


Figure 1.18. *Représentation classique d'un chemin de données*

Si l'on observe la partie mémorisation/contrôle d'un chemin de données (cf. Figure 1.18), on constate que cet ensemble a pour objectif de temporiser les données et de les (ré)orienter vers différents opérateurs. Nous sommes bien en présence d'une problématique de réordonnancement des données similaire à celle du réordonnancement des données dans le cadre de l'adaptation de la communication entre composants.

Il est à noter que dans le cadre de travaux menés en collaboration avec C. Andriamissaina (Doctorant LESTER), nous avons montré que la durée de vie des données dans le chemin de données (Durée de vies des registres associés) dépend directement du résultat de l'ordonnancement. La présence de multiplexeur dépend quand à elle du résultat de l'étape d'assignation et de l'étape de fusion des registres en fonction de leurs durées de vie [CHA07b].
Dans ce contexte, nous devons donc exploiter les résultats des étapes d'ordonnancement et d'allocation pour explorer et générer la partie Contrôle/Mémoire (cf. Figure 1.18) du chemin de donnée qui temporisera et distribuera les données vers différents ports. Nous sommes de nouveau en présence d'un problème d'optimisation d'une architecture de mémorisation et de contrôle (contrôleur et logique d'aiguillage).

4.3. Composant réalisant un brassage de données

Les algorithmes de décodage sont conçus pour corriger des erreurs dans une trame de données [BER93]. Toutefois, pour que ces corrections soient efficaces, il faut disperser ces erreurs sur toute la trame. Ainsi, dans le cas de chaînes de traitement du signal (cf. Figure 1.19), l'un des éléments les plus importants pour la qualité du décodage est le bloc d'entrelacement/dé-entrelacement. Sa fonction est de briser les relations de voisinage entre les différents éléments d'une trame de données. Ainsi, si une perturbation modifie plusieurs bits consécutifs de la trame entrelacée, une fois les données dé-entrelacées ces erreurs se retrouvent réparties sur toute la trame.

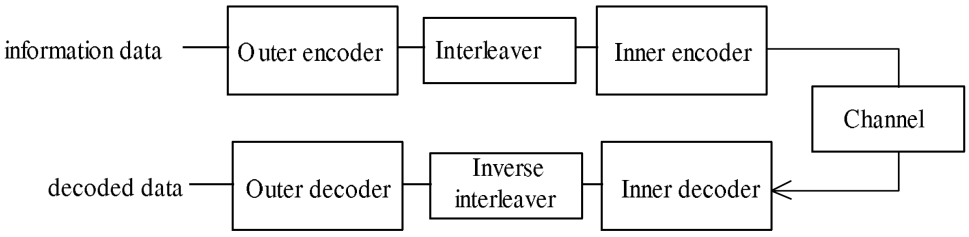


Figure 1.19. *Représentation simplifiée d'une chaîne de traitement du signal*

On notera que la fonction de l'entrelaceur consiste à mélanger des données : les données entrent dans un ordre O_A, et ressortent de l'entrelaceur dans un ordre O_B. Nous sommes donc bien en présence d'une problématique de réordonnancement des données, similaire à celle du réordonnancement des données dans le cadre de l'adaptation de la communication entre composants.

Le problème va consister à générer une architecture de mémorisation/temporisation des données et le contrôle associé. De nouveau, nous sommes en présence d'un problème d'optimisation d'une architecture de mémorisation et de contrôle (contrôleurs et logique d'aiguillage).

Bilan

Notre problématique consiste donc à proposer une méthodologie d'exploration et l'architecture associée, pour réaliser une interface de communication (temporisation, réordonnancement) entre deux composants. Cette interface est généralement constituée d'une architecture de mémorisation optimisée, basée sur des blocs mémoires dédiés, un réseau d'interconnexion et un contrôleur.

La méthodologie que nous proposons repose sur l'exploitation d'une modélisation formelle adaptée à notre problématique, et permettant de réaliser l'adaptation des communications notamment dans le cadre d'architectures intégrant plusieurs modes de fonctionnement. On parlera dans ce cas de systèmes ou d'architectures ***multi-modes***.

Chapitre 2

ETAT DE L'ART

Dans ce chapitre nous présentons un état de l'art des principales approches pour réaliser l'adaptation des communications entre éléments de calcul. Cet état de l'art adresse les trois domaines d'application présentés dans le chapitre précédent, à savoir : l'adaptation des échanges de données entre composants virtuels, les parties mémorisation et aiguillage des chemins de données et les composants de brassage de données.

1. Introduction

La problématique de l'adaptation des communications consiste à définir des architectures de communication évitant les congestions de transferts sur les média de communication et dans les tampons, tout en optimisant (1) le coût du stockage des données [HAN06] et (2) la complexité de la logique de multiplexage [CHE04].

Nous avons observé que les contraintes de coût et de performances imposent que les accélérateurs matériels soient conçus sur la base d'architecture point à point. Toutefois, cette solution suppose que le concepteur dispose de méthodologies lui permettant de réaliser l'adaptation des communications entre les différents blocs de calcul qui composent l'accélérateur. Notamment, si un élément de calcul est sensé traiter les données qui lui sont transmises dans un ordre précis, et que celui-ci diffère de l'ordre dans lequel les données lui sont fournies, alors il faudra nécessairement ajouter une architecture de réordonnancement des données au sein de l'accélérateur.

Les solutions classiques consistent à utiliser une architecture mémoire reposant sur une analyse des communications à gros grain (e.g. matrice). Le principe est de mémoriser l'ensemble des données dans un bloc mémoire, dans l'ordre où elles arrivent. Puis, l'élément de calcul accèdera aux données dans l'ordre pour lequel il a été conçu. Pour assurer des performances élevées en termes de débit, la solution consiste alors à utiliser plusieurs blocs mémoire en mode «ping-pong» [XIL07], [MAL06].
Cette approche permet de réaliser l'adaptation de la communication, mais elle entraîne un surcoût en termes d'élément mémorisant qui peut s'avérer très important dans le cas d'architectures à haut débit.
De même, devoir attendre que toutes les données aient été mémorisées avant de lancer le processus de calcul, diminue les performances de l'application en termes de débit, à moins de multiplier le nombre de blocs mémoire (cf. Chapitre I).

Un des objectifs de nos recherches est de réaliser cette adaptation de la communication entre composants matériels, en minimisant le surcoût dû à l'architecture générée. L'architecture que nous ciblons est basée sur une analyse à grain fin des communications. Notre approche repose sur des échanges de données déterministes sur des ordres partiels de transfert (i.e. l'ordre de transfert est déterministe, mais la séquence de transfert peut être non continue).
Nous proposons une méthodologie permettant d'explorer l'espace des solutions architecturales et de générer automatiquement l'adaptateur de communication. Celui-ci doit permettre de réaliser la temporisation et le réordonnancement des données.
Dans le cas d'architectures devant gérer plusieurs modes de fonctionnement différents (i.e. plusieurs modes de transfert des données), notre approche doit également permettre de générer une architecture optimisée intégrant ces différents modes de fonctionnement.

Nous avons observé que cette problématique de l'adaptation des comportements aux entrées/sorties des composants matériels peut se généraliser comme l'adaptation des échanges de données entre une source et une cible quelconques. De ce point de vue, il est alors possible de faire des analogies entre (1) l'adaptation des échanges de données entre IP, (2) la génération de chemin de données en synthèse de haut niveau (cf. Figure 2.1), et (3) la synthèse de composants de brassage (typiquement des entrelaceurs, [AND97] [GAR01]).

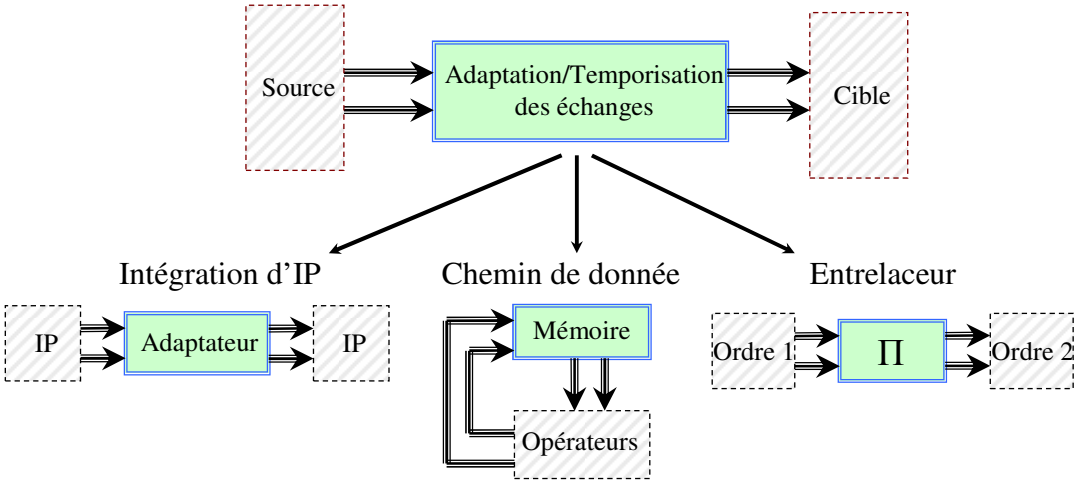


Figure 2.1 . *Adaptation de la communication : une problématique partagée*

Ainsi, dans ce chapitre, nous présentons :

- Des approches et méthodologies permettant l'adaptation de la communication dans le cadre de l'intégration de composants virtuels.
- Un état de l'art des méthodologies visant la génération de chemins de données.
- Un état de l'art des principales méthodologies à notre disposition pour la conception de composants d'entrelacement.

Nous concluons ce chapitre en réalisant un bilan des différentes approches.

2. Synthèse d'interface entre composant IPs / Intégration d'IPs

Les approches de synthèse des communications classiques visent, pour la plupart, l'adaptation de protocoles ou la synthèse d'interface logicielle/matérielle. Peu de travaux visent la synthèse d'adaptateur de communication dans l'espace et dans le temps, entre composants matériels. Parmi ces travaux, nous présentons dans cette partie un état de l'art représentatif du domaine.

2.1. Systèmes sur puce exploitant des interfaces de communication FIFO

Dans [FRA04] les auteurs présentent une méthodologie permettant d'interfacer des accélérateurs matériels avec un système sur puce. Plus particulièrement, ils s'intéressent à une catégorie d'accélérateurs dédiés au traitement intensif de données, basée sur des architectures de type systolique. Dans un premier mode de fonctionnement, les données sont stockées dans une RAM et un processeur est là pour piloter les différents éléments du système (cf. Figure 2.2.a) et gérer le transfert des données depuis ou vers l'accélérateur matériel (IP dans la Figure 2.2).

Dans un second mode (cf. Figure 2.2.b), les auteurs utilisent en plus un contrôleur DMA pour accélérer les transferts de données. Le DMA permet en effet de faire des échanges de données en mode rafale (*Burst*) ce qui a pour effet d'augmenter le débit des transferts de données.

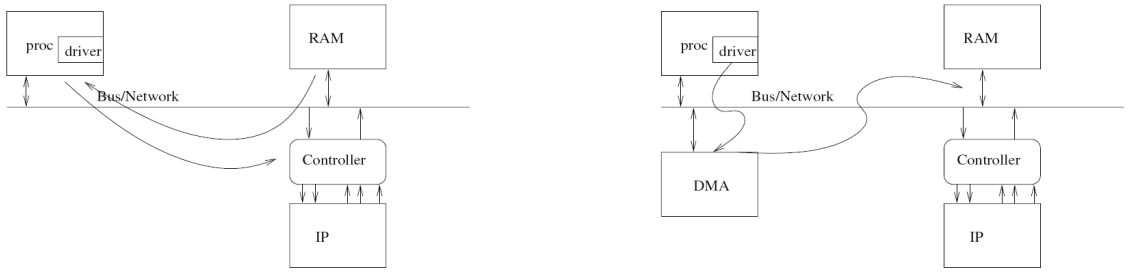


Figure 2.2 . *Modes de fonctionnement envisagés*

L'architecture de l'interface de communication proposée est présentée dans le Figure 2.3. Elle repose sur un ensemble de file (FIFO) pour temporiser les données transférées. Pour générer cette interface, l'approche proposée exploite une abstraction de la communication aux interfaces de l'IP et du système. Il s'agit d'une forme de diagrammes temporels dans lesquels les échanges de données sont spécifiés. A partir de l'analyse de ces informations, l'outil est capable de générer le contrôleur matériel et les pilotes logiciels. Différents types de paramètres doivent être fournis à l'outil comme le type de communication (avec ou sans DMA), les tailles des trames, le nombre de ports …

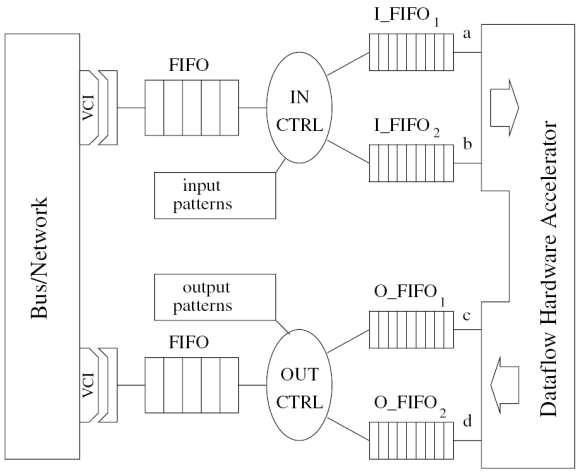


Figure 2.3 . *Architecture de l'interface à base de FIFO*

Toutefois, les auteurs ne proposent pas de méthodologie permettant de dimensionner automatiquement les tailles des FIFOs autrement que par simulation.

De plus, si les échanges de données ne respectent pas une sémantique FIFO, l'architecture retenue n'est plus valide. Dans ce cas, les auteurs admettent que leur approche ne permet pas de résoudre ce type de problème. Ils proposent plutôt de prendre en compte cette contrainte au niveau du processeur qui devra se charger de gérer la communication donnée par donnée, perdant ainsi le bénéfice apporté par le DMA.

Enfin, cette solution ne prend pas non plus en compte la problématique de la génération d'architectures multi-modes.

2.2. Adaptation des échanges de données : Extended Linearization Model

Ces travaux sont menés dans le cadre du développement de la suite logicielle *Compann/Laura* [KIE00]. Le principe est ici de linéariser un modèle d'échange de données entre des processus autonomes [ZIS02], [TUR02]. Ceux-ci communiquent en point à point au travers de FIFO non bornées (modèle de *Khan*), en respectant un mécanisme de lecture bloquante. Dans un tel réseau, chaque processus exécute une fonction interne en fonction d'un ordonnancement local. A chaque exécution, (ou itération) cette fonction lit/écrit des données depuis/vers différentes FIFO.

Dans l'outil *Compaan*, une des étapes impliquée dans la transformation d'un nœud de boucle en un réseau de processus de Khan, ou KPN (pour Khan Process Network), est la *linéarisation*. Le but de cette étape est de transformer une structure à *N* dimensions en un flux de données monodimensionnel. En règle générale, le modèle de linéarisation utilisé dans les KPN est un tampon FIFO (cf. Figure 2.4). Ceci permet au consommateur de lire les données dans l'ordre dans lequel le producteur les génère, comme si ces deux éléments étaient directement connectés l'un à l'autre. Toutefois, lorsque l'ordre de production des données diffère de leur ordre de consommation, une simple FIFO ne peut pas être utilisé comme modèle de linéarisation.

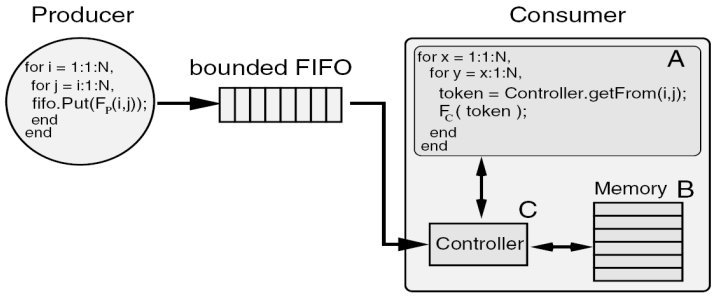


Figure 2.4 . *Extended Linearization Model*

Pour résoudre ce problème, les auteurs proposent un nouveau modèle de linéarisation : Extended Linearization Model (ELM). Un modèle ELM est constitué de deux éléments principaux : une mémoire de réordonnancement et un contrôleur. La Figure 2.4 est une représentation type d'un modèle ELM avec un processus consommateur (A), une mémoire de réordonnancement (B) et un contrôleur (C).

Pour implémenter le modèle ELM en matériel, les auteurs ont utilisé des mémoires CAMs (*Content Addressable Memory*). Les travaux présentés dans [TUR02] tendent en effet à montrer que l'utilisation d'une CAM permet de réduire la complexité du contrôleur de réordonnancement.
Une CAM diffère d'une RAM dans le sens où l'on utilise une table d'indirection pour accéder aux données. Ce principe est inspiré des systèmes de gestion des mémoires caches. Les performances d'une CAM dépendent fortement du temps nécessaire à retrouver une clef dans la table d'indirection. A chaque donnée échangée doit correspondre une clef unique.

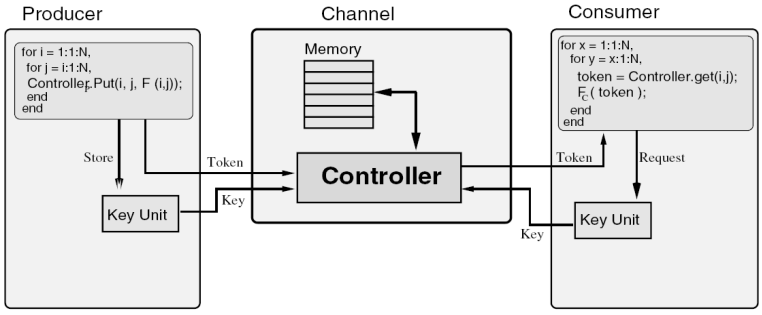


Figure 2.5 . *Principe de l'architecture retenue pour implémenter un ELM*

L'implémentation matérielle (cf. Figure 2.5) fonctionne ainsi: un producteur génère une donnée et une clef unique. Puis cette donnée est mémorisée dans le canal et peut être accédée à l'aide de la clef associée. Quand le consommateur veut accéder à cette donnée, son générateur de clef générera la clef correspondante. Cette information servira ensuite à retrouver la donnée par l'intermédiaire de la table d'indirection.

Un des points forts du modèle architectural retenu est qu'une seule mémoire est utilisée pour implémenter le canal FIFO et la mémoire de réordonnancement. Cette architecture se comporte comme une FIFO du point de vu du producteur, et comme une mémoire de réordonnancement du point de vue du consommateur, la sémantique du modèle de Khan est ainsi préservée.
Toutefois, cette sémantique est également un facteur limitant important de cette approche. En effet, dans un réseau de processus de Khan, un canal ne peut être relié qu'à un unique couple producteur/consommateur. Ainsi, si un processus consommateur peut être connecté à plusieurs producteurs, dans ce cas le modèle impose d'utiliser un canal unique par processus producteur. Donc, l'architecture résultante utilisera autant de canaux à base de CAM, alors qu'il serait peut être possible de mutualiser ces ressources.
De plus, la taille de la mémoire dans le canal n'est pas fixée. Pour éviter les phénomènes d'inter-blocages il convient donc de dimensionner ces CAMs avec soin. La solution proposée par les auteurs repose sur l'utilisation d'un simulateur de KPN pour fixer la taille de cette mémoire par simulation.
L'architecture des CAMs est quand à elle basée sur une table d'indirection, une RAM et des contrôleurs. Elle s'avère complexe et trop lourde pour être utilisable dans une architecture ASIC à un coût non prohibitif. Cette approche peut toutefois être avantageusement utilisée pour la modélisation sur FPGA. C'est là le domaine d'application retenu par les auteurs, ce qui leur permet d'exploiter les mémoires CAMs natives proposées sur les FPGA à leur disposition.
Enfin, la notion d'architecture multi-mode n'est pas évoquée dans cette approche.

2.3. Interface à base d'éléments mémorisants hétérogènes

Dans [BAG98] une approche formelle pour l'interfaçage de composants matériels est présentée. Pour ce faire, la solution repose sur une modélisation formelle des contraintes d'entrée/sortie (cf. Figure 2.6) A partir de ces contraintes, les auteurs construisent des graphes de contraintes appelés : Graphe de Compatibilité des Transferts (cf. Figure 2.7). Le problème revient alors à identifier et à dimensionner de façon optimale les éléments mémorisants.

Cette approche est basée sur une analyse des communications à grain fin (niveau scalaire) et l'architecture ciblée exploite des éléments mémorisants à sémantique forte (FIFO, LIFO et registres) pour implémenter le réordonnancement des données.

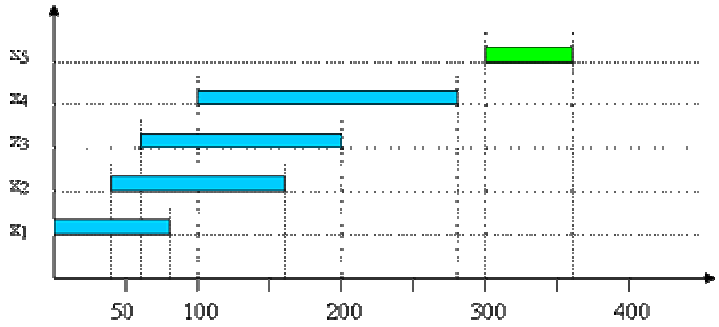
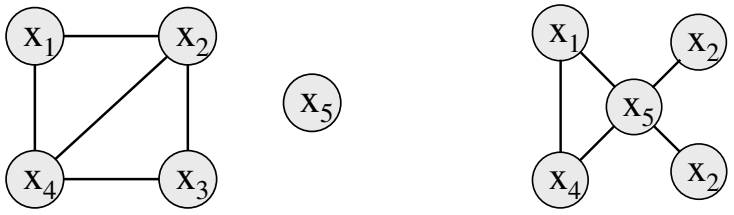


Figure 2.6 . *Exemple de contrainte d'entrée/sortie*

Le problème de dimensionnement est un problème connu. Il se rapporte à **un problème NP-complet de coloration de graphe**. L'identification des éléments FIFO ou LIFO sur le graphe, se fait de façon indirecte, en commençant par identifier des cliques de compatibilité maximale (problème NP-complet). Puis, les cliques identifiées sont alors analysées pour déterminer si l'on se trouve en présence d'un ensemble de données respectant entre elles une sémantique de type FIFO ou LIFO.



(a) (b)

Figure 2.7 . *(a) Graphe de compatibilité des transferts, (b) Graphe d'incompatibilité des transferts*

La recherche de la meilleure solution architecturale doit donc pouvoir prendre en compte ces deux aspects (identification et dimensionnement) en même temps. Pour résoudre ces problèmes, les auteurs proposent une approche heuristique (cf. Figure 2.8) qui consiste à identifier un maximum de FIFO sur les graphes, puis un maximum de LIFO et enfin les données restantes qui seront stockées dans des registres.

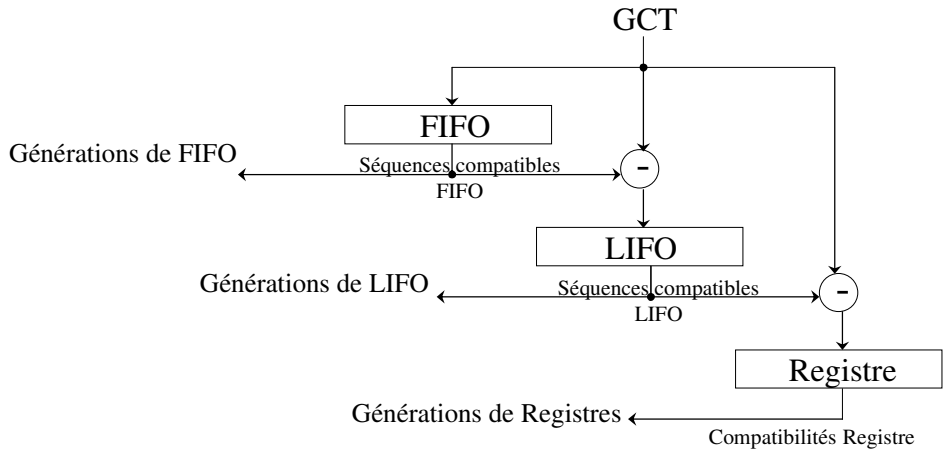


Figure 2.8 . *Heuristique d'exploration de l'espace des solutions architecturales*

Cet ordre de priorité résulte d'une hypothèse formulée par les auteurs : le coût d'une structure FIFO est toujours meilleur que le coût d'une structure LIFO et est toujours meilleur que le coût d'un ensemble de registres, pour mémoriser un ensemble de données.
Or, avec l'évolution de la technologie, cette hypothèse n'est plus valide dans le cas général. En effet, le coût relatif de ces structures (FIFO, LIFO, registre) est fortement corrélé au nombre et à la taille des données à mémoriser et à la bibliothèque technologique employée.
Les priorités définies par les auteurs constituent ainsi une heuristique dont la pertinence est dépendante du contexte de chaque application visée.
De plus, deux aspects fondamentaux ne sont pas abordés par les auteurs :

- Comment dimensionner les structures de mémorisation identifiées ?
- Comment générer le contrôleur et la logique d'aiguillage ?

Enfin, la problématique de la synthèse d'architecture de communication multi-modes n'est pas abordée dans ces travaux.

3. Synthèse de chemin de données

Nous l'avons évoquée précédemment, la problématique d'adaptation des communications entre éléments de calcul est très similaire aux problèmes d'optimisation de la mémorisation dans les chemins de données. Ainsi, on peut voir les opérateurs de calcul comme des blocs de traitement, et la mémoire et les multiplexeurs comme un adaptateur de communication devant temporiser et transférer les données d'un point à un autre.

3.1. Intégration de séquenceurs dans un chemin de données

Dans [ALO94], les auteurs proposent d'optimiser l'architecture de mémorisation du chemin de données en remplaçant les registres par des *séquenceurs* (cf. Figure 2.9). Ces séquenceurs (piles ou files) sont des éléments mémorisants à sémantique forte, et c'est en exploitant ces sémantiques que les auteurs se proposent d'implémenter la temporisation et les échanges de données.

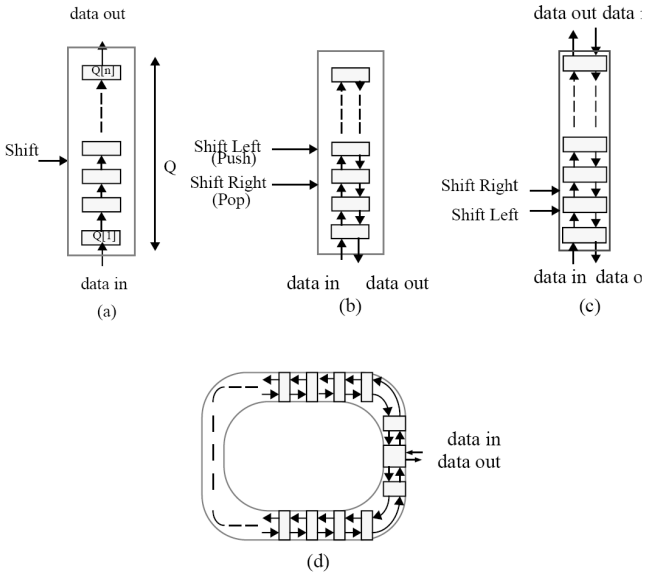


Figure 2.9 . *Différents modèles de séquenceurs*

Cette solution est rendue possible par le fait que les algorithmes de traitement du signal ont la caractéristique d'avoir un niveau de régularité très élevé. Il est alors possible de reconnaître ces trames régulières dans les échanges de données à partir de l'analyse des durées de vie des variables (cf. Figure 2.10.b). A partir de ces durées de vie, les auteurs construisent un graphe de compatibilité des données qu'ils vont ensuite explorer pour réaliser l'affectation des données dans les séquenceurs.

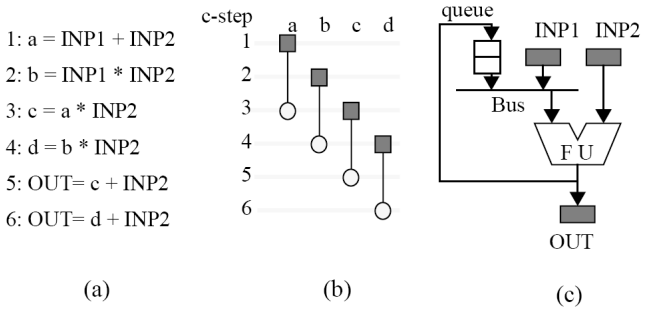


Figure 2.10. *Exemple de chemin de données utilisant une FIFO (queue)*

L'idée est donc d'exploiter au mieux cette régularité, en maintenant ces données dans le chemin de données au lieu de les transférer en RAM. Il est ainsi possible de réduire le coût global de l'architecture en limitant la taille et le nombre de blocs RAMs, ainsi que le contrôle associé à ces RAMs et les transferts de données entre RAMs et partie opérative.

Toutefois, l'assignation des données dans les séquenceurs utilise la même heuristique que l'approche proposée par [BAG98] et souffre donc des mêmes lacunes. De même, le dimensionnement des FIFOs n'est pas optimisé.

Enfin, la notion d'architectures reconfigurables n'est pas prise en compte.

3.2. Optimisation du multiplexage lors de l'assignation des registres

Dans [CHE04] les auteurs attirent notre attention sur un aspect peu évoqué dans les articles traitant de la génération de chemins de données : lorsque l'on souhaite optimiser le nombre de points mémoire, l'affectation des données dans les registres peut entraîner un surcoût en termes de composants de multiplexage (cf. Figure 2.11). Or, comme le montre les auteurs, le coût d'un multiplexeur complexe (8vers1, 16vers1…) devient rapidement équivalent, voir même supérieur au coût d'un opérateur ou d'un registre. Les auteurs proposent donc une approche permettant de prendre en compte cette contrainte lors de l'étape d'assignation des registres.

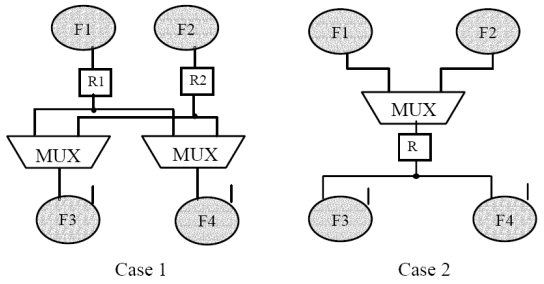
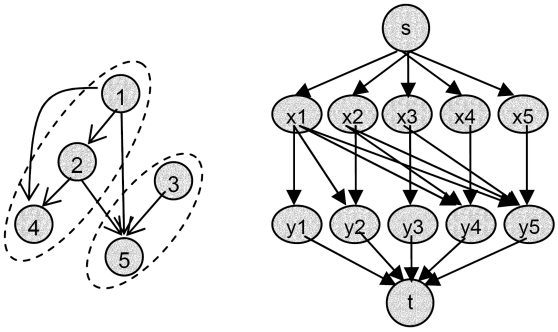


Figure 2.11. *Cas pouvant entraîner la création de multiplexeurs*

La solution proposée repose sur une modélisation à base d'un graphe de compatibilité $G_c = (V_c, A_c)$, construit à partir du graphe de flot de données de l'application.
Dans un tel graphe de compatibilité, V_c représente l'ensemble des variables du graphe de flot de données et A_c représente l'ensemble des arcs orientés du graphe. Il existe un arc orienté $a_c = (v_i, v_j)$ entre deux nœuds si et seulement si leurs durées de vie sont non recouvrantes. A chaque arc est associé un poids w_{ij} qui représente le coût relatif de la fusion des deux données dans un même registre.

Les auteurs exploitent alors la notion d'ensembles partiellement ordonnés (ou POSET, pour *Partially Ordered SET*) pour retrouver dans le graphe les variables qui constituent entre elles des chaînes. Dans un POSET *P*, une chaîne est un sous ensemble de *P* tel que toutes paires de variables de ce sous ensemble sont reliées les unes aux autres par un arc ou une séquence d'arcs. Les auteurs démontrent ensuite que l'assignation de *k* registres sur un graphe de compatibilité tel qu'ils le définissent, revient à trouver *k* chaînes disjointes dans un ensemble partiellement ordonné (cf. Figure 2.12.a).



(a) Exemple de POSET *Pc* (b) Graphe partagé *G(Pc)*

Figure 2.12. Modèle de graphes proposés

A partir de cette première analyse, les auteurs vont construire un graphe partagé (cf. Figure 2.12.b), nommé "*split graph*". Dans ce nouveau modèle, les auteurs représentent chaque nœud *i* du POSET originel par deux nœuds x_i et y_i dans le nouveau graphe. Les arcs sont étiquetés avec des poids :

- Les arcs reliant les nœuds source et puits aux nœuds x_i et y_i sont étiquetés avec le poids 0.
- Les arcs reliant un nœud x_i à un nœud y_i sont étiquetés avec la valeur -1.
- Les arcs reliant un nœud x_i à un nœud y_j sont étiquetés avec le poids de l'arc reliant ces deux nœuds dans le graphe de compatibilité.

Le problème d'assignation des registres sous contrainte de multiplexage consiste alors à trouver dans ce graphe partagé un flot de coût minimum.

Un autre aspect du problème concernant la réduction du multiplexage découle directement de l'assignation des données sur les ports des opérateurs. Si l'on considère les exemples présentés Figure 2.13, on observe que dans le premier cas, l'architecture utilise deux multiplexeurs à deux entrées. Dans le second cas, l'architecture utilise un multiplexeur à deux entrées et un autre à trois entrées.

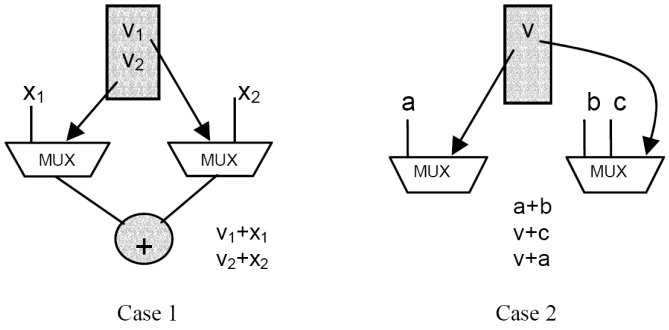


Figure 2.13. Exemples de registres connectés à de multiples ports

Toutefois, on constate (cf. Figure 2.14) qu'une assignation judicieuse des données sur les ports de l'opérateur permet dans le premier cas de supprimer un multiplexeur (un multiplexeur à une entrée n'a pas de sens), et dans l'autre de réduire la complexité du multiplexage en remplaçant un multiplexeur à trois entrées par un autre multiplexeur à deux entrées. Naturellement, pour que cette permutation soit possible, il faut que l'opération soit symétrique.

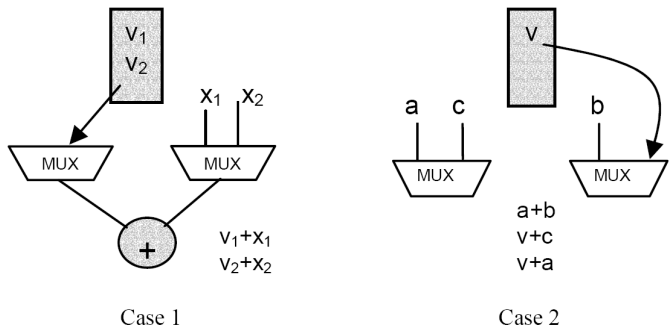


Figure 2.14. Modification de l'affectation des opérandes sur les ports

Le modèle proposé dans ces travaux permet de prendre en compte efficacement l'optimisation du multiplexage lors de la phase d'assignation des registres. Toutefois, ce modèle reste limité dans le sens où seules des architectures à base de registres sont ciblées dans le cadre de la synthèse de chemins de données.
Pour prendre en compte les problèmes de réordonnancement des données, nous souhaitons utiliser un modèle permettant d'exploiter des éléments mémorisants ayant une sémantique plus riche, comme ce que nous avons pu voir avec la solution proposée par A. Baganne.
Enfin, le modèle tel qu'il est proposé ne permet pas de générer des chemins de données intégrant plusieurs modes de fonctionnement.

4. Synthèse de composants de brassage de données

Dans cette troisième partie, nous présentons une famille de composants très répandus dans le domaine du traitement du signal et dont la fonction est de réaliser un brassage de données : les entrelaceurs [GAR01]. De ce point de vue, une interface de communication qui est supposée temporiser et réordonnancer les données a un comportement très similaire, c'est la raison pour laquelle nous rapprochons cette problématique de nos travaux.

Les turbo-codes [BER93] sont constitués d'un ou plusieurs éléments de calcul (encodeurs/décodeurs) qui fonctionnent de façon itérative, et d'un réseau de communication qui mélange les données échangées afin d'améliorer les résultats de la correction d'erreurs en brisant les relations de voisinage.
La nature itérative de ces architectures est une contrainte forte pour permettre aux concepteurs de répondre aux exigences en termes de performances (débit, surface, consommation) et de complexité du système. Une solution classique consiste à réaliser des turbo-codeurs/décodeurs avec une architecture parallèle. L'avantage de cette solution est que l'on augmente le débit du système puisque la latence du système devient la latence des sous-blocs qui le constituent. En revanche, la complexité et le coût du système sont accrus de par la nature parallèle de l'architecture. De plus, pour que les sous-blocs puissent travailler en parallèle, il est nécessaire que chacun accède à ses données puis les mémorise dans un bloc RAM. On trouvera donc autant de blocs RAMs que de blocs de traitement.

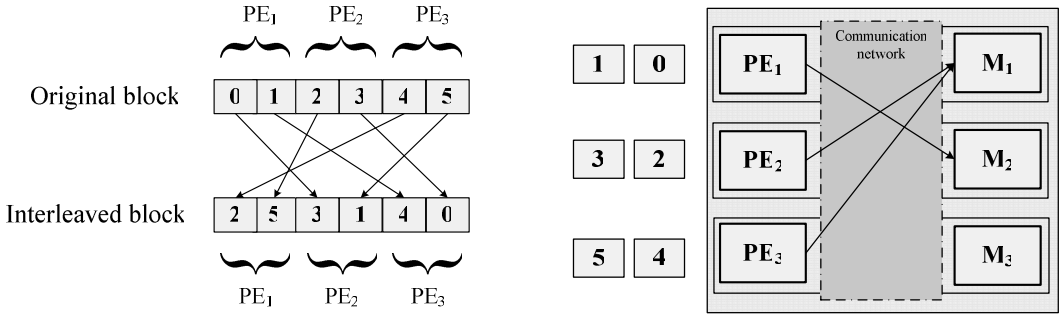


Figure 2.15. *Exemple d'un conflit d'accès à un banc mémoire*

Toutefois, en fonction de la règle d'entrelacement, différents modules peuvent essayer d'accéder simultanément à la même RAM [GIU02]. C'est l'exemple que nous avons représenté Figure 2.15,

dans lequel les éléments de calcul PE_2 et PE_3 tentent d'accéder en même temps au même banc mémoire M_1.
Dans une telle situation, il n'est pas possible de garantir la cohérence de fonctionnement du système. Ce problème est connu comme un problème dit de « collision ». Dans ce cas-ci, les accès à la mémoire doivent être décalés et soigneusement arbitrés, ce qui ralentit d'autant le processus de décodage. La solution consiste à concevoir un entrelaceur adapté et/ou à modifier l'architecture des blocs de codage/décodage. Plusieurs approches ont été proposées pour résoudre ces problèmes.

4.1. Les turbo-codes à roulettes

L'architecture des turbo-codes dite "Turbo-codes à roulettes", [GNA03a] [GNA03b], est conçue pour résoudre ce type de problème. Les auteurs proposent une nouvelle famille de turbo-codes dans laquelle le brassage se fait dans les deux dimensions (cf. Figure 2.16) et est réalisé par un ensemble de codes Convolutifs Récursifs Systématiques Circulaires (CRSC) [BER99], indépendants appelés "roulettes".
Cette contrainte fonctionnelle forte permet ensuite de garantir un comportement sans conflit pour l'entrelaceur. Les échanges de données sont alors réalisés par le jeu d'un ensemble de permutations circulaires.

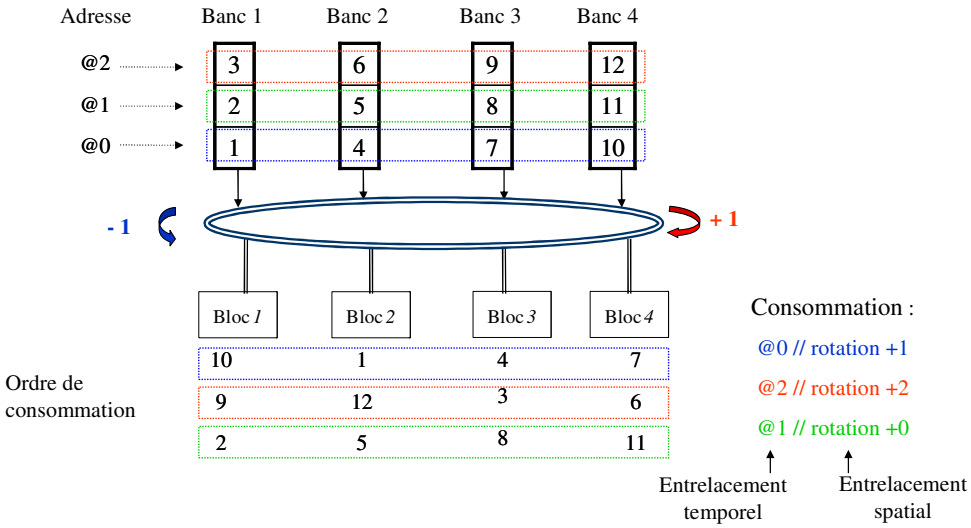


Figure 2.16. *Solution architecturale utilisant une permutation circulaire*

La Figure 2.16 présente l'architecture proposée qui réalise un mélange temporel et spatial. On trouve ainsi 4 blocs de traitements reliés à 4 bancs mémoires à travers un réseau de communication qui implémente une permutation circulaire. Les données mémorisées à une même adresse $@_i$ sont toujours transmises ensembles, mais leur cible, $Bloc_j$, peut varier selon cette permutation circulaire. Ainsi, les données sont mélangées dans deux dimensions :

- Dimension temporelle, en jouant sur les adresses d'accès aux bancs mémoires.
- Dimension spatiale, en jouant sur le nombre de décalages qui seront effectués par permutation circulaire.

Ces informations peuvent ensuite être utilisées pour générer les structures de contrôle de l'application. Néanmoins, la principale limitation est que les auteurs ne se conforment pas à un standard, mais doivent construire leur propre règle d'entrelacement.
De même, les permutations qu'ils peuvent réaliser sont limitées par le fait que les accès aux bancs mémoires doivent se faire sur les tous les bancs en même temps, et à la même adresse, ce qui limite la portée de l'entrelacement.
Enfin, bien que cette approche ne soit pas utilisable en l'état pour des architectures re-configurables (le contrôleur est implémenté en dur dans l'architecture), il est possible d'en imaginer une utilisation dans le cadre d'architectures multi-modes. Ainsi, un tel système utiliserait par exemple un contrôleur plus "souple", reprogrammable en cours d'exécution, à condition naturellement que tous les modes de fonctionnement respectent les contraintes fonctionnelles d'un CRSC, et à condition de dimensionner correctement les blocs mémoires.

4.2. Résolution des conflits à la conception

Lorsque l'utilisateur n'est pas libre de définir lui-même l'algorithme d'entrelacement (e.g. si celui-ci est imposé par un standard de communication [IEE07], [LAG00]…), plusieurs solutions ont été proposées.

On peut ainsi citer l'approche proposée dans [WEH04] et [THU02] qui vise à corriger les éventuels conflits sans modifier l'algorithme d'entrelacement. Le principe est ici de corriger les éventuels conflits en cours d'exécution : c'est le réseau d'interconnexion qui va se charger de mémoriser la/les donnée(s) conflictuelle(s), comme représenté Figure 2.17, jusqu'à ce que le bloc ciblé puisse traiter cette donnée.

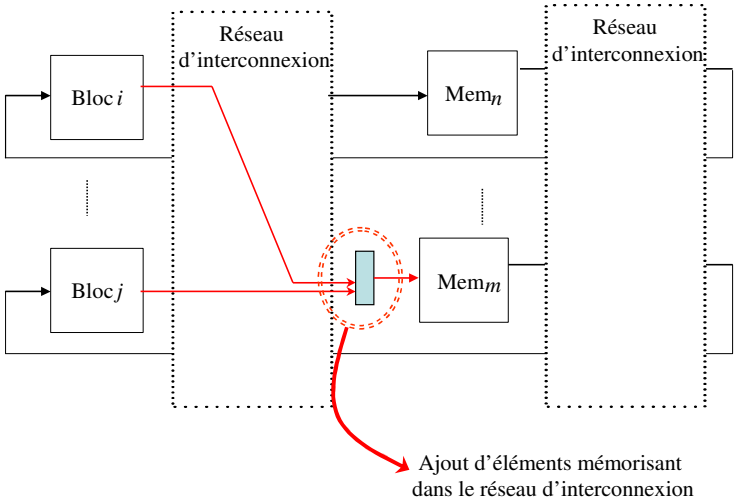


Figure 2.17. *Ajout d'éléments mémorisants dans le réseau d'interconnexion en cas de conflit*

Les éventuels conflits sont ainsi résolus à la volée, grâce à ces mémoires (registres ou FIFOs) additionnelles. Toutefois, cette approche suppose un surcoût, de par l'ajout d'éléments mémorisants dans un réseau d'interconnexion lui-même complexe. Ce dernier implémente, en effet, une

architecture de type réseau de Bënes [BEN65]. De plus, la latence globale de l'application est dégradée puisque ces éléments mémorisants sont utilisés pour temporiser les données conflictuelles. Cette solution peut également se révéler complexe à mettre en œuvre dans le cas d'architectures à fort parallélisme.
Enfin, une fois encore cette solution n'a pas été pensée pour des architectures multi-modes. Son application pour de tels systèmes supposerait que le concepteur soit à même de déterminer pour chaque bloc et dans chaque mode le nombre maximum de conflits à corriger. Puis de construire une architecture en prenant le pire cas pour chacun des blocs, et de générer un contrôle ad hoc capable de fonctionner dans chacun des modes.

4.3. Résolution des conflits par placement mémoire

Une troisième approche consiste à corriger les éventuels conflits dès la conception du circuit, en jouant sur la façon dont les données sont mémorisées dans les blocs mémoires. Ainsi, l'approche proposée dans [TAR04], consiste à trouver un placement des données en mémoire qui permette d'éviter tout conflit.
Pour ce faire, l'approche repose sur un algorithme à deux étapes :

- Dans un premier temps, les données sont affectées à des emplacements mémoires de façon arbitraire. En cas de conflit, la donnée n'est pas affectée.
- Puis, un algorithme de recuit-simulé ([KIR83]) est utilisé pour affecter les données restantes à des emplacements mémoires, tout en corrigeant au fur et à mesure les conflits créés.

Les auteurs ont démontré que leur approche convergeait toujours vers une solution car il en existait toujours une pour ce type d'architecture [TAR04]. Toutefois, ils ne peuvent estimer le temps que mettra leur algorithme pour converger vers un placement mémoire sans conflit.
L'approche retenue permet de proposer un placement des données en mémoire qui s'adapte à tous les standards de communication (à la différence de [GNA03a]) et qui évite un surcoût mémoire dans le chemin de données (à la différence de [WEH04]).

On peut toutefois reprocher à cette solution que l'architecture générée ne tire pas profit, d'une certaine régularité de l'architecture, lorsque c'est possible, pour générer un réseau d'interconnexion optimisé comme celui présenté dans [GNA03a].
De plus, cette approche ne cherche pas à réduire la quantité de mémoire utilisée (si possible), ni à intégrer plusieurs modes de fonctionnement. A ce titre [DIN05] propose une autre approche pour intégrer plusieurs modes de fonctionnement dans un même entrelaceur, à condition toutefois de modifier l'algorithme d'entrelacement. Toutefois cette dernière solution ne permet pas de se conformer à tous les standards de communication.

5. Bilan

Notre problématique consiste à proposer une solution permettant de réaliser l'adaptation des échanges de données entre deux éléments de calcul, en passant aléatoirement d'un mode de fonctionnement déterministe à un autre. Nous appellerons ce type de composant **Adaptateur Spatio-Temporel** (ou **STAR**, pour Space Time AdapteR). Comme nous l'avons évoqué précédemment, ce problème se retrouve dans le cadre (1) de l'intégration de composant virtuels, (2) la synthèse de chemins de données reconfigurables et (3) la synthèse de composant de brassage de données, type entrelaceur. Notre approche doit donc être suffisamment générique pour s'adapter aux différents domaines précités.

Définition

> *Un **Adaptateur Spatio-Temporel** est un composant qui réorganise les séquences de données dans une approche déterministe, afin de réaliser l'adaptation de la communication dans l'espace et dans le temps.*

Pour répondre à cette question, nous proposons une architecture de communication à grain fin (niveau scalaire) basé sur un ensemble d'élément mémorisants à sémantique forte (FIFO, LIFO, Registres) comme proposé dans [BAG98]. Toutefois, nous tirerons parti du fait que nous connaissons l'ordonnancement des transferts, ce qui nous permettra de proposer un modèle de graphe plus efficace que les graphes de compatibilité proposés par les auteurs. Comme indiqué dans [CHE04] notre approche doit pouvoir prendre en compte la complexité de l'architecture de multiplexage pendant l'exploration architecturale.
Nous nous inspirerons des modèles de graphes de contraintes présentés dans ces différents travaux pour modéliser des relations sémantiques entre les données. Afin de permettre une exploration efficace de l'espace des solutions, nous proposons un modèle formel permettant l'analyse des communications à un niveau scalaire. De plus, ce modèle devra être capable de modéliser différents modes de fonctionnement, et pour cela nous nous inspirons du modèle de graphe hiérarchique proposé dans [GUP04], que nous présenterons dans le chapitre suivant.

Chapitre 3

GRAPHE DE COMPATIBILITE DES RESSOURCES MULTI-MODES

Dans ce chapitre nous présentons une modélisation formelle des contraintes de communication supportant l'expression de relation sémantique forte entre les données échangées, et pouvant modéliser des systèmes devant fonctionner selon des modes de fonctionnement multiples.

Nous présentons également un ensemble de propriétés et de transformations permettant d'explorer efficacement l'espace des solutions architecturales.

1. Introduction

Pour pouvoir explorer l'espace des solutions architecturales, les flots de synthèse de haut niveau sont basés sur des représentations internes formelles à partir desquelles on cherche à générer une description structurelle de l'architecture.
Ainsi, la spécification algorithmique de l'application est compilée en une représentation interne à l'outil de synthèse. Les étapes de sélection, d'allocation et d'ordonnancement sont ensuite exécutées pour aboutir, après optimisation, à une description de niveau transfert de registre ou RTL (pour Register Transfer Level) de l'architecture.
Ces outils exploitent tous une représentation leur permettant d'appliquer leurs analyses. Les modèles les plus utilisés sont eux basés sur des graphes flot de données exprimant, avec plus ou moins de détails, les dépendances de contrôle et de données de l'application.

Nos travaux visent à proposer une solution permettant l'adaptation des communications entre des éléments de calcul. Nous n'avons pas à gérer de problème d'ordonnancement car le séquencement des données est dans notre cas une contrainte. Toutefois, la solution de génération d'architecture que nous proposons nécessite l'utilisation d'une représentation interne permettant d'explorer l'espace des solutions architecturales.

2. Modèles formels pour la synthèse de haut niveau

Les modèles de graphe de type flot de données DFG (DataFlow Graph), flot de contrôle CFG (ControlFlow Graph), flot de données et de contrôle CDFG (Control and DataFlow Graph)… sont les plus utilisés pour la synthèse de haut niveau. La contrainte que doit satisfaire l'exécution de tels modèles est de respecter l'ordre partiel imposé par la production/consommation des données (i.e. les dépendances de contrôle et de données).
Le lecteur intéressé trouvera un état de l'art des modèles formels en annexe B.

2.1. Les graphes de flot de données

Les graphes de flot de données, DFG, sont utilisés pour représenter les dépendances de données d'une application. Les DFG sont des graphes orientés acycliques dans lesquels les nœuds représentent les opérations et les variables, et où les arcs représentent les dépendances de données entre opérations et variables (cf. Figure 3.1).
Au cours du temps, un noeud du graphe ne peut être exécuté que lorsque toutes ses données d'entrée sont présentes. Chaque opération, à chacune de ses exécutions, consomme une donnée sur chacun de ses arcs d'entrée et combine ses entrées de manière à produire une donnée sur chacun de ses arcs de sortie. Cela induit une relation de dépendance d'exécution entre les différents noeuds du graphe. Cette relation implique qu'un noeud ne peut être exécuté qu'après complétion de l'ensemble de ses prédécesseurs (noeuds reliés à ses arcs présents en entrée), ce qui introduit un ordre partiel d'exécution des opérations. Dans le cas d'opérations indépendantes (opérations sans dépendances), il n'existe pas

de relation d'ordre d'exécution : ces dernières peuvent donc être exécutées dans n'importe quel ordre. Les graphes de type DFG permettent de mettre en évidence, par analyse des dépendances de données, les opérations pouvant être exécutées en parallèle.

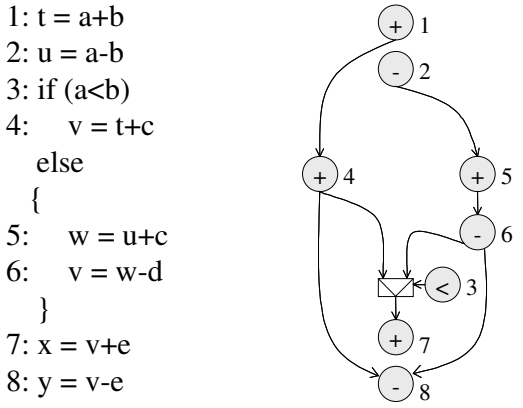


Figure 3.1. *Exemple simple de graphe de flot de données*

Une des principales limitations des DFG réside dans leur incapacité à exprimer des comportements non déterministes, où le nombre d'itérations d'une boucle pourrait par exemple varier en fonction des données fournies au composant.
Un DFG n'a pas pour vocation de modéliser des dépendances de contrôle d'une application. En effet, ce type de modèle est utilisé pour des applications dans lesquelles il n'existe que très peu, voir pas de contrôle. La non expression des dépendances de contrôle n'est donc pas pénalisante.

2.2. Les graphes de flot de contrôle

Les graphes de flot de contrôle, CFG, sont utilisés pour représenter les dépendances de contrôle d'une application. Les CFG sont des graphes orientés acycliques dans lesquels les noeuds représentent les structures conditionnelles ou des opérations, et les arcs modélisent des dépendances de contrôle et de séquence entre ces blocs (cf. Figure 3.2). Les noeuds conditionnels (disjonctions) présents dans ce graphe permettent l'expression de sémantiques de contrôle telles les branches conditionnelles et les boucles.

Ce type de graphe est généralement utilisé dans la modélisation d'applications dominées par le contrôle. Les noeuds spécifiques servant à modéliser le contrôle possèdent une sémantique analogue aux structures de contrôles rencontrées dans les langages de programmation impératifs (mise en séquence, exécution conditionnelle, boucles).

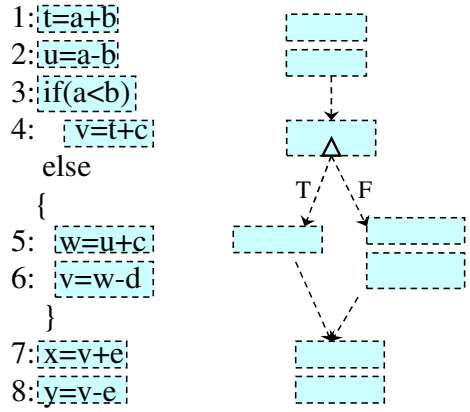


Figure 3.2. *Exemple simple de graphe de flot de contrôle*

A un instant donné un seul état du graphe de contrôle peut être actif, cela limite fortement la notion de parallélisme exploitable dans un tel graphe. La non expression des dépendances de données n'est pas pénalisante car ce type de modèle est dédié à la modélisation d'application orientée contrôle.

2.3. Graphes de flot de contrôle et de données

Pour modéliser des applications intégrant à la fois du contrôle et du traitement de données, il faut pouvoir combiner les informations présentes dans le DFG et dans le CFG. La combinaison de ces deux modèles au sein d'une même entité est appelée CDFG (pour Control-Data Flow Graph). Ce dernier permet de modéliser une application en prenant en compte ces deux aspects (cf. Figure 3.3).
Cette modélisation est utilisée dans de nombreux outils de synthèse de haut niveau car elle permet une représentation uniforme des structures de contrôle tout en permettant une certaine exploitation du parallélisme entre les opérations indépendantes

Le graphe de flot de contrôle va permettre un découpage de l'application en zone de contrôle de base ou *bloc de base* (cf. Figure 3.3.b). Puis le graphe de flot de données de l'application (cf. Figure 3.3.c) est superposé au CFG. Ainsi, dans chaque bloc de base de ce CFG, on va trouver les tâches du DFG qui y correspondent d'après le code originel (cf. Figure 3.3.d).
Ainsi, le CFG modélise les dépendances de contrôle entre les différents blocs de base au moyen de noeuds spécifiques dont la sémantique est analogue aux structures de contrôles rencontrées dans les langages de programmation impératifs.
Nous pouvons toutefois remarquer que les dépendances de données issues du DFG ne sont en revanche pas modélisées entre les blocs de base d'un CDFG, mais seulement localement à chaque bloc de base.

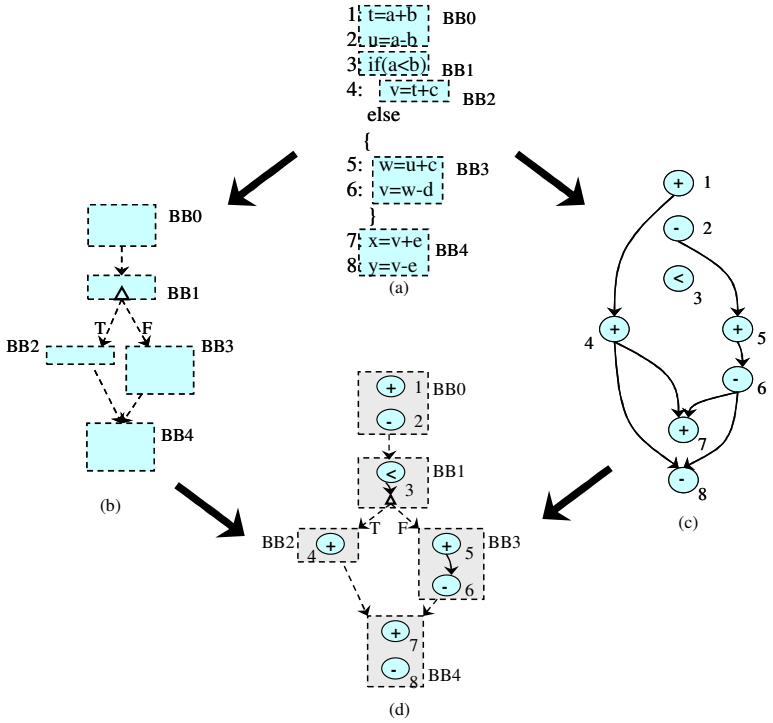


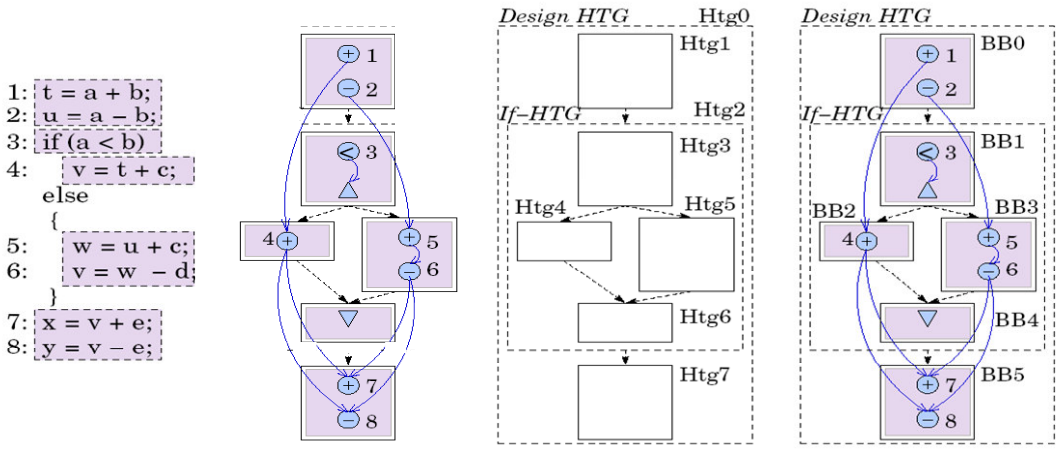
Figure 3.3. *Exemple de construction d'un graphe de flot de contrôle et de données*

Une autre limitation importante du modèle provient de l'impact du style d'écriture de la description algorithmique qui, pour des descriptions sémantiquement équivalentes, va fournir des graphes différents à cause des noeuds représentant les structures conditionnelles (Comme pour un CFG).

2.4. Les graphes de tâches hiérarchiques

Dans [GUP04], les auteurs présentent un outil de synthèse de haut niveau appelé Spark. En utilisant une représentation modélisant les dépendances de contrôle et de données, celui-ci est capable d'explorer et d'optimiser des architectures en jouant sur le parallélisme des opérations. Pour pouvoir efficacement exploiter ce concept, les auteurs ont besoin d'un modèle leur permettant de modéliser sans restriction cette notion de parallélisme dans un CDFG. C'est pourquoi ils proposent d'utiliser un modèle de graphe appelé graphe de tâches hiérarchiques, HTG (pour Hierarchical Task Graph) [GIR91].

La notion de graphe de tâches hiérarchiques vient enrichir les CDFGs classiques en modélisant des niveaux de hiérarchie sur ces CDFGs (cf Figure 3.4). Ainsi, les blocs de base du CDFG seront considérés comme des éléments de niveau hiérarchique le plus bas.
Les nœuds de type branchement conditionnel (*if-then-else*, *switch-case*) seront modélisés par un niveau de hiérarchie supérieur incluant un ensemble de bloc de base.
Les nœuds représentant des boucles (*for*, *while-do*, *do-while*) seront représentés par un niveau de hiérarchie supplémentaire et encapsuleront les branchements conditionnels et les blocs de base.
Enfin, à la différence des CDFG classiques, les dépendances de données peuvent être exprimées entre les blocs de base (*CDFG++* dans la Figure 3.4).



Exemple CDFG++ Hiérarchie HTG

Figure 3.4. *Niveaux hiérarchiques d'un HTG*

Cette formalisation en niveaux de tâches permet ensuite aux auteurs de proposer un ensemble de techniques de transformations de code visant à réduire l'impact du style de programmation sur l'architecture synthétisée.
Ces transformations, que nous ne détaillerons pas ici, vont consister à déplacer les opérations à travers les différents niveaux de hiérarchie. L'outil Spark synthétise des architectures sous contraintes de latence. Ce modèle lui permet donc de faire varier le parallélisme de l'application pour respecter cette contrainte de latence. Ces transformations auront également un impact sur le nombre d'opérateurs dans le circuit final.

Ce modèle offre donc une palette de transformations riche, permettant d'optimiser l'architecture finale en fonction des contraintes définies par l'utilisateur. Toutefois, le modèle HTG est basé sur des graphes de flot de données et de contrôle qui ne sont pas adaptés à notre problématique. Dans notre cas, nous n'avons pas à modéliser une véritable dépendance de données, seul le séquencement de celles-ci est pertinent. En revanche, nous pouvons nous inspirer de la hiérarchie et du contrôle proposé dans ce modèle pour formaliser les relations temporelles entre variables (résultats de l'ordonnancement) dans un contexte multi-modes.

3. Graphe de Compatibilité des Ressources Multi-Modes

Nous présentons ici un modèle de graphe appelé Graphe de Compatibilité des Ressources Multi-Modes. Cette représentation nous permet de formaliser les échanges de données entre des éléments de calcul. L'exploration de ce graphe doit nous permettre d'explorer l'espace de conception des adaptateurs spatio-temporels STAR.
Dans un premier temps, nous présentons un formalisme permettant de modéliser les échanges de données dans le cas d'architectures mono-modes. Ce modèle est ensuite étendu au cas des architectures multi-modes.

3.1. Les Graphes de Compatibilité des Ressources

3.1.1. Construction d'un Graphe de Compatibilité des Ressources

Le modèle formel que nous proposons est basé sur une analyse à grain fin (niveau scalaire) des échanges de données. De cette étude, nous allons pouvoir extraire l'information nécessaire à la définition d'une modélisation des communications adaptée à notre problème : les durées de vie de chacune des données échangées.

Problématique

Le problème de la synthèse des communications peut être énoncé comme suit :
Soit une séquence de transferts de données entre deux éléments de calcul,

> Quelle est : (1) la solution de mémoires tampons permettant de stocker à moindre coût toutes les données échangées et (2) quelle est la séquence de contrôle nécessaire à la gestion des communications tout en assurant la cohérence des données et la synchronisation des transferts ?

La notion de cohérence de données fait ici référence à trois critères :

- *Cohérence temporelle* : deux données transférées simultanément ne doivent pas être mémorisées dans le même élément de mémorisation.
- *Cohérence spatiale* : les lectures et écritures d'une même donnée doivent être effectuées sur le même élément de mémorisation.
- *Cohérence fonctionnelle* : dans le cas d'une architecture pipeline, les écritures d'un étage de pipeline ne doivent pas écraser des données encore utiles pour un autre étage du pipeline (recouvrement des communications entre itérations de calcul).

Complexité du problème

La recherche du nombre minimum de points mémoire pour stocker l'ensemble des données est équivalente à la recherche de l'index chromatique dans un graphe [STO94]. Il s'agit de rechercher le nombre minimum de couleurs nécessaires pour colorier tous les nœuds du graphe, en tenant compte de la contrainte suivante : deux nœuds reliés par une arête ne peuvent porter la même couleur.

Le problème de coloration de graphe est équivalent à un problème de partitionnement en cliques du graphe complémentaire. La théorie des graphes a très largement abordé ces problèmes de coloration et de partitionnement en cliques. Ils sont connus pour être de nature NP-complets.

Durée de vie d'une donnée

Notion clef dans notre approche, le concept de *durée de vie* des données est connu et exploité dans de nombreux travaux (e.g. [CHE04], [BAG98], [ZIS02]…). Nous formalisons ici la notion de durée de vie d'une donnée dans notre architecture d'adaptation des communications (cf. Figure 3.5).

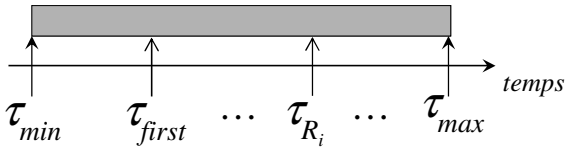


Figure 3.5. *Durée de vie d'une donnée*

Définition

La **durée de vie** *d'une donnée* v *dans le composant d'adaptation des communications est définie par :*

- *Soit* τ_{min_v}*, la date de réception de la donnée* v *dans l'adaptateur de communication depuis un port d'entrée,*
- *Soit* τ_{max_v}*, la date de dernière émission de la donnée* v *depuis l'adaptateur vers les ports de sortie,*

La durée de vie de la donnée est alors définie par l'intervalle temporel $\Gamma_v = [\tau_{min_v}, \tau_{max_v}]$

Comme représenté sur la Figure 3.5, nous formalisons également deux informations supplémentaires dans le cas d'émissions multiples d'une donnée depuis l'adaptateur vers des ports de sortie :

- τ_{first} : date de première émission de la donnée depuis l'adaptateur vers les ports de sortie.
- τ_{R_i} : date de i^{ième} émission de la donnée depuis l'adaptateur vers les ports de sortie.

Remarque

Si une donnée n'est lue qu'une seule fois, alors $\tau_{first} = \tau_{max}$.

Si une donnée est émise à plusieurs reprises, alors $\tau_{first} \leq \tau_{R_i} \leq \tau_{max}$, $\forall i \in \mathbb{N}$.

A partir de ces informations, nous pouvons associer une sémantique aux relations temporelles entre les données. Ces relations vont nous permettre d'identifier des *types de compatibilité* entre les données. A l'aide des informations temporelles de chaque donnée échangée, nous pouvons identifier le type de relation existant entre ces données.

Compatibilité Registre
Soient *a* et *b* deux données,

Règle :

Si ($\tau_{min_b} \geq \tau_{max_a}$) alors,
la relation temporelle entre *a* et *b* est dite de type "Registre".

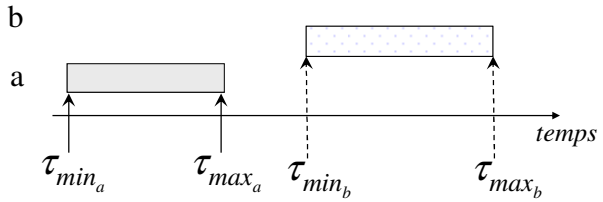


Figure 3.6. *Exemple de relation temporelle de type "Registre"*

Dans ce cas (i.e. Figure 3.6), les intervalles de durée de vie des données sont dits "non recouvrants". En d'autres termes, ces deux données peuvent être stockées dans le même point mémoire.

Compatibilité FIFO
Soient *a* et *b* deux données,

Règle :

Si [($\tau_{min_b} > \tau_{min_a}$) et ($\tau_{min_b} < \tau_{max_a}$) et ($\tau_{fisrt_b} > \tau_{max_a}$)] alors,
la relation temporelle entre *a* et *b* est dite de type "FIFO".

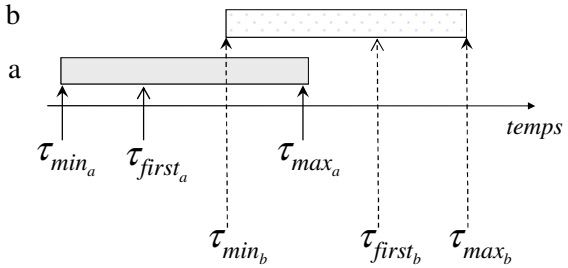


Figure 3.7. *Exemple de relation temporelle de type "FIFO"*

Dans ce cas (i.e. Figure 3.7), les intervalles de durée de vie des données sont dits "partiellement recouvrants". Ces deux données respectent bien entre elles une sémantique de type "première entrée, première sortie" ou FIFO (pour *First-In First-Out*).

Cela signifie que ces deux données peuvent être mémorisées dans une même file ou FIFO, en augmentant la taille au besoin (+1 case mémoire).

Nota Bene

La relation $(\tau_{min_b} < \tau_{max_a})$ permet une distinction formelle avec une compatibilité de type Registre.

Compatibilité LIFO

Soient *a* et *b* deux données,

Règle :

Si $[[(\tau_{min_a} < \tau_{min_b})$ et $(\tau_{max_b} < \tau_{first_a})]$ ou $[(\tau_{R_{ia}} < \tau_{min_b} < \tau_{max_b} < \tau_{R_{i+1a}})]]$ alors,
la relation temporelle entre *a* et *b* est dite de type "LIFO".

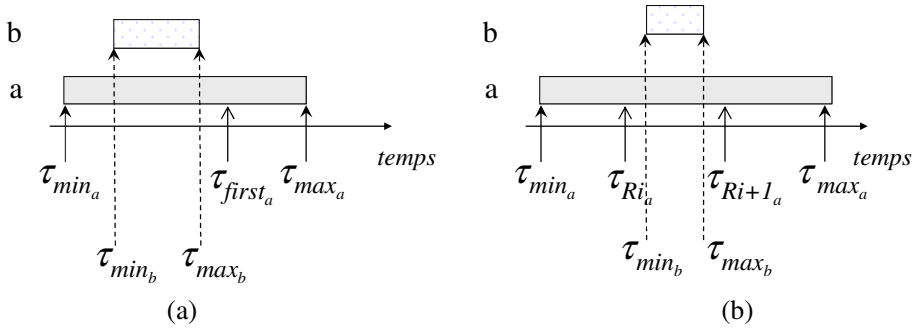


Figure 3.8. *Deux exemples de relation temporelle de type "LIFO"*

Dans ce cas (i.e. Figure 3.8), les intervalles de durée de vie des données sont dits "recouvrants-incluants". Ces deux données respectent bien entre elles une sémantique de type "dernière entrée, première sortie" ou LIFO (pour *Last-In First-Out*).
Cela signifie que ces deux données peuvent être mémorisées dans une même pile ou LIFO, en augmentant la taille au besoin (+1 case mémoire).

Notons que les deux sous expressions utilisées pour exprimer une compatibilité de type LIFO peuvent en fait se ramener à une même formulation :

- il existe une relation LIFO entre deux données *a* et *b* ssi,
la durée de vie de la donnée *b*, $[\tau_{min_b}, \tau_{max_b}]$ est entièrement comprise entre deux accès consécutifs à la donnée *a*. (τ_{min_a} puis τ_{first_a}, ou bien, $\tau_{R_{ia}}$ puis $\tau_{R_{i+1a}}$)

Dans tous les autres cas, les données sont dites incompatibles. Dans ce cas, il n'y a pas d'autre possibilité que d'utiliser deux éléments mémorisants distincts : un élément par donnée.

Nous savons maintenant identifier des types de compatibilité entre deux données. En extrapolant cette analyse à l'ensemble des données échangées, nous pouvons construire un graphe de compatibilité des ressources.

Définition d'un Graphe de Compatibilité des Ressources

Définition

Un **graphe de compatibilité des ressources**, *ou* RCG *(pour Resources Compatibility Graph), est un graphe de contraintes polaire orienté acyclique* G(V,E) *:*

- *L'ensemble des nœuds du graphe* **V= {Nv ∪ Nh}** *contient un unique nœud source et un unique nœud puits, tels qu'il existe un chemin partant du nœud source vers tout nœud de* V*, et que depuis tout nœud de* V *il existe un chemin vers le nœud puits.* **Nv** *contient les nœuds variables représentant les données ;* **Nh** *contient l'ensemble des nœuds de structure qui modéliseront les éléments mémorisants (FIFO, LIFO, Registres) assignés lors de l'exploration du RCG.*
- *L'ensemble des arcs de compatibilité* $E = \{(v_i, v_j)\}$ *représente les types de compatibilité entre les nœuds* v_i *et* v_j
- *Une étiquette* $t_{ij} \in T$ *est associée chaque arc* (v_i,v_j), $T=\{R, F, L\}$. *Cette étiquette permet d'identifier le type de compatibilité entre les nœuds* v_i *et* v_j.

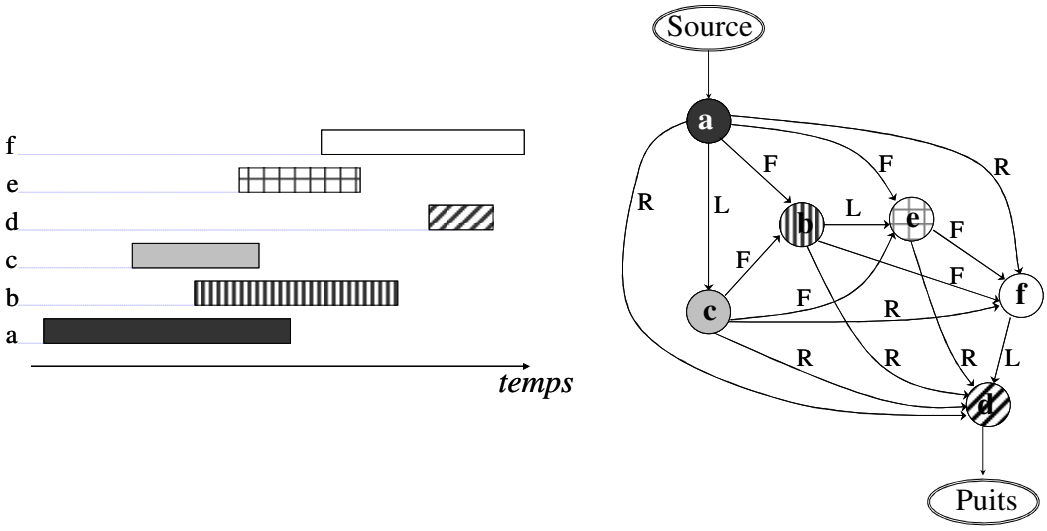


Figure 3.9. *Exemple de graphe de compatibilité des ressources*

Exemple

Un arc $e_{ij} = (v_i,v_j)$ étiqueté t_{ij} = L signifie que les données représentées par les nœuds v_i et v_j peuvent être mémorisées dans une même structure de mémorisation, si celle-ci respecte une sémantique LIFO.

Afin de simplifier les figures, nous ne représenterons plus dans la suite de ce document les noeuds *source* et *puits* dans les schémas.

Le graphe de compatibilité des ressources est construit en traitant les données dans l'ordre chronologique de leur arrivée dans l'adaptateur de communication (i.e. Figure 3.9). Pour chaque

donnée, on construira alors les arcs de compatibilité la reliant aux autres données, selon les règles de constructions énoncées précédemment.
Ainsi, si l'on considère l'exemple présenté dans la Figure 3.9, on constate que l'on construit :

- un arc de compatibilité Registre entre les données *a* et *d*,
- un arc de compatibilité LIFO entre les données *a* et *c*,
- un arc de compatibilité FIFO entre les données *a* et *b*,
- …

Complexité du graphe de compatibilité des ressources
S'il y a *n* noeuds à instancier, dans le pire cas il y aura *n-1* arrêtes partant des nœuds directement successeur du nœud source et allant vers tous les autres noeuds, *n-2* arrêtes partant des nœuds suivants vers tous les nœuds restants (dans l'ordre chronologique) ...
Dans le pire cas, le graphe contiendra :

n(n-1)/2 arcs.

Nous disposons maintenant d'un graphe synthétisant les informations concernant les relations temporelles entre les données. Il s'agit maintenant de parvenir à identifier sur ce graphe un ensemble d'éléments mémorisants qui constitueront l'architecture de mémorisation de notre adaptateur de communication.

3.1.2. Parcours d'un Graphe de Compatibilité des Ressources

Pour identifier les éléments mémorisants sur le modèle de graphe proposé dans [BAG98], les auteurs recherchent des *cliques maximales* dans des *graphes de compatibilité*. On peut néanmoins avoir trois objections à la solution proposée :

(1) la recherche de cliques maximales dans un graphe non orienté est un problème NP-complet,
(2) l'approche et la modélisation retenues ne permettent pas de dimensionner les éléments identifiés,
(3) dans cette solution, le type de la structure identifiée n'est pas connu à priori, il faut une analyse supplémentaire sur la clique pour déterminer si l'on se trouve face à une clique de compatibilité respectant une sémantique FIFO ou LIFO.

Nous avons vu que dans notre modèle l'information sur le type de structure pour mémoriser deux données est portée par les étiquettes des arcs. L'identification des éléments mémorisants (FIFO ou LIFO) en est simplifiée.

Notion de chemin

Définition

> *Un* ***chemin*** *est une séquence d'arcs parcourus dans la même direction. Pour qu'un chemin existe entre deux noeuds, il faut qu'il soit possible de se déplacer entre ces noeuds par une séquence d'arcs ininterrompue.*

Les règles d'identification des types de relations temporelles entre les données nous permettent d'étendre cette notion de chemin pour notre modèle formel.

Définition : Chemin de compatibilité FIFO

Un **chemin de compatibilité FIFO** *est un chemin dont tous les arcs sont étiquetés par des étiquettes FIFO.*

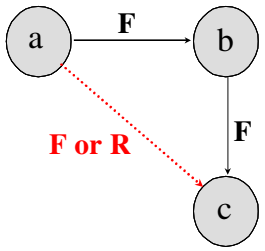


Figure 3.10. *Chemin de compatibilité FIFO*

Théorème 1

Soient *a*, *b*, *c* trois données distinctes en ordre chronologique ($\tau_{min_a} < \tau_{min_b} < \tau_{min_c}$),

Si *a* est compatible FIFO avec *b* et *b* est compatible FIFO avec *c*,

Alors,

***a* est compatible FIFO (ou Registre) avec *c* par transitivité.**

Démonstration

Comme nous l'avons formalisé précédemment, la relation FIFO entre a et b se traduit ainsi :

$$\tau_{min_b} > \tau_{min_a} \quad (1)$$

$$\tau_{first_b} > \tau_{max_a} \quad (2)$$

$$\tau_{min_b} < \tau_{max_a} \quad (3)$$

De la même façon, la relation FIFO entre b et c se traduit ainsi :

$$\tau_{min_c} > \tau_{min_b} \quad (1')$$

$$\tau_{first_c} > \tau_{max_b} \quad (2')$$

$$\tau_{min_c} < \tau_{max_b} \quad (3')$$

Par transitivité de la relation d'ordre on obtient :

$$\tau_{min_b} > \tau_{min_a} \text{ et } \tau_{min_c} > \tau_{min_b} \Rightarrow \tau_{min_c} > \tau_{min_a}$$

$$\tau_{first_c} > \tau_{max_b} \text{ et } \tau_{first_b} > \tau_{max_a} \text{ et } \tau_{max_b} \geq \tau_{first_b} \Rightarrow \tau_{first_c} > \tau_{max_a}$$

Nous obtenons :

$$\tau_{min_c} > \tau_{min_a}$$

$$\tau_{first_c} > \tau_{max_a} \quad \text{QED}$$

Ainsi, les données *a* et *c* ne peuvent pas être compatibles LIFO puisque $\tau_{first_c} > \tau_{max_a}$, mais elles sont compatibles FIFO par définition.

Toutefois, les relations (3) et (3') permettant de faire la distinction entre compatibilité FIFO et Registre ne peuvent être retrouvées dans le résultat final. Puisque ces relations sont utilisées pour distinguer les compatibilités *F* et *R* nous n'avons pas suffisamment d'information pour pouvoir faire cette distinction sur l'arc induit.
Donc, la compatibilité entre *a* et *c* peut être FIFO ou Registre (cf. Figure 3.10).

Corollaire

Un chemin de données compatibles FIFO, P_F , est par construction une ***clique de compatibilité FIFO*** regroupant un ensemble de données pouvant être mémorisées dans une même FIFO.

Ce corollaire peut être prouvé en appliquant récursivement le *Théorème 1* à l'ensemble des noeuds du chemin P_F.

Remarque

Puisque l'on ne peut déterminer si la relation induite est de type FIFO ou Registre, la profondeur de la FIFO n'est pas nécessairement égale au nombre d'éléments du chemin. Nous avons donc besoin de proposer une analyse spécifique pour dimensionner une FIFO, comme nous le verrons par la suite.

Définition : Chemin de compatibilité LIFO

Un **chemin de compatibilité LIFO** *est un chemin dont tous les arcs sont étiquetés par des étiquettes LIFO.*

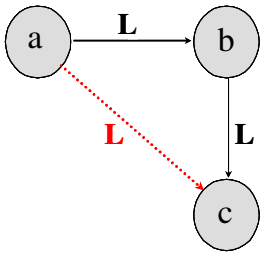


Figure 3.11. *Chemin de compatibilité LIFO*

Théorème 2

Soient *a*, *b*, *c* trois données distinctes en ordre chronologique ($\tau_{min_a} < \tau_{min_b} < \tau_{min_c}$),

Si *a* est compatible LIFO avec *b* et que *b* est compatible LIFO avec *c*,
Alors,

***a* est compatible LIFO avec *c* par transitivité.**

Démonstration

Comme nous l'avons formalisé précédemment, une compatibilité LIFO peut être identifiée selon deux types de relations distinctes.

Dans le premier cas, une compatibilité LIFO s'exprime ainsi: $[(\tau_{min_b} > \tau_{min_a})$ *et* $(\tau_{first_a} > \tau_{max_b})]$

Alors, la relation LIFO entre *a* et *b* se traduit par :

$$\tau_{min_b} > \tau_{min_a} \quad (1)$$

$$\tau_{fisrt_a} > \tau_{max_b} \quad (2)$$

De la même façon, la relation LIFO entre b et c se traduit ainsi :

$$\tau_{min_c} > \tau_{min_b} \quad (1')$$

$$\tau_{fisrt_b} > \tau_{max_c} \quad (2')$$

Par transitivité de la relation d'ordre on obtient :

$$\tau_{min_b} > \tau_{min_a} \text{ et } \tau_{min_c} > \tau_{min_b} \Rightarrow \tau_{min_c} > \tau_{min_a}$$

$$\tau_{fisrt_a} > \tau_{max_b} \text{ et } \tau_{fisrt_b} > \tau_{max_c} \text{ et } \tau_{max_b} \geq \tau_{first_b} \Rightarrow \tau_{fisrt_a} > \tau_{max_c}$$

Nous obtenons :

$$\tau_{min_c} > \tau_{min_a}$$

$$\tau_{first_c} > \tau_{max_a} \quad \text{QED}$$

Ainsi, les données *a* et *c* sont compatibles LIFO par définition.

Dans le cas où la relation LIFO s'exprime par : $[(\tau_{Ri_a} < \tau_{min_b} < \tau_{max_b} < \tau_{Ri+1_a})]$

Alors, la relation LIFO entre a et b se traduit par :

$$\tau_{Ri_a} < \tau_{min_b} < \tau_{max_b} < \tau_{Ri+1_a} \quad (1)$$

De la même façon, la relation LIFO entre b et c se traduit ainsi :

$$\tau_{Ri_b} < \tau_{min_c} < \tau_{max_c} < \tau_{Ri+1_b} \quad (1')$$

Par transitivité de la relation d'ordre, on obtient :

$$\tau_{Ri_a} < \tau_{min_c} < \tau_{max_c} < \tau_{Ri+1_a} \quad \text{QED}$$

Là encore, les données *a* et *c* sont compatibles LIFO par définition.

Dans le cas où les compatibilités entre *a* et *b*, et entre *b* et *c* ne sont pas construites à partir de la même règle, la compatibilité LIFO entre *a* et *c* est toujours valide. En effet, par définition des durées de vie nous avons, pour toute donnée λ:

$$\tau_{\min_\lambda} < \tau_{\text{first}_\lambda} < \tau_{\text{Ri}_\lambda} < \tau_{\text{Ri+1}_\lambda} < \tau_{\max_\lambda}$$

Corollaire

Un chemin de données compatibles LIFO, P_L , est par construction une ***clique de compatibilité LIFO*** regroupant un ensemble de données pouvant être mémorisées dans une même LIFO.

Ceci peut être prouvé en appliquant récursivement le *Théorème 2* à l'ensemble des noeuds du chemin P_L.

Remarque

A partir d'un même nœud origine, il est possible de construire plusieurs chemins LIFO (cf. Figure 3.12).
Dans ce cas, nous ne sommes plus en présence d'un chemin unique mais de chemins LIFO multiples P_{L_i} , tous issus d'un même nœud originel. Nous sommes alors en présence d'un ***arbre de compatibilité LIFO***.

Propriété : Arbre de compatibilité LIFO

Les nœuds de ces chemins respectent la propriété suivante :
- Soient P_{L_i} et P_{L_j} deux chemins LIFO issus d'un arbre de compatibilité LIFO,
- Soient v_x et w_y deux nœuds issus respectivement de P_{L_i} et P_{L_j},

alors,
les nœuds v_x et w_y sont reliés par un arc étiqueté avec une compatibilité *Registre*.

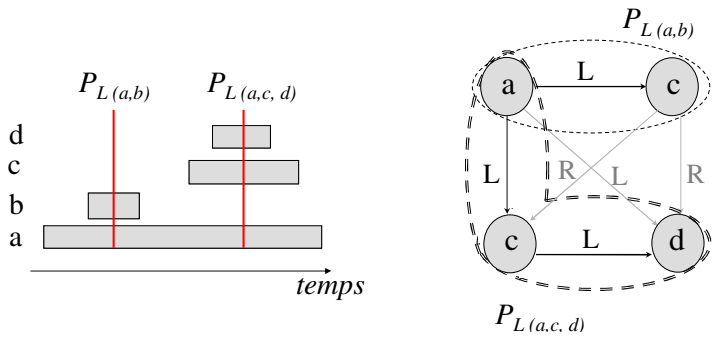


Figure 3.12. *Arbre de compatibilité LIFO*

Les données présentes dans un tel arbre peuvent toutes être mémorisées dans une seule et même LIFO. Toutefois, il faudra en tenir compte lors du dimensionnement. En effet, pour un chemin LIFO, la taille de la LIFO correspondante est égale au nombre de données qui constituent le chemin. Dans le cas d'arbres de compatibilité LIFO, les différentes LIFO qui en constituent les branches sont totalement non recouvrantes. Ainsi, si l'on regroupe toutes ces données dans une même LIFO, la taille de celle-ci sera égale à la longueur de la plus grande branche.

Définition : Chemin de compatibilité Registre

Un **chemin de compatibilité Registre** *est un chemin dont tous les arcs sont étiquetés par des étiquettes Registre.*

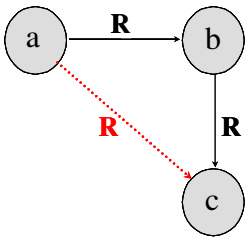


Figure 3.13. *Chemin de compatibilité Registre*

Théorème 3

Soient *a*, *b*, *c* trois données distinctes en ordre chronologique ($\tau_{min_a} < \tau_{min_b} < \tau_{min_c}$),

Si *a* est compatible Registre avec *b* et que *b* est compatible Registre avec *c*,

Alors,

***a* est compatible Registre avec *c* par transitivité.**

Démonstration

Comme nous l'avons formalisé précédemment, la relation Registre entre *a* et *b* se traduit ainsi :

$$\tau_{min_b} \geq \tau_{max_a} \qquad (1)$$

De la même façon, la relation FIFO entre b et c se traduit ainsi :

$$\tau_{min_c} \geq \tau_{max_b} \qquad (1')$$

Or, on sait que :

$$\tau_{max_b} > \tau_{min_b}$$

Donc, par transitivité nous obtenons :

$$\tau_{min_c} \geq \tau_{max_a} \qquad \text{QED}$$

Ainsi, les données *a* et *c* sont bien compatibles Registre par définition.

Corollaire

Un chemin de données compatibles Registre, P_R, est par construction une ***clique de compatibilité registre*** regroupant un ensemble de données pouvant être mémorisées dans un même Registre.

Ceci peut être prouvé en appliquant récursivement le *Théorème 3* à l'ensemble des noeuds du chemin P_R.

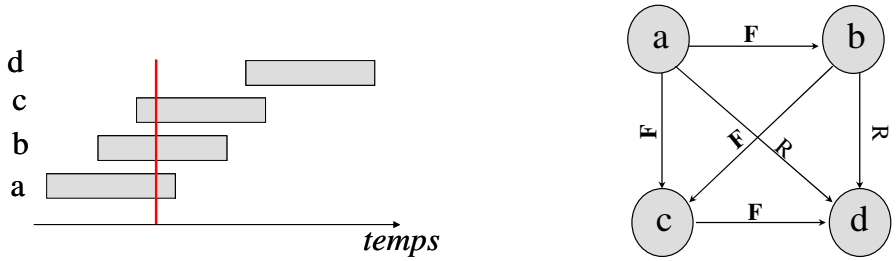
Remarque

Les données présentes sur un chemin Registre peuvent être regroupées dans un seul et même registre.

Cette notion de *fermeture transitive* des chemins nous permet d'identifier aisément les éléments de mémorisation. La modélisation retenue permet également de dimensionner directement ces éléments mémorisants. En effet, les corollaires nous indiquent que toutes les données d'un chemin de compatibilité FIFO (resp. LIFO) peuvent être mémorisées dans une seule et même FIFO (resp. LIFO), puisqu'elles constituent une clique de compatibilité FIFO (resp. LIFO).

Dimensionnement des structures FIFO

Nous avons remarqué que dans le cas des chemins FIFO, nous ne pouvions déterminer par simple parcours du chemin si les arcs induits portaient des compatibilités FIFO ou Registre. La profondeur d'une FIFO n'est donc pas égale au nombre maximum d'éléments qu'elle va temporiser, comme le montre l'exemple de la Figure 3.14.



(a) Contraintes temporelles (b) RCG correspondant

Figure 3.14. *Dimensionnement d'une FIFO*

Notre modélisation permet toutefois de proposer une solution simple : en observant le chronogramme des contraintes temporelles (cf. Figure 3.14.a), on constate que la profondeur de la FIFO est égale au nombre maximum de données recouvrantes entre elles parmi toutes les données du chemin. L'algorithme doit donc chercher à retrouver le recouvrement maximum dans le RCG.

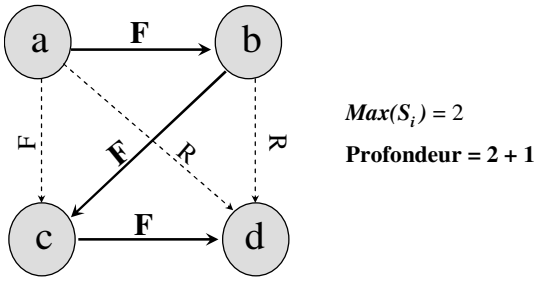


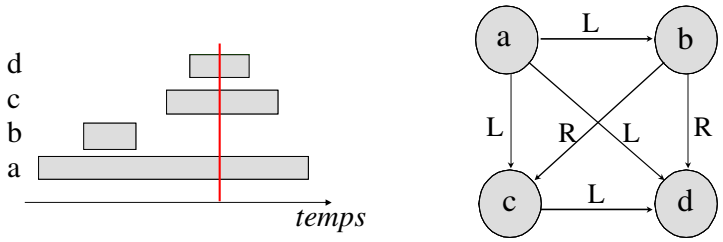
Figure 3.15. *Calcul de la profondeur d'une FIFO*

Algorithme de dimensionnement FIFO

Soit *P* un chemin de compatibilité FIFO,
Soit *i* un noeud du graphe appartenant à *P*,
Soit S_i le nombre d'arcs étiquetés FIFO incident au nœud *i* et
provenant d'un autre nœud du chemin *P*,
Alors,
Profondeur = 1 + max ({ S_i | pour tout noeud *i* de *P*}).

Dimensionnement des structures LIFO

Le dimensionnement des structures de type LIFO est légèrement différent. En effet, comme nous l'avons vu, il est possible d'identifier des sous-arbres de compatibilité LIFO sur le graphe. Dans ce cas, la profondeur de la LIFO n'est pas égale au nombre de données temporisées.



(a) Contraintes temporelles (b) RCG correspondant

Figure 3.16. *Dimensionnement d'une LIFO*

Dans le cas représenté sur la Figure 3.16.a, on constate qu'il est possible de mémoriser les données *a*, *b*, *c* et *d* dans une même LIFO. Toutefois, on constate qu'une LIFO de profondeur 3 est suffisante pour mémoriser ces quatre données. De fait, la profondeur de la LIFO sera égale à la longueur du plus long chemin du sous-arbre LIFO.

Algorithme de dimensionnement LIFO

Soit *A* un arbre de compatibilité LIFO,

Soit *c* un chemin de compatibilité LIFO issu de *A*,

Soit T_c la taille du chemin LIFO *c* (= nombre de nœuds du chemin),

Alors,

Profondeur = max ({ T_c | pour tout chemin *c* de *A*}).

Fusion de données - Nœud hiérarchique de structure

Comme nous l'avons indiqué précédemment, l'identification des éléments mémorisants se fait en parcourant le graphe. Lorsqu'une structure (FIFO, LIFO ou Registre) est identifiée, les données qui la composent sont alors regroupées dans un nœud hiérarchique de structure. Ces nœuds hiérarchiques regroupent un ensemble de variables. Ils seront considérés comme des éléments de mémorisation qui seront inclus dans le RCG. Pour ce faire, il est nécessaire de définir formellement la durée de vie d'un nœud hiérarchique :

Soit P un chemin de données compatibles, P = (v_1, ..., v_n),

- *si P est un* chemin de compatibilité FIFO ou Registre*,*

alors la durée de vie du nœud hiérarchique qui sera créé sera $[\tau_{min_{v_1}}, \tau_{max_{v_n}}]$.

- *si P est un* chemin de compatibilité LIFO*,*

alors la durée de vie du nœud hiérarchique qui sera créé sera $[\tau_{min_{v_1}}, \tau_{max_{v_1}}]$.

L'insertion de ce nœud hiérarchique dans le RCG se fait en respectant les mêmes règles de construction des compatibilités que nous venons de définir. Toutefois, pour garantir la cohérence sémantique du modèle :

- Nous considèrerons que des nœuds hiérarchiques LIFO et des nœuds hiérarchiques FIFO sont incompatibles (pas d'arcs de compatibilité les reliant).
- Nous ne construirons que les relations de type Registre entre le nœud hiérarchique que nous ajoutons et les autres nœuds du RCG (durée de vie non recouvrantes).

Si le parcours des chemins amène à la fusion de plusieurs nœuds hiérarchiques en un seul (i.e. dans un chemin de compatibilité Registre, cf. chapitre IV) dans ce cas, le nœud structure résultant possèdera un ensemble de durée de vie obtenu en faisant l'union des durées de vie des structures originelles.

L'assignation d'éléments mémorisants de types FIFO ou LIFO va permettre de réduire la complexité du contrôleur. En effet, pour piloter une FIFO (ou une LIFO) mémorisant *n* données seuls 2 signaux (*push* et *pop*) sont requis. En comparaison, pour piloter *n* registres il faut nécessairement utiliser *n* signaux distincts.

Le formalisme RCG permet d'exprimer des compatibilités entre données et des relations sémantiques fortes permettant l'exploration architecturale de l'adaptateur de communication. Toutefois, ce modèle ne permet de modéliser qu'un seul comportement. Or, nous souhaitons pouvoir modéliser et explorer plusieurs comportements déterministes distincts, c'est pourquoi nous proposons d'utiliser une modélisation par niveaux de hiérarchie pour prendre en compte ces configurations multiples.

3.2. Les Graphes de Compatibilité des Ressources Multi-Modes

Le modèle RCG présenté précédemment permet de formaliser des relations de compatibilité entre des données en analysant leurs relations temporelles. Toutefois, l'utilisation d'un RCG ne permet de modéliser qu'un unique mode de fonctionnement. Dans le cas d'architectures multi-modes ou multi-configurations, un modèle formel doit permettre la distinction de chacun des modes pour pouvoir ensuite explorer efficacement l'espace des solutions architecturales. Pour ce faire, nous proposons d'adjoindre deux concepts à notre modèle :

- Des niveaux hiérarchiques : bloc de base, nœuds multi-modes.
- Du contrôle : arcs de précédence, nœuds de disjonction/conjonction, coloration.

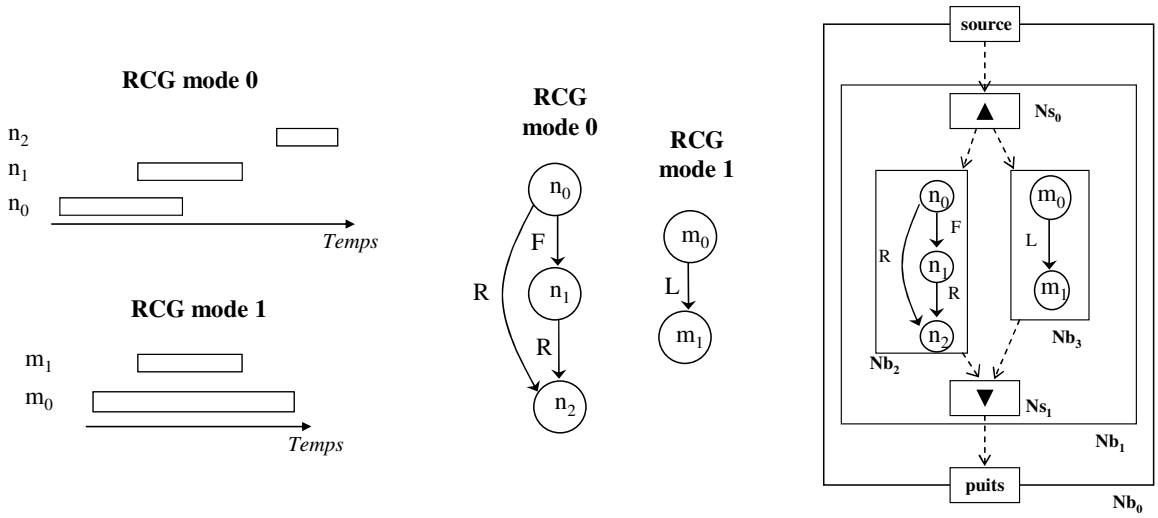
3.2.1. Construction d'un Graphe de Compatibilité des Ressources Multi-Modes

La construction d'un *Graphe de Compatibilité des Ressources Multi-Modes* (MMRCG, pour *Multi-Mode Resource Compatibility Graph*) repose sur deux étapes : premièrement, un RCG est construit pour chaque mode de fonctionnement de l'architecture ; puis ces RCGs sont encapsulés dans des blocs hiérarchiques.

Définition : Chemin de compatibilité Registre

Un **Graphe de Compatibilité des Ressources Multi-Modes** (MMRCG) *est un graphe polaire orienté acyclique* **G(V, E)** *:*

- *L'ensemble des nœuds du graphe* **V= {Nb ∪ Ns ∪ Nv ∪ Nh ∪ Nm}** *contient un unique nœud source et un unique nœud puits, tel qu'il existe un chemin de contrôle partant du nœud source vers tout nœud de* V*, et que depuis tout nœud de* V *il existe un chemin de contrôle vers le nœud puits.* **Nb** *contient les blocs de base modélisant les niveaux de hiérarchie ;* **Ns** *contient les nœuds de sélection qui modélisent les disjonctions/conjonctions entre les différents modes de fonctionnement ;* **Nv** *contient l'ensemble des nœuds variables modélisant chacune des données échangées ;* **Nh** *contient l'ensemble des nœuds de structure qui modéliseront les éléments mémorisants (FIFO, LIFO, Registres) assignés lors de l'exploration du MMRCG ;* **Nm** *contient l'ensemble des nœuds multi-variables/multi-modes (Utilisés pour modéliser l'assignation virtuelle de nœuds de différents modes).*
- *L'ensemble des arrêtes est défini par* **E= {Ac ∪ Av ∪ As}**. **Ac** *contient les arrêtes orientées (ou arcs) de compatibilité qui modélisent les types de compatibilité (FIFO, LIFO, Registre) entre des nœuds* (n_i, n_j) *tel que* n_i et $n_j \in C= \{Nv \cup Nh \cup Nm\}$ *;* **Av** *contient les arrêtes de vraisemblance pondérées entre les nœuds* (n_i, n_j) *ssi* $n_i \in Nb_k$, $n_j \in Nb_l$ et $Nb_k \neq Nb_l$. *;* **As** *contient les arrêtes orientées (ou arcs) de contrôles entre des blocs de base* Nb.
- *Un poids* **w** *est associé pour chaque arrête de vraisemblance* $a_v \in Av$.



(a) Contraintes de communication (b) RCG correspondant (c) Niveaux de hiérarchie

Figure 3.17. *Exemple de construction d'un graphe hiérarchique de compatibilité des ressources*

La Figure 3.17 montre un premier exemple de construction d'un MMRCG pour deux modes d'échanges de données (cf. Figure 3.17.a). Les RCG modélisant chacun de ces modes sont représentés Figure 3.17.b. La Figure 3.17.c représente le MMRCG regroupant ces deux modes de fonctionnement, les RCGs sont devenus les feuilles dans cette représentation et sont inclus dans des blocs de base (Nb_2 et Nb_3).

Définition :

Un **bloc de base** *représente un niveau hiérarchique. Il peut contenir :*
- *Un autre bloc de base (disjonction / nœud de sélection),*
- *Un RCG (niveau hiérarchique le plus bas).*

Pour pouvoir exploiter cette représentation MMRCG dans le cadre de la fusion de modes de fonctionnement, il faut déterminer comment fusionner au mieux les nœuds pour minimiser le coût de l'architecture finale. Pour ce faire, nous construisons les arcs de vraisemblance entre les nœuds des différents RCGs (cf. Figure 3.18).

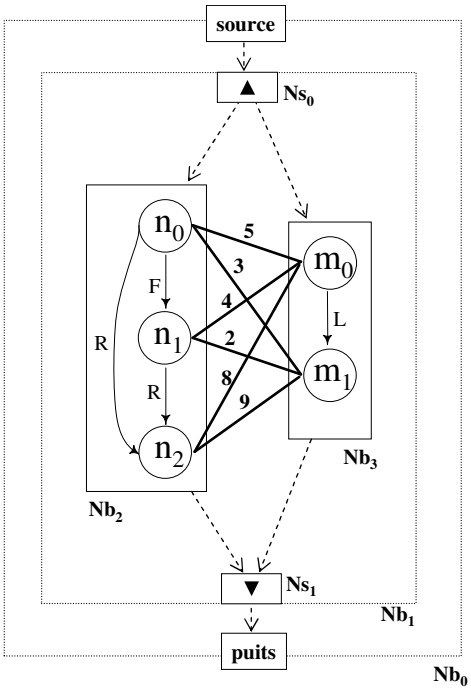


Figure 3.18. *Construction des arêtes de vraisemblance*

Définition :

Une **arête de vraisemblance** *pondérée* a_v *est une arrête reliant deux nœuds* n_i, *et* m_j *telle que*,
$n_i \in Nb_k$, $m_j \in Nb_l$ et $Nb_k \neq Nb_l$.
et, $n_i \in RCG_p$, $m_j \in RCG_q$ et $p \neq q$.

Les poids des arrêtes de vraisemblance sont calculés selon une heuristique qui sera présentée dans le chapitre IV.

Le principe de fusion retenu s'inspire des algorithmes de fusion dit de "*Maximal Weighted Bipartite Matching*" [CHE96]. Cet algorithme cherche à apparier deux à deux les nœuds de deux graphes. Le but est alors de maximiser le gain cumulé en additionnant les poids des arrêtes de fusion sélectionnées. La solution que nous avons retenue est sensiblement différente et s'appuie sur les propriétés des RCGs.

3.2.2. Parcours d'un Graphe de Compatibilité des Ressources Multi-Modes

Les poids des arrêtes de vraisemblance représente le coût de la *pseudo-assignation* (défini ci-après) des données reliées par cette arrête, dans un même point mémoire. Notre objectif sera donc de minimiser le coût de la fusion en minimisant la somme totale des poids des arrêtes sélectionnées. De plus, comme nous allons le voir, la sémantique de construction des RCGs va nous permettre de réduire de façon très importante l'espace des solutions de fusion, en supprimant un grand nombre d'arrêtes de vraisemblance après chaque fusion de nœuds.

Pseudo assignation de deux nœuds

Lorsque deux nœuds appartenant à différents modes sont fusionnés, on créé un nœud multi-variables/multi-modes (*Nm*) qui contiendra les deux nœuds. Chaque nœud sera alors coloré avec une couleur permettant de distinguer son mode d'origine (cf. Figure 3.19).

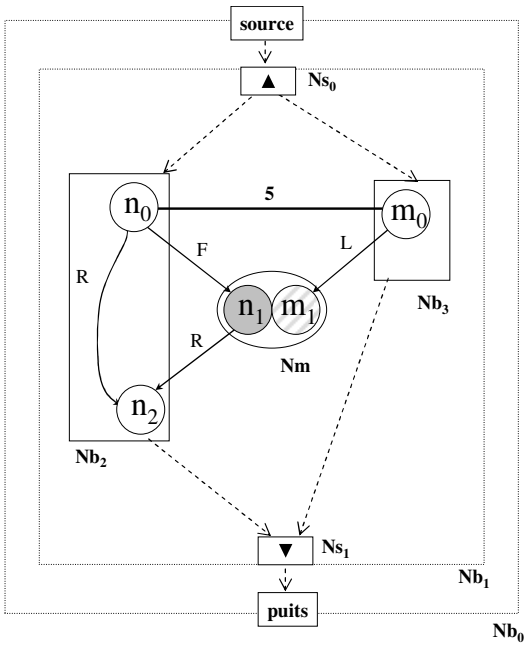


Figure 3.19. *Exemple de fusion des modes – Pseudo-assignation des nœuds* n_1 *et* m_1

Cette opération revient à assigner virtuellement les deux variables à un même point mémoire. Ce point mémoire sera ensuite piloté différemment dans l'architecture finale selon le mode de fonctionnement. Nous parlons ici de ***pseudo-assignation*** en comparaison à la véritable assignation d'une donnée sur un élément mémorisant (FIFO, LIFO ou Registre).

Règles de pseudo-assignation de nœud :

(1) Pseudo-assignation

- La fusion de nœuds ne peut se faire qu'entre des nœuds n'appartenant pas au même bloc de base (i.e. entre des nœuds appartenant à différents modes).

(2) Unité de comportement

- Dans un nœud multi-variables/multi-modes *Nm,* on trouvera au plus un nœud variable issu de chaque mode de fonctionnement.

(3) Pseudo-assignation de nœuds variables

- Si la fusion se fait entre deux nœuds variables Nv_i de niveau hiérarchique h_i et Nv_j de niveau hiérarchique h_j *alors,*

  un nœud multi-variables/multi-modes *Nm* est créé et ajouté au niveau hiérarchique $h_n = min(h_i, h_j) - 1$.

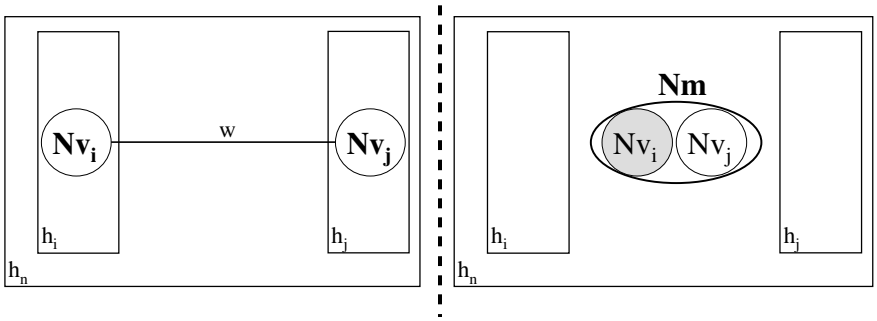


Figure 3.20. *Pseudo-assignation de nœuds variables*

Ce nœud *Nm* contiendra (cf. Figure 3.20) les nœuds Nv_i et Nv_j colorés avec les couleurs attribuées à leurs modes de fonctionnement respectifs (e.g. Figure 3.19).

(4) Pseudo-assignation d'un nœud variable dans un nœud multi-variables/multi-modes

- Si la fusion se fait entre un nœud multi-variables/multi-modes *Nm* de niveau hiérarchique h_m et un nœud variable *Nv* de niveau hiérarchique h_v, alors,

  le nœud *Nv*, coloré avec la couleur attribuée au mode de fonctionnement dont il est originaire, est ajouté au nœud *Nm* de niveau hiérarchique h_m.

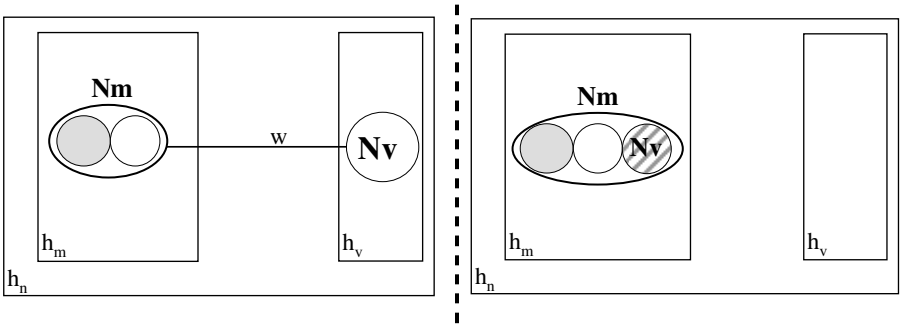


Figure 3.21. *Pseudo-assignation d'un nœud variable dans un nœud multi-variables*

(5) Cohérence de compatibilités

- Lorsqu'un nœud variable *Nv*, provenant d'un RCG_i, est ajouté à un nœud multi-variables/multi-modes *Nm*, les arcs de compatibilités entre *Nv* et les autres nœuds variables issus de RCG_i sont maintenus.

- Lorsqu'un nœud variable Nv_i est ajouté à un nœud multi-variables/multi-modes Nm_i, s'il existe un nœud multi-variables/multi-modes Nm_j, tel qu'il existe un arc de compatibilité entre Nv_i et un nœud variable Nv_j dans Nm_j , et s'il existe un autre nœud variable Nw_i dans Nm_i qui possède un arc de compatibilité avec un noeud variable Nw_j dans Nm_j,

  les arcs de compatibilité reliant les nœuds variables de Nm_i et les nœuds variables de Nm_j sont transformés selon les lois de transformation reportées dans la Table 3-1 (cf. Figure 3.22 et l'exemple de transformation Figure 3.23).

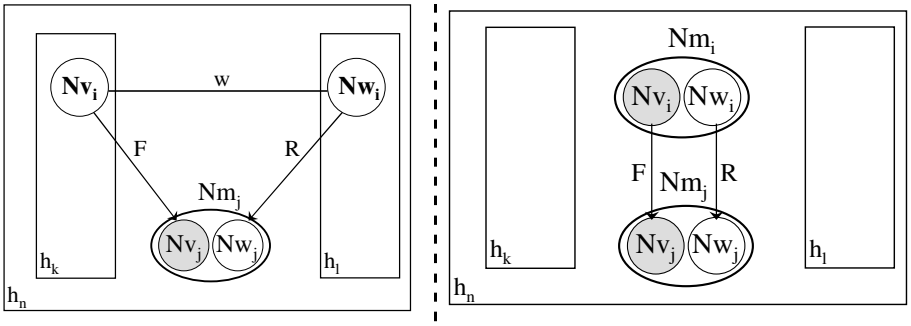


Figure 3.22. *Cohérence de compatibilité – Avant transformation des compatibilités*

(6) Cohérence temporelle du modèle

- Les arcs de compatibilité reliant les variables des nœuds multi-variables/multi-modes doivent être orientés dans le même sens.

(7) Maintien de la pertinence du modèle

- Lorsqu'une pseudo-assignation d'un nœud variable est effectuée, les ***arrêtes de vraisemblance non pertinentes*** sont supprimées (cf. Figure 3.18 et Figure 3.19).

(8) Condition d'arrêt de l'exploration et homogénéisation du MMRCG

- Lorsqu'il n'existe plus d'arrêtes de vraisemblance dans le MMRCG, l'algorithme de pseudo-assignation est terminé. Les éventuels nœuds variables non fusionnés restant sont alors colorés avec la couleur de leurs modes respectifs et déplacés au même niveau d'abstraction que les nœuds multi-variables/multi-modes. Les blocs de base qui servaient à modéliser formellement les différents modes sont alors supprimés : c'est l'étape d'***homogénéisation*** du MMRCG(cf. Figure 3.25).

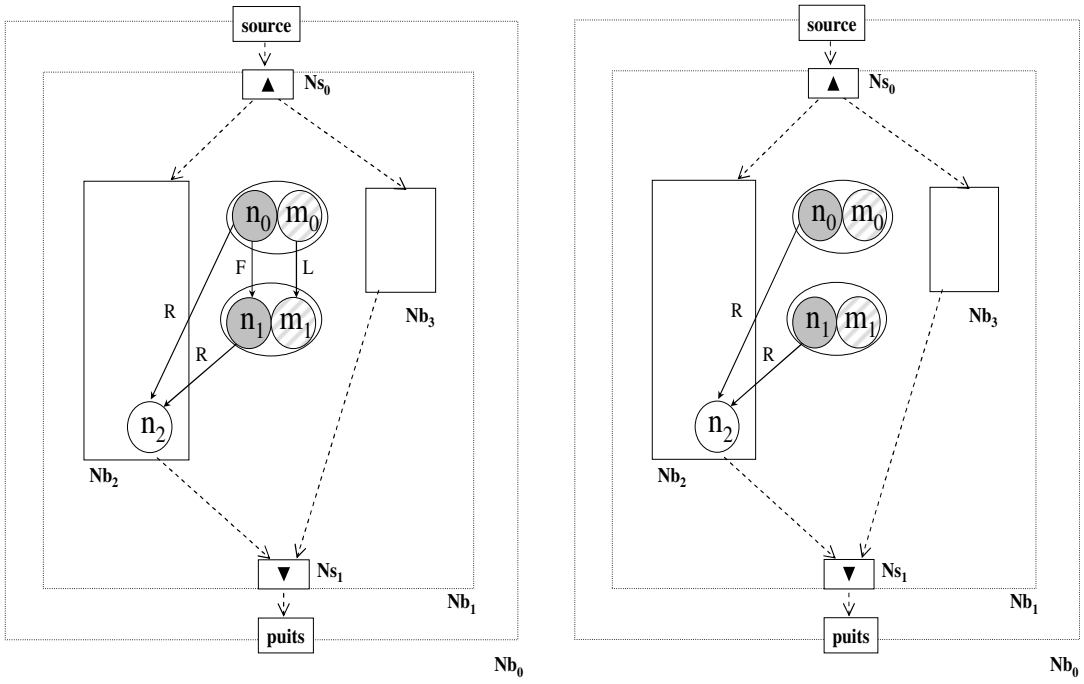


Figure 3.23. *Exemple de fusion des modes – Pseudo-assignation des nœuds* n_0 *et* m_0

Lois de transformation des compatibilités

La pseudo-assignation de deux nœuds variables dans un même point mémoire va avoir un impact sur les types de compatibilité représentés sur les arcs entre les nœuds multi-variables/multi-modes. Les arcs de compatibilité entre ces nœuds vont être transformés selon les lois reportées dans la Table 3-1. On notera que selon ces transformations, la relation de compatibilité Registre se comporte comme un élément neutre.

Table 3-1

Lois de transformation des arcs de compatibilités

RCG_1 \ RCG_2	FIFO	LIFO	Reg	*Rien*
FIFO	F	-	F	-
LIFO	-	L	L	-
Reg	F	L	R	-
Rien	-	-	-	-

Arrête de vraisemblance non pertinente

La fusion de nœuds (cf. Figure 3.19) entraîne la destruction d'arrêtes de compatibilité devenues non pertinentes, réduisant d'autant l'espace des fusions à explorer. De fait, les RCGs sont des graphes orientés avec une notion de précédence temporelle entre les données. Ainsi, lorsque deux nœuds sont fusionnés, les arrêtes de vraisemblance reliant le passé de l'un au futur de l'autre n'ont plus de raison d'être. En effet, comme le montre la Figure 3.24, dans une telle situation, les arcs reliant les nœuds multi-variables/multi-modes seraient inversés selon le mode de fonctionnement, la cohérence temporelle ne serait plus respectée.

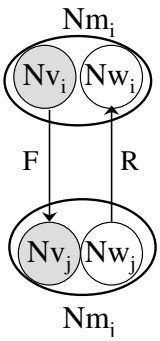


Figure 3.24. *Violation de la cohérence temporelle du MMRCG*

Définition : Arrêtes non pertinentes

Soient deux graphes de compatibilité des ressources G_1 *et* G_2 *à fusionner,*
Soient n_1 *et* n_2 *deux nœuds appartenant respectivement à* G_1 *et* G_2*,*
Soient Pre_1 *et* $Post_1$ *l'ensemble des nœuds appartenant à* G_1 *et ordonnancés respectivement avant et après* n_1*,*
Soient Pre_2 *et* $Post_2$ *l'ensemble des nœuds appartenant à* G_2 *et ordonnancés respectivement avant et après* n_2*,*

L'ensemble des **arrêtes de vraisemblance non pertinentes** ENP *est défini par,*

$ENP = \{ENP_{ij} = (h_1v_i, h_2v_j) / (h_1v_i \in Pre_1 \text{ et } h_2v_j \in Post_2) \text{ ou alors, } (h_1v_i \in Post_1 \text{ et } h_2v_j \in Pre_2)\}$.

Complexité d'un MMRCG

Dans un MMRCG on trouvera, après fusion, au maximum un nœud de multi-variables/multi-modes par combinaison de modes possible :

Soit n *le nombre de modes de fonctionnement à fusionner,*
Alors le nombre maximum de combinaisons de modes est au plus :

$$\sum_{i=1}^{n} C_n^i$$

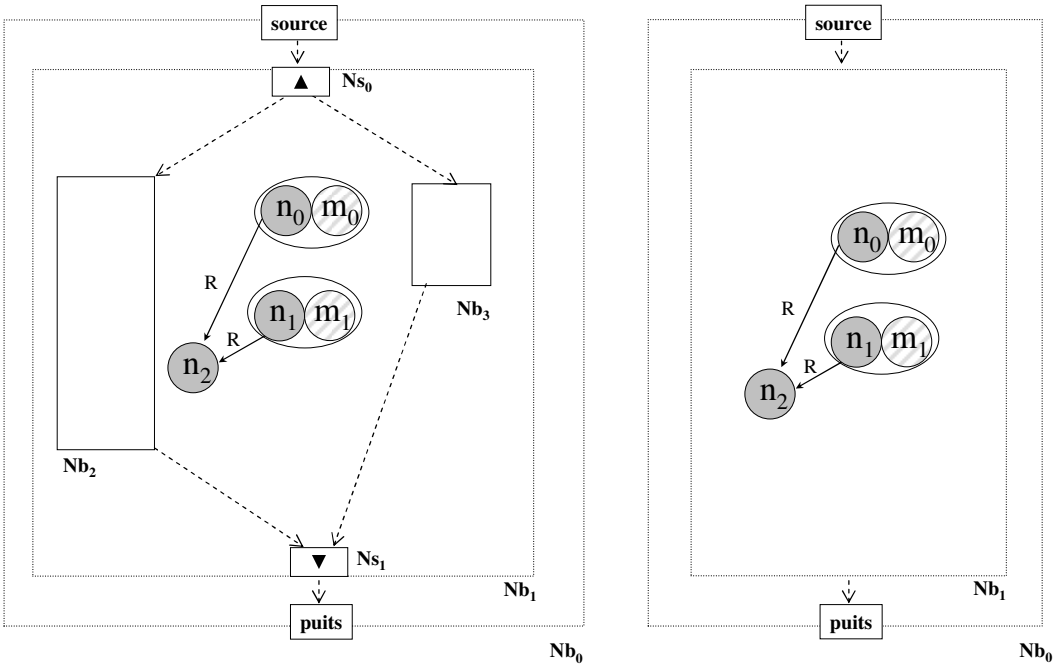


Figure 3.25. *Exemple de fusion des modes – Homogénéisation du MMRCG*

L'exploitation du MMRCG fusionné va permettre l'exploration et la génération de l'architecture finale selon les méthodologies d'exploration relatives à un RCG mono-mode classique, telles que nous les avons présentées précédemment.

Remarque

Dans le cadre de l'utilisation de notre modèle dans des flots de synthèse de haut niveau (synthèse de chemins de données), l'utilisation de blocs de base et d'arcs de contrôle permet également de représenter des branchements conditionnels (i.e. mutuellement exclusifs). L'utilisation de niveaux de hiérarchie imbriqués est alors une approche classique, c'est par exemple celle utilisée dans les HTG.

Il serait également possible de modéliser ces branchements conditionnels sous la forme d'un arbre n-aire, comme dans le cas d'un MMRCG représentant plusieurs modes de fonctionnement, à la place de niveaux hiérarchiques imbriqués. Toutefois, on s'éloigne là de la structure de contrôle de l'application. L'utilisation de notre modèle n'a pour l'instant pas été explorée dans ce cadre.

L'utilisation de couleurs pour distinguer les différents modes peut quand à elle sembler redondante avec l'utilisation de blocs de base dédiés à chaque mode, mais cette approche nous permet d'obtenir une représentation finale unique et compacte du MMRCG en supprimant la notion de hiérarchie, facilitant ainsi l'exploitation du modèle.

4. Conclusion

Dans ce chapitre, nous avons proposé une modélisation des contraintes de communication supportant l'expression sémantique des relations temporelles entre les données échangées et pouvant modéliser des modes de fonctionnement multiples. Nous avons présenté un ensemble de propriétés et de transformations permettant d'explorer efficacement l'espace des solutions architecturales.
Il convient de noter que ce formalisme et le flot qui lui est associé sont aisément réutilisables dans divers environnements de conception : dans le de cadre de l'intégration d'IPs, pour la génération des partie mémorisation et aiguillage des chemins de données et pour la génération de composant de brassage de données. Comme nous allons le voir dans le chapitre suivant, un ensemble d'outils dédiés ont étés proposés. Notre approche est ainsi indépendante de l'outil de synthèse *GAUT*. Le chapitre suivant détaille le flot de conception et les outils proposés pour l'utilisation du modèle MMRCG.

Chapitre 4

IMPLEMENTATION

Dans chapitre nous présentons le modèle architectural ciblé par notre approche. Nous décrivons également le flot de synthèse associé qui exploite le modèle MMRCG présenté dans le chapitre précédent.

1. Introduction

Le processus de synthèse consiste à passer d'un niveau de description à un niveau de description moins abstrait (i.e. une suite de raffinement permet donc de transformer une spécification en une implémentation). Pour être fiable, ce processus doit se baser sur des modèles formels et des transformations prouvées (correctes par construction). Ainsi, le modèle *MMRCG*, présenté dans le chapitre précédent, est exploité par le flot de synthèse que nous proposons afin de générer des composants d'adaptation d'échanges de données.

Dans ce chapitre, nous présentons tout d'abord l'architecture ciblée pour implémenter le composant d'adaptation de communications. Cette architecture est constituée d'éléments mémorisants distincts à sémantique forte: des FIFOs, des LIFOs et des Registres. L'ensemble de ces éléments forme, après synthèse, un adaptateur de communication de niveau RTL que nous avons nommé *STAR* pour *Space-Time AdapteR*.
Dans une seconde partie, nous présentons le flot de conception que nous proposons pour générer cette architecture. Ce flot réalise un ensemble de transformations successives du MMRCG pour générer une description structurelle du composant STAR, en faisant notamment appel aux notions de chemins typés (cf. Chapitre III) et à des métriques d'exploration (coût de l'architecture générée, contraintes technologiques…). L'approche de conception proposée repose sur trois étapes : la modélisation des contraintes de communication du système à l'aide du formalisme MMRCG (construction d'un MMRCG), la synthèse sous contrainte du composant STAR (transformations du modèle MMRCG) et la validation fonctionnelle de l'architecture obtenue (génération de bancs de tests et simulation).
Enfin, dans une dernière partie, nous évoquerons une extension de notre modèle architectural pour la génération de composant d'entrelacement exploitant une architecture particulière. Cette solution a fait l'objet d'une demande de dépôt de brevet auprès de l'INPI (Institut National de la Propriété Industrielle) [CHA07a].

1.1. Architecture cible

La temporisation et le réordonnancement des données (adaptation temporelle) sont réalisés par les éléments mémorisants à sémantique forte (FIFOs, LIFOs et Registres). L'aiguillage des données depuis un port d'entrée vers n'importe quel port de sortie (adaptation spatiale) est réalisé par la logique de multiplexage.

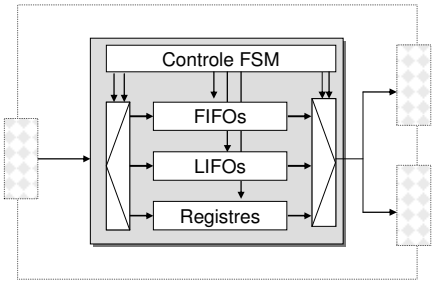


Figure 4.1. *Schéma bloc simplifié d'une architecture STAR*

La Figure 4.1 représente une vue schématique d'une architecture STAR simplifiée. Ce composant STAR est constitué d'un ensemble d'éléments mémorisants, de logique d'aiguillage des données (composant de multiplexage), d'une machine de contrôle (FSM) et de ports de communication.

Le modèle MMRCG permet de mettre en exergue les relations temporelles (FIFO, LIFO et Registre) entre les durées de vie des données échangées. Il s'agit d'exploiter ce modèle pour générer un composant STAR (cf. Figure 4.1). Nous exploitons la sémantique de ces éléments pour implémenter le réordonnancement des données.

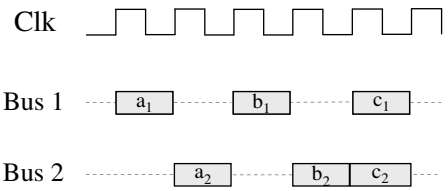


Figure 4.2. *Utilisation d'adaptateurs STAR pour l'intégration de composants multiples*

Définition : Notion de bus

> *Nous appelons* **bus** *un ensemble de signaux (nappe de fils).*

Définition : Notion de port

> *Nous appelons* **port** *un ensemble de bus.*

Notre approche consiste à générer un composant STAR par ensemble de bus d'entrée synchronisés. Nous parlerons dans ce cas de *port synchrone*. Ainsi, sur l'exemple de la Figure 4.2 si les chronogrammes des bus 1 et 2 sont synchronisés, alors ces deux bus seront regroupés dans un unique port et partageront une même architecture STAR.

Définition : Notion de port synchrone

> *Nous appelons* **port synchrone** *un ensemble de bus d'entrée ou un ensemble de bus de sortie dont les chronogrammes de communication sont synchronisés.*

Dans ce cas, il ne peut exister aucun décalage entre les comportements temporels de ces ports.

1.1.1. Intégration de composants virtuels

Dans le cadre de l'intégration de composant virtuels, si les contraintes architecturales impliquent que l'élément dont le comportement requiert une adaptation des communications soit connecté en entrée à un ensemble de composants non synchronisés, alors un STAR sera généré pour chaque port d'entrée synchrone (e.g. deux ports d'entrée sur la Figure 4.3).

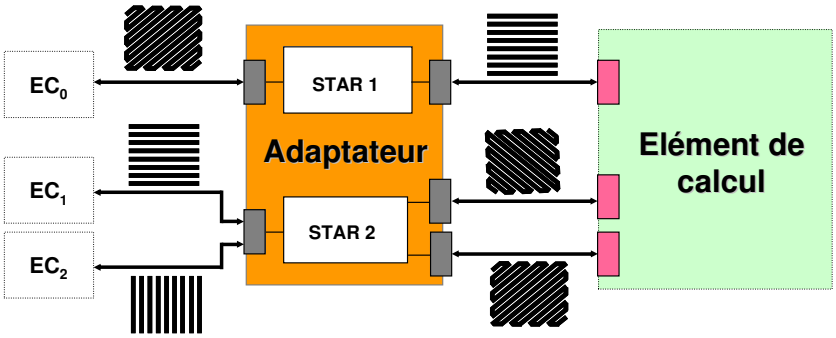


Figure 4.3. *Adaptation des communications avec ports désynchronisés*

L'architecture ciblée (cf. Figure 4.4) sera composée d'un adaptateur STAR par port d'entrée synchronisé, et chacun de ces STARs peut cibler n'importe quel port de sortie. Le contrôle de l'architecture est réparti entre des *contrôleurs locaux* aux composants STAR, les signaux de commande de tous les composants STARs et enfin un *contrôleur principal* (ou processeur de synchronisation) gérant l'interface avec le reste du système.

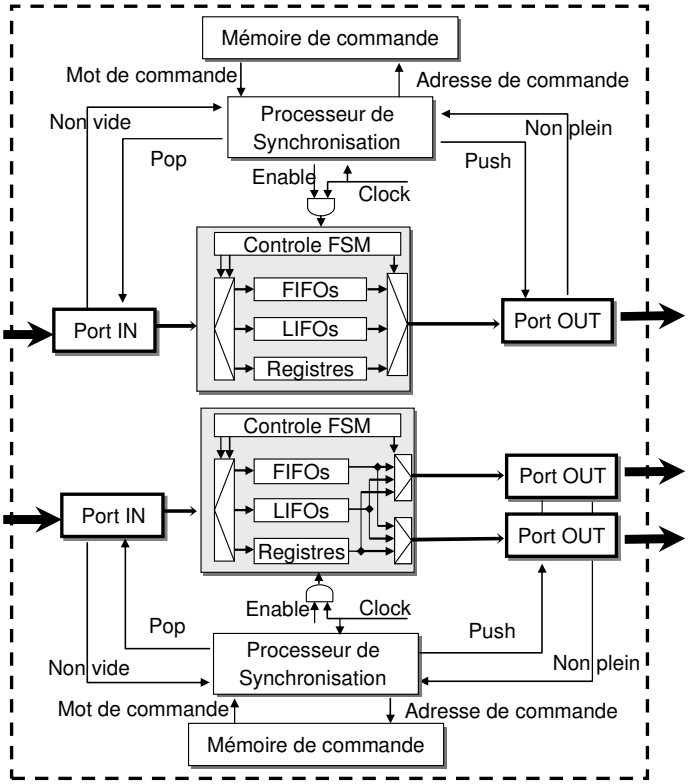


Figure 4.4. *Deux architectures STAR avec mécanisme LIS pour l'intégration*

En effet, chaque composant doit pouvoir fonctionner sans nécessairement dépendre des autres composants du système. Pour ce faire, l'adaptateur de communication STAR peut utiliser un schéma de communication et une interface de type *Globalement Asynchrone – Localement Synchrone* (*GALS*).

Les systèmes GALS

Le principe de l'approche système "globalement asynchrone et localement synchrone", introduit dans [CHA84], est basé sur le découpage d'un circuit complexe en sous-systèmes localement synchrones et optimisés en fréquence. Ils combinent des sous-systèmes synchrones qui communiquent entre eux au travers d'un réseau asynchrone point à point.

Une évolution très prometteuse des GALS est la théorie des systèmes insensibles à la latence ou LIS (pour *Latency Insensitive System*) [CAR01], dont le réseau de communication n'est plus asynchrone mais pseudo asynchrone. Elle partage avec les NoCs le haut degré de parallélisme potentiel et avec les bus la simplicité de mis en oeuvre. Le réseau est constitué de liaisons point à point pipelinées par adjonction d'unités de stockage nommées stations de relais. Cette approche, qui consiste à stocker temporairement les données lors de leurs transits, utilise un protocole de synchronisation des transferts de données entre blocs de traitements et stations de relais. L'objectif est de réduire le coût et la complexité du réseau d'interconnexion (fils).

Nous avons représenté**,** Figure 4.4**,** deux composants STAR intégrés dans un système LIS tel que proposé dans [BOM05]. Le principe est de geler l'interface de communication et l'IP qu'elle encapsule, s'il manque une donnée en entrée d'un bus (canal FIFO vide), ou s'il est impossible de transmettre une donnée en sortie (canal FIFO plein). Ce mécanisme permet de prendre en compte l'indéterminisme des latences de transferts de données. *Notre architecture est ainsi déterministe sur des ordres partiels de transfert de données* (ordre déterministe mais avec des "latences de transferts" variables). Cette architecture est utilisée pour implémenter l'unité de communication des composants générés par l'outil GAUT. Elle sera également utilisée dans le cadre de la génération de composants de brassage de données (e.g. entrelaceurs) avec notre approche.

Génération d'un chemin de donnée

Dans le cadre de la génération de chemin de données, nous avons indiqué que les opérateurs présents dans le chemin de donnée sont considérés comme des éléments de calcul devant être interconnectés. Dans cette situation, la contrainte de synchronisation n'a plus de raison d'être. En effet, cela reviendrait à créer autant de composants STAR distincts qu'il n'y aurait d'opérateurs dans le chemin de données. Ces différents STARs ne pouvant mutualiser leurs ressources, cette solution empêcherait toute optimisation de la partie mémorisation du chemin de données.

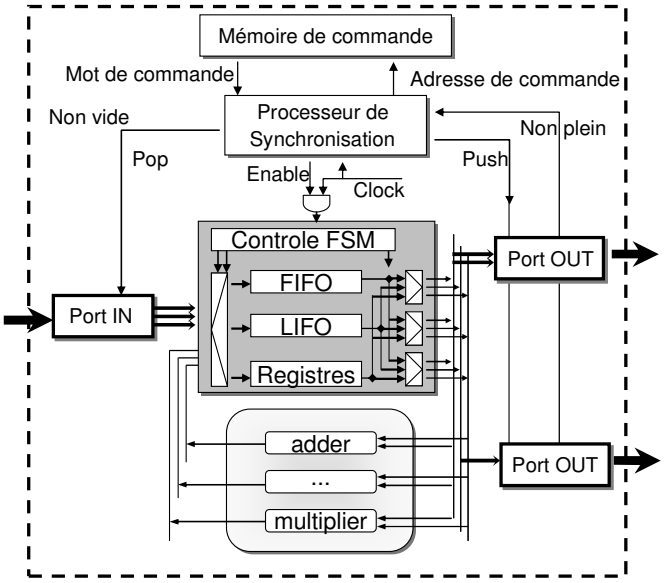


Figure 4.5. *Utilisation d'une architecture STAR dans le cadre de la synthèse de chemins de données*

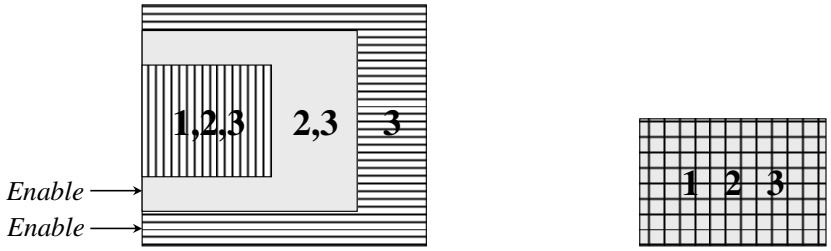
La Figure 4.5 présente la cible architecturale que nous avons retenue dans le cadre de la synthèse de chemins de données. Cette architecture peut, comme toute autre architecture STAR, exploiter des éléments mémorisants de types FIFOs, LIFOs et Registres. Toutefois, si le concepteur souhaite n'utiliser que des registres dans le chemin de données, il peut le spécifier directement sous la forme d'une contrainte dans notre flot de conception.

1.1.2. Solutions architecturales

Dans le cadre d'architectures multi-modes (adaptateurs de communication, entrelaceur ou chemin de données), deux approches d'optimisation se sont imposées :

- Optimisation en surface.
- Optimisation en consommation (statique et dynamique).

Deux solutions d'exploration architecturale ont donc été proposées pour optimiser l'architecture générée en surface (et donc en consommation statique), ou en consommation dynamique.



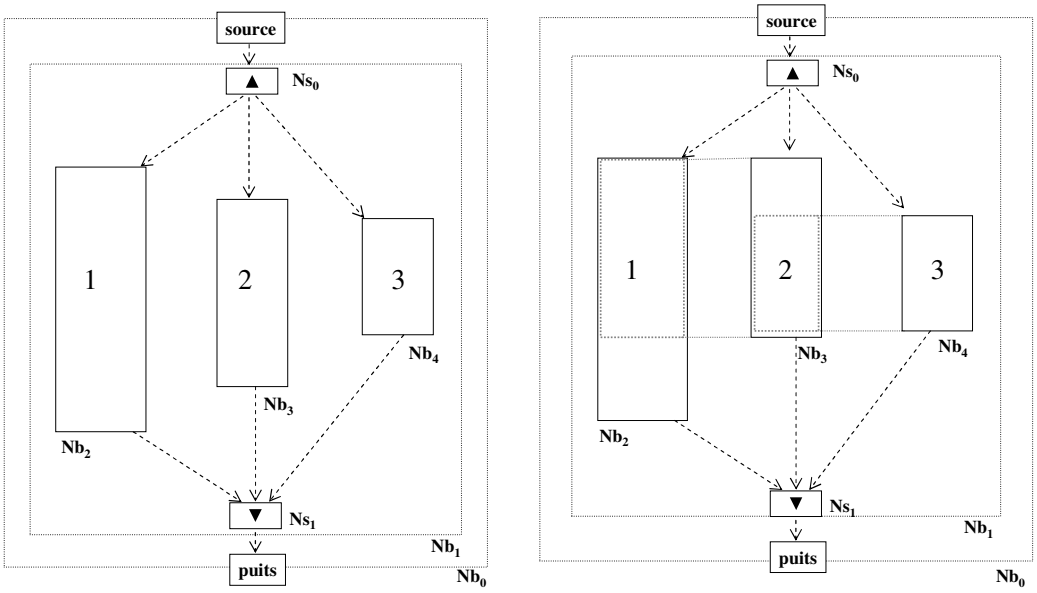
(a) Optimisation en consommation (b) Optimisation en surface

Figure 4.6. *Solutions architecturales distinctes en fonction des objectifs d'optimisation pour une architecture intégrant trois modes de fonctionnement*

Comme nous le verrons par la suite, cette analyse de l'impact de la solution architecturale est pour l'instant une hypothèse théorique : des expérimentations actuellement en cours en détermineront la validité.
Si le concepteur estime que la consommation statique et la surface du circuit sont les contraintes les plus fortes, alors le flot de conception que nous proposons génèrera une architecture optimisée en surface et intégrera tous les modes de fonctionnement en seule et même entité (cf. Figure 4.6.b).
Si au contraire, le concepteur considère que la consommation dynamique de l'application est le critère le plus important, notre flot génèrera des architectures multi-modes intégrant un mécanisme de gel d'horloge (cf Figure 4.6.a). Dans ce cas, lorsque le mode de fonctionnement 2 est exécuté, les zones de l'architecture correspondant à ce mode seront en fonctionnement et les zones dédiées aux autres modes (ici le mode 3) seront gelées. Toutefois, l'architecture finale sera plus coûteuse en surface, car l'algorithme cherche à maximiser les zones qui pourront être gelées.
Ces deux classes de solutions architecturales résultent de deux parcours différents du modèle MMRCG. En particulier pour la génération d'une architecture avec mécanisme de gel, qui tire parti de la possibilité de procéder à une étape d'*équilibrage* sur le MMRCG.

Equilibrage d'un MMRCG
Comme nous l'avons évoqué précédemment, la fusion des différents modes de fonctionnement peut entraîner la création de nœuds multi-variables/multi-modes regroupant toutes les combinaisons possibles de modes.

Pour la génération de l'architecture finale, l'exploitation de cette représentation peut s'avérer complexe car le nombre de combinaisons à prendre en compte, notamment pour la génération du contrôle, peut s'avérer très élevé. Il est toutefois possible de réduire la complexité du MMRCG fusionné en optimisant le nombre de combinaisons de modes dans le MMRCG résultant. Pour ce faire, avant de fusionner les modes, nous procédons à une étape d'***équilibrage*** du MMRCG (cf. Figure 4.7).



(a) MMRCG avant équilibrage (b) MMRCG après équilibrage

Figure 4.7. *Equilibrage d'un MMRCG modélisant trois modes de fonctionnement*

L'équilibrage d'un MMRCG cherche à définir une solution de fusion des différents modes de fonctionnement (cf. Figure 4.7.b) sans créer de nœuds variables isolés, excepté pour le RCG ayant le plus grand nombre de données (cf. Figure 4.8).

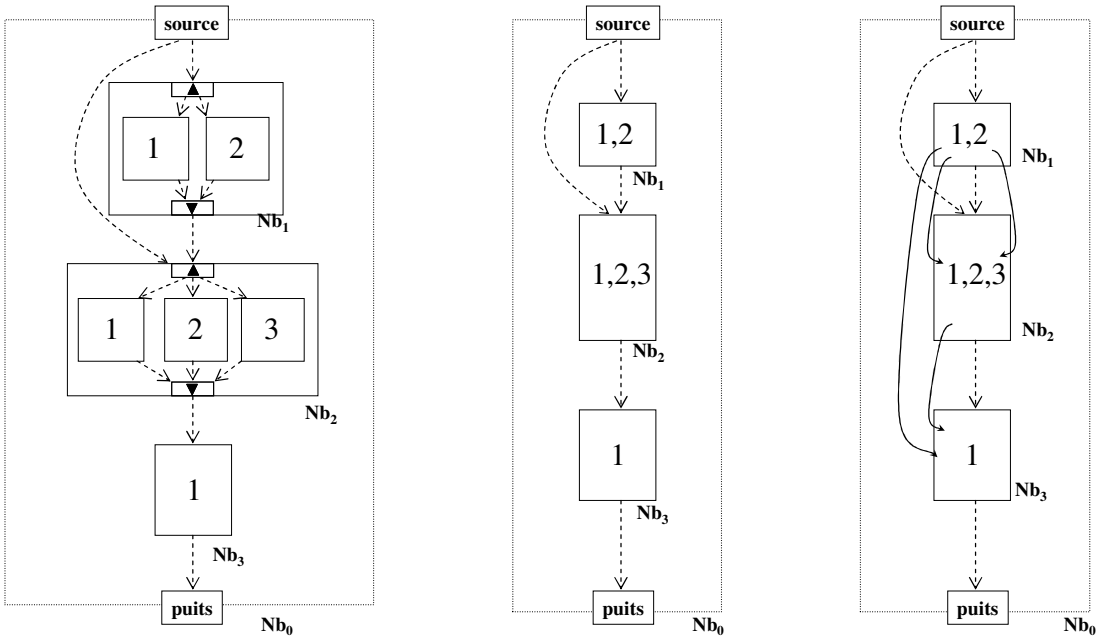


Figure 4.8. *Exploitation des niveaux hiérarchiques issus de l'équilibrage*

En maximisant les recouvrements entre les différents modes, les architectures STAR générées peuvent exploiter efficacement des mécanismes de gel. En effet, lorsque le mode 2 est utilisé, il est possible de geler les parties de l'architecture correspondant au bloc Nb_3 (cf. Figure 4.8.b).

Le maintien ou non des arcs de compatibilité entre les niveaux hiérarchiques (cf. Figure 4.8.b et c) aura un impact sur l'architecture finale en autorisant, ou non, l'assignation d'éléments mémorisants fonctionnant dans plusieurs modes.
En effet, si ces arcs ne sont pas maintenus (cf. Figure 4.8.b), alors les éléments mémorisants seront assignés localement à chaque bloc de base. Il est alors possible de générer un mécanisme de gel local à chaque bloc de base.
Si au contraire ces arcs de compatibilité sont maintenus (cf. Figure 4.8.c), l'espace des solutions architecturales pour le composant STAR multi-modes sera plus important. Toutefois il est impossible de générer des composants pouvant être gelés par simple application d'un mécanisme de "gel par bloc". Par exemple, l'exploration du modèle peut assigner une FIFO fonctionnant dans les modes 1, 2 et 3 mais en manipulant des données issues des blocs Nb_2 et Nb_3. Or, si le mode d'exécution demandé est le mode 3, comme tous les autres éléments mémorisants manipulant des données issues du bloc Nb_3 , cette FIFO ne pourra être gelée.

Ainsi, l'équilibrage réduit la complexité du MMRCG fusionné et facilite la génération d'une architecture finale plus régulière. Toutefois, en réduisant la portée des heuristiques de fusion des

modes, cette solution génèrera des architectures multi-modes moins optimisées en surface (nombre d'éléments mémorisants, complexité de la logique d'aiguillage).

1.2. Flot de conception "STARSystem"

L'exploration architecturale et la génération d'un composant STAR à partir du modèle MMRCG, présenté dans le chapitre III de ce manuscrit, requièrent un flot de synthèse dédié. Le flot que nous proposons est composé d'un ensemble d'outils logiciels que nous avons développés et que nous regroupons sous l'appellation *STARSystem*.

1.2.1. Flot de synthèse de haut niveau

Comme nous l'avons évoqué dans le chapitre I, les flots de synthèse de haut niveau peuvent être décomposés en cinq grandes étapes : compilation, sélection, allocation, ordonnancement et assignation.

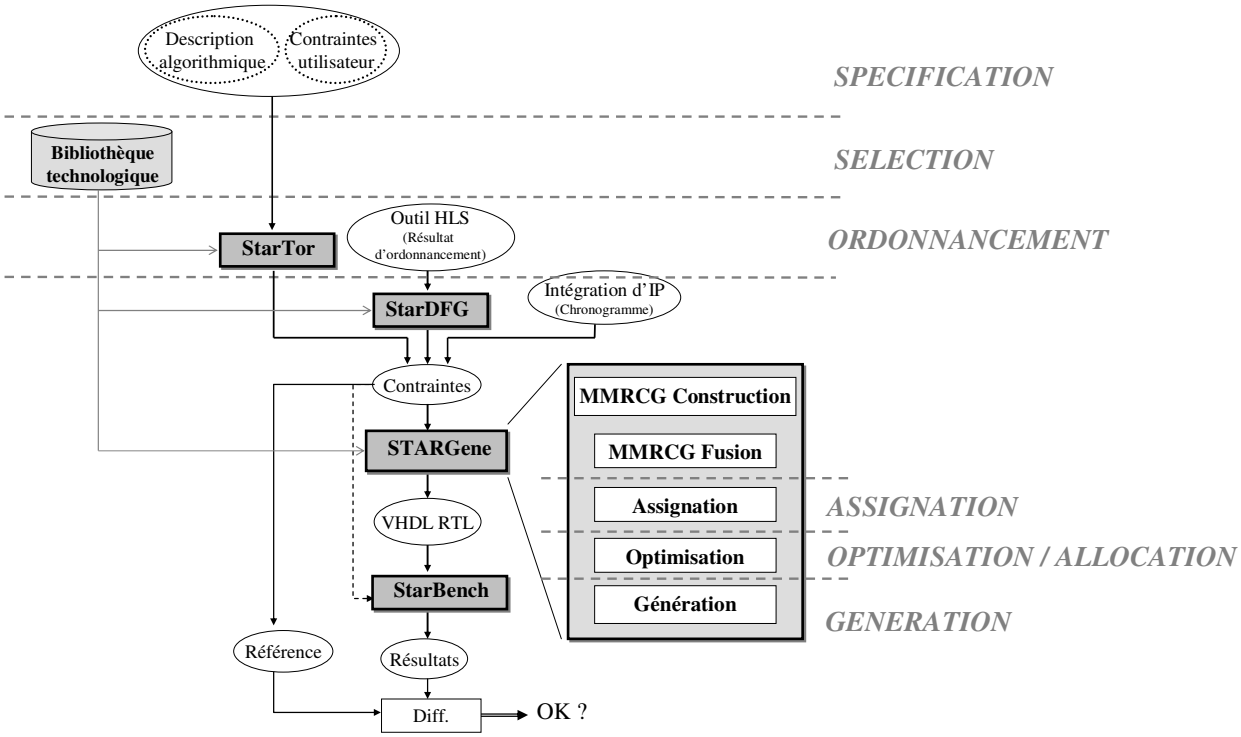


Figure 4.9. *Le flot de synthèse STARSystem*

Si l'on considère notre approche, la ***sélection*** correspond à la sélection par l'utilisateur de bibliothèques d'éléments mémorisants (FIFOs, LIFOs et Registres) avec les caractéristiques propres (nombre de cycles d'une lecture, d'une écriture…). L'étape d'***ordonnancement*** permet quant à elle de définir la durée de vie des données, comme nous allons le voir par la suite. De cette étape, il est alors possible de tirer les contraintes de communication requises pour pouvoir exploiter notre flot de synthèse. L'étape d'***assignation*** consiste à assigner les données dans des éléments mémorisants sous contrainte de temps et en minimisant les ressources utilisées. Enfin, l'étape d'optimisation définit le

nombre d'éléments mémorisants qui devront être alloués dans l'architecture finale. Ces étapes sont représentées sur la Figure 4.9 qui montre également le flot de conception que nous proposons.

Notre flot de synthèse génère un composant STAR sous contrainte de temps, en cherchant à minimiser les ressources matérielles utilisées. La Figure 4.9 montre un ensemble constitué de quatre outils logiciels. Selon les cas d'utilisation de notre approche, tous ne seront pas utilisés :

- *Synthèse à partir d'une description algorithmique* : l'utilisateur ne dispose que d'une description de haut niveau du schéma de communication des données (e.g. modèle C d'un algorithme d'entrelacement). L'outil ***StarTor*** va analyser cette description et va en extraire les contraintes de communication (Un diagramme temporel). Pour ce faire, StarTor procède à une étape d'ordonnancement de la trace de description fonctionnelle du schéma de communication pour pouvoir déterminer la durée de vie des données dans le STAR (cf. Section 2.1).
- *Synthèse d'un chemin de données* : dans ce cas, notre flot s'intègre dans un flot de synthèse de haut niveau (e.g. GAUT) et va exploiter le modèle CDFG utilisé par ce dernier pour extraire les contraintes de communication correspondantes. C'est l'outil ***StarDFG*** qui réalise cette analyse, après ordonnancement du/des CDFG(s) par l'outil de synthèse de haut niveau (cf. Section 2.2).
- *Synthèse à partir d'un chronogramme* : si l'utilisateur dispose des contraintes de communication du système qu'il souhaite intégrer (e.g. Chronogramme de communication des composants virtuels [COU02]), il lui suffit de les fournir directement à notre flot de synthèse sous le format défini pour les contraintes de communication (cf. Annexe D).

Ensuite, les étapes d'assignation, d'optimisation et de génération du VHDL RTL sont réalisées par l'outil ***STARGene***. Celui-ci va dans un premier temps modéliser les contraintes de communication sous la forme d'un modèle MMRCG. Puis, après fusion des modes de fonctionnement (dans le cas d'architectures multi-modes), STARGene va assigner les données dans des éléments mémorisants (FIFO, LIFO, Registres) et générer le modèle VHDL de niveau transfert de registres de l'architecture STAR. L'exploration du modèle MMRCG est basée sur une heuristique exploitant un ensemble de métriques permettant de prendre en compte : les contraintes technologiques, le coût de l'architecture générée, les problématiques de consommation (Etudes en cours).

La dernière étape de ce flot consiste à valider fonctionnellement l'architecture obtenue. Pour ce faire, l'outil ***StarBench*** génère automatiquement un banc de test (ou *testbench*) à partir des contraintes de communication. Il est ensuite possible de valider par simulation l'architecture STAR à l'aide de ce testbench et d'outils classiques comme ModelSim [MOD07] ou NCsim [CAD07]. Les résultats issus de cette simulation de l'architecture peuvent ensuite être comparés au comportement théorique de l'architecture (obtenu à partir des contraintes de communication), cf. Figure 4.9.

1.2.2. Métriques d'exploration architecturale

De l'assignation des données dans des éléments mémorisants et de l'optimisation de l'architecture obtenue, dépend le coût de l'architecture générée en termes d'éléments mémorisants. Afin d'obtenir une architecture globale optimisée, une fonction de coût est utilisée pour guider ces étapes

d'assignation et d'optimisation. Le critère de décision est une somme pondérée des différents coûts suivants :

- *Taille minimale/maximale des FIFOs/LIFOs* : éviter l'assignation d'éléments de mémorisation (FIFO ou LIFO) trop petits ou trop grands (Taille de structure en nombre de case mémoire). En utilisant ce paramètre, l'utilisateur peut limiter la taille des éléments mémorisants FIFO/LIFO afin de respecter des contraintes technologiques qui lui sont propres (e.g. taille de RAM maximum de 1024 éléments).
- *Taux d'utilisation moyen des FIFOs/LIFOs* : l'utilisateur peut définir une contrainte minimale pour le taux d'utilisation moyen d'un élément mémorisant (pour déterminer si celui-ci doit être assigné ou non). Cette métrique est calculée en dénombrant les données qui seront mémorisées dans l'élément pendant une itération du système (e.g. une itération d'entrelacement). On détermine ensuite le taux d'utilisation moyen de cette structure en calculant le ratio entre sa taille et le nombre de données qui transitent par cet élément mémorisant (e.g. si une FIFO de 64 cases manipule 274 éléments, son taux d'utilisation moyen de 4,28 données/case mémoire).
- *Taux de remplissage des FIFOs/LIFOs* : l'utilisateur peut définir un taux de remplissage minimal des FIFOs/LIFOs. En effet, les tailles des FIFOs/LIFOs générées sont des puissances de 2. Ainsi, si au maximum une FIFO doit mémoriser, à un instant donné, 52 des données, alors une FIFO de 64 cases mémoire sera générée (Taux de remplissage : 81%). Cette métrique permet ainsi de limiter le nombre de cases mémoires non utilisées des structures FIFOs/LIFOs dans l'architecture finale.
- *Facteur de complexité de multiplexage* : au cours de l'exploration de l'architecture, des données provenant de différentes sources (e.g. ports d'entrée synchronisés) peuvent être assignées au même élément mémorisant. De même, plusieurs éléments mémorisants peuvent envoyer des données vers un même port de sortie. Dans ces deux situations, il sera nécessaire d'implémenter des éléments de multiplexage dans notre architecture. Toutefois, dans [CHE06], les auteurs indiquent que le coût des éléments de multiplexage devient critique pour des multiplexeurs complexes. C'est pourquoi, afin d'éviter la génération d'un réseau d'interconnexion trop complexe, les algorithmes d'assignation et d'optimisation (cf. Figure 4.9) prennent également en compte la complexité du multiplexage.

Chacune de ces métriques est pondérée par l'utilisateur au moyen de coefficients spécifiques. L'utilisateur peut également décider (selon ses propres critères technologiques) de favoriser l'assignation de structures FIFO ou LIFO en accordant un poids plus important aux chemins de compatibilités FIFO ou LIFO.

Cette fonction de coût permet de définir un coût qui guidera l'heuristique d'exploration des chemins. Ce score est calculé selon la formule suivante :

- *Soit* M *l'ensemble des métriques d'exploration et* $m_i \in M$ *une métrique,*
- *Soit* C *l'ensemble des coefficients associés aux métriques et* $c_i \in C$ *le coefficient associé à la métrique* m_i,
- *Soit* P *un chemin de compatibilité dans un MMRCG et* $m_i(P)$ *le résultat de l'application du calcul de la métrique* m_i *sur le chemin* P,

$$score = \frac{\sum_{i=1}^{n} c_i * m_i(P)}{\sum_{i=1}^{n} c_i}$$

L'ensemble de ces métriques élémentaires va nous permettre de composer (1) une fonction de coût locale à chaque chemin de compatibilité identifié sur un MMRCG pour l'étape d'*assignation*, et (2) une fonction de coût locale à chaque chemin de compatibilité identifié pour l'étape d'*optimisation*. Il est également possible pour l'utilisateur de limiter l'exploration de l'espace des solutions architecturales à un sous ensemble d'éléments mémorisants (e.g. en désactivant l'assignation d'éléments FIFOs, l'algorithme ne va assigner que des LIFOs et des Resgistres).

2. Contraintes de communication

Le flot que nous avons présenté est basé sur la description des échanges de données sous la forme de contraintes de communication. Un tel fichier de contraintes contient des informations relatives :

- A la bibliothèque d'éléments mémorisants retenue (contraintes technologiques, étape de sélection).
- Au nombre de ports d'entrées/sorties (contraintes architecturales).
- Aux durées de vie des données (contraintes temporelles, étape d'ordonnancement).
- Aux différents transferts de données depuis les ports d'entrée, vers les différents ports de sortie (contraintes fonctionnelles et architecturales).

Ces informations sont naturellement présentes pour tous les modes de fonctionnent de l'application dans le cas d'architectures multi-modes. Un tel ensemble de contraintes de communication contient toutes les informations nécessaires à la modélisation des échanges de données sous la forme d'un MMRCG.

Selon l'origine de ces informations (CDFG, description algorithmique ou chronogramme), nous utilisions différentes approches pour générer un fichier de contraintes de communication (cf. Figure 4.9).

2.1. Règle de brassage de données

Dans le cas où les schémas d'échange de données sont modélisés par une description algorithmique (e.g. un entrelaceur décrit en langage C) il est possible, après analyse, d'extraire les informations relatives à la durée de vie des données, sous réserve de procéder à une étape de transformation.
Pour ce faire, nous proposons une étape de conception (et un outil associé appelé ***StarTor***) qui va extraire de la description algorithmique les ordres d'échange des données (e.g. ordre d'entrée dans l'entrelaceur et ordre de sortie de l'entrelaceur), puis procéder à une étape transformation (ordonnancement) pour déterminer les dates d'écriture/lecture des différentes données.

Algorithme de transformation
Nous illustrons ici le principe de l'algorithme de transformation que nous utilisons sur un exemple pédagogique. Supposons qu'un algorithme d'entrelacement requiert de mélanger huit données selon la règle suivante :

- Ordre des données en entrée : {a, b, c, d, e, f, g, h}
- Ordre des données en sortie : {b, d, g, f, h, a, e, c}

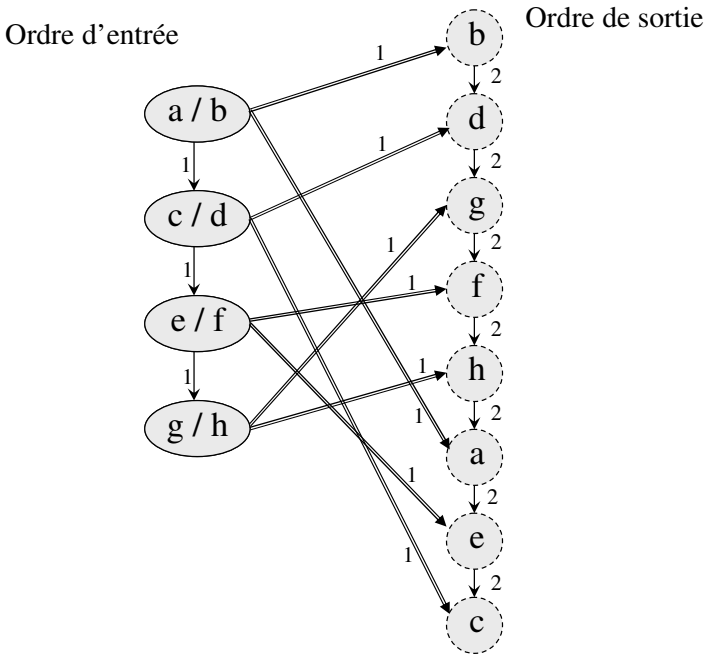


Figure 4.10. *Ordres d'entrée et de sortie des données*

Supposons que l'utilisateur ait spécifié des débits différents sur les ports d'entrée et de sortie de l'entrelaceur : les données entrent deux par deux à chaque cycle dans l'entrelaceur (i.e. sur un port sur lequel deux données sont transférées en parallèle) et ressortent une par une tous les deux cycles. La Figure 4.10 montre une trace de lecture et une trace d'écriture de la description algorithmique décrivant cette règle d'entrelacement. Les nœuds représentent les données et les arcs les relations de précédence entre les données. La contrainte de débit en entrée (en sortie) est également spécifiée sur les arcs sous la forme d'un délai inter nœuds de lecture (d'écriture). Le temps d'accès minimum aux éléments mémorisants (contrainte technologique) est spécifié sur les arcs (en traits doubles) reliant les nœuds de lecture et les nœuds d'écriture.

Les nœuds en trait plein représentent les données à leur en entrée dans l'entrelaceur. Les nœuds en trait pointillé représentent ces mêmes données à leur sortie de l'entrelaceur (émission). Par souci de lisibilité, dans la suite de ce document nous ne représenterons que les arcs représentant les contraintes de temps d'accès pertinentes pour les données étudiées.

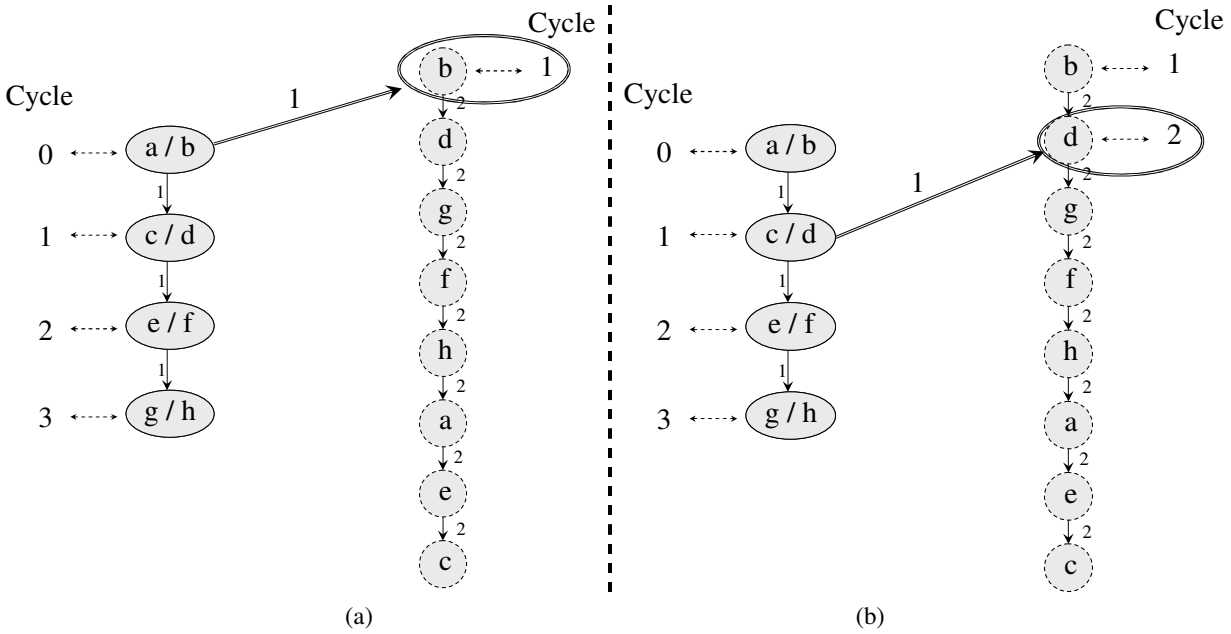


Figure 4.11. *Ordonnancement des données d'entrée et report sur les données de sortie*

L'étape suivante (cf. Figure 4.11.a) consiste à ordonnancer les données d'entrée (dans les cycles 0 à 3). Il est ensuite possible d'ordonnancer les données de sortie. Ainsi la donnée *b* sortira au plus tôt au cycle 1 (ordonnancement ASAP, *As Soon As Possible*) puisqu'elle est entrée dans l'entrelaceur au cycle 0.

> En effet, pour cet exemple nous postulons qu'au minimum un cycle doit séparer l'entrée d'une donnée dans le STAR de sa sortie. Ce délai, que nous nommerons ***temps d'accès***, est un paramètre spécifié par l'utilisateur, en fonction des contraintes technologiques propres aux éléments de mémorisation utilisés. Par défaut, et c'est le cas dans cet exemple, ce paramètre est fixé à un cycle et est représenté grâce aux poids des arcs entre les séquences d'entrées et de sortie des données.

Ensuite, puisqu'un cycle d'horloge est au minimum nécessaire entre l'arrivée de la donnée *d* et sa sortie, alors cette dernière est ordonnancée au cycle 2 (cf. Figure 4.11.b). Ainsi, le postulat stipulant qu'il doit y avoir un minimum d'un cycle entre l'entrée d'une donnée dans l'entrelaceur et sa sortie est respecté, puisque que cette donnée est entrée dans l'entrelaceur au cycle 1.

Toutefois, la contrainte de débit spécifié par l'arc reliant les données *b* et *d* n'est pas respectée. Au mieux, la donnée *d* peut être ordonnancée au cycle 3 (cf. Figure 4.12.a).

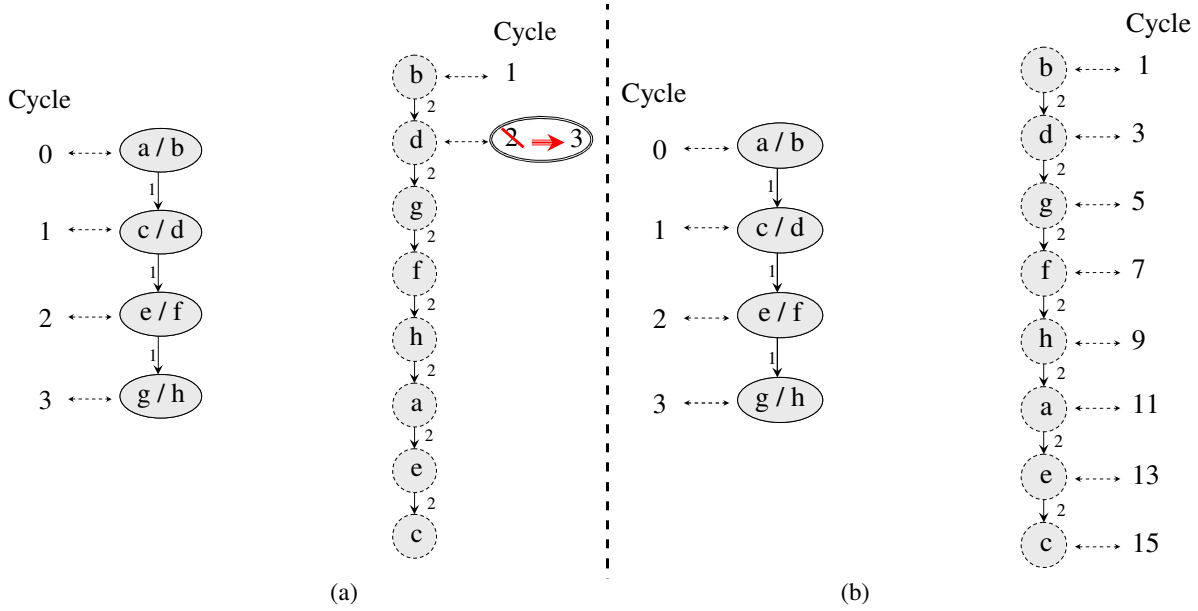


Figure 4.12. *Impact de la contrainte de débit sur l'ordonnancement*

Les données de sortie sont donc ordonnancées selon la règle suivante :

Soient d *une donnée et* C_d *le cycle d'émission de la donnée* d,
Soit c_{IN} *le cycle d'entrée de la donnée* d,
Soit c_{OUT} *le cycle d'émission de la donnée sortie juste avant l'émission de* d,
Soit T *la contrainte technologique de temps d'accès,*
Soit D *la contrainte de débit de sortie en nombre de cycle entre deux émissions,*

$$C_d = \text{Max}\{ (c_{IN} + T) ; (c_{OUT} + D) \}$$

Le même principe est appliqué pour l'ordonnancement de toutes les données de sortie (cf. Figure 4.12.b). Au final, nous obtenons pour chaque donnée sa durée de vie dans l'entrelaceur. Par exemple, la donnée *a* doit être mémorisée du cycle 0 au cycle 11.

En comparant (cf. Figure 4.13) le nombre maximum de données à mémoriser et la latence totale du cycle acquisition/temporisation/émission, résultant de notre algorithme d'ordonnancement (trait plein sur la courbe de la Figure 4.13) avec une solution classique (trait en pointillés) n'autorisant pas de recouvrement entre ces trois étapes (acquisition, temporisation et émission), nous constatons que notre approche permet d'offrir potentiellement plus de solutions d'optimisation (gain maximum de deux points mémoire sur cet exemple) pour le flot de conception qui sera utilisé par la suite. Il est même possible pour l'utilisateur, d'exploiter cette analyse pour déterminer la latence minimale pour réaliser cet algorithme d'entrelacement (ici 15 cycles au minimum).

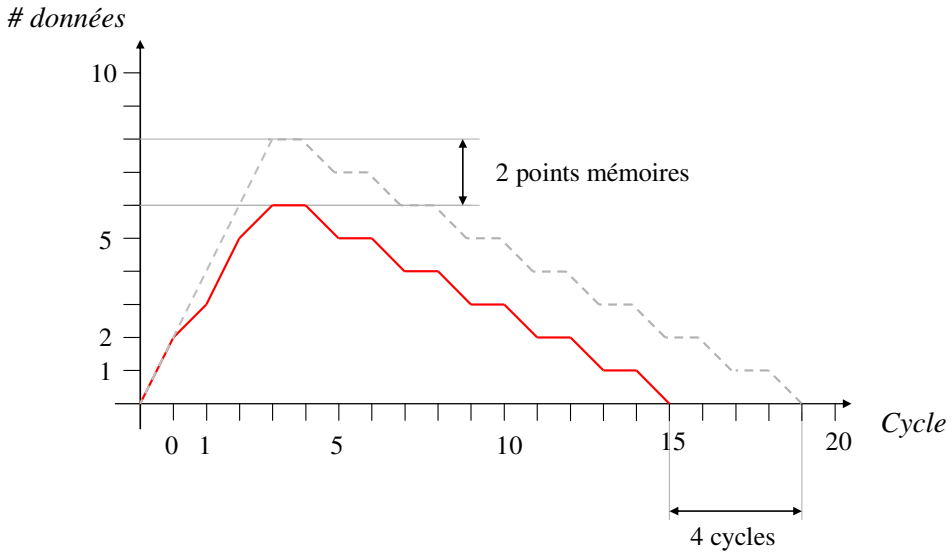


Figure 4.13. *Bilan de l'ordonnancement*

Avec une contrainte de débit plus forte en sortie (cf. Figure 4.14**,** 1 donnée par cycle), notre approche permet de proposer un ordonnancement qui ne dégrade pas les performances en termes de points mémoires et qui réduit la latence totale des échanges (ce qui peut permettre d'augmenter le débit applicatif sans avoir recours à une architecture pipeline).

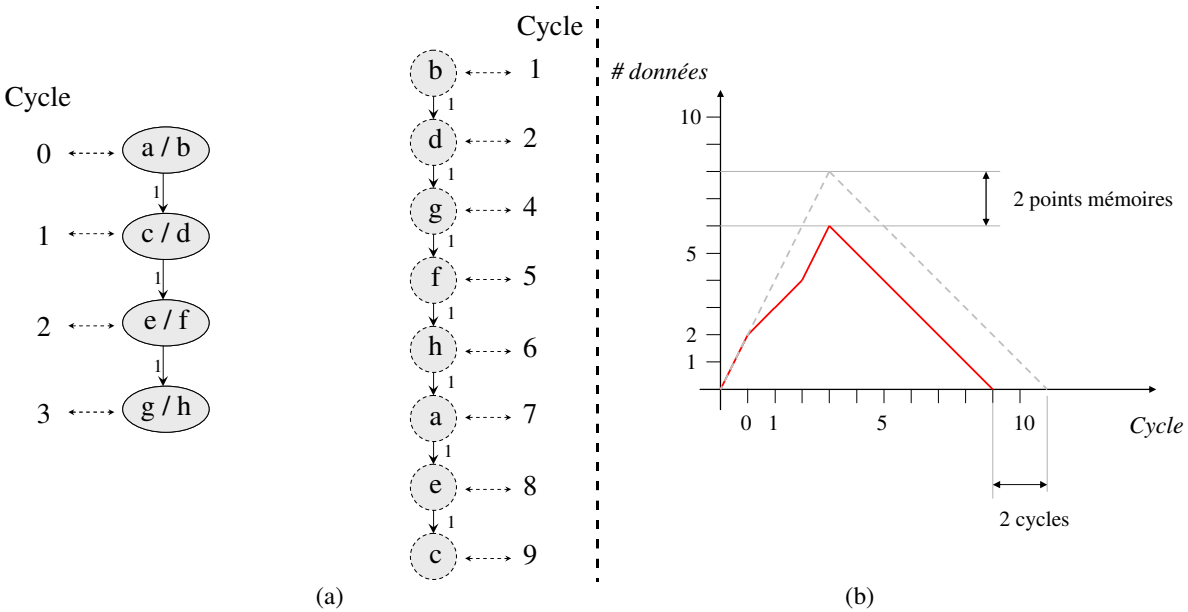


Figure 4.14. *Bilan de l'ordonnancement avec une contrainte de débit plus forte*

Ainsi sur notre exemple, il est possible d'utiliser au maximum six points mémoire pour temporiser les huit données (gain mémoire maximum de deux données), et une latence minimale de neuf cycles. Toutefois, le débit des données en sortie n'est plus continu. La raison en est que pour déterminer le cycle auquel ordonnancer une sortie, la fonction va choisir le maximum entre : la date respectant la contrainte de débit de sortie (i.e. $c_{OUT}+D$, dans la formule présentée précédemment), et la contrainte de

temps d'accès (i.e. $c_{IN}+T$, dans la formule présentée précédemment). Ainsi, si cette seconde date est sélectionnée, alors le délai entre deux émissions sera supérieur à la contrainte de débit de sortie.

Il est toutefois possible de lisser le débit de sortie en procédant à une étape de rétro-propagation de la contrainte de débit (cf. Figure 4.15). Cette rétro-propagation ne dégrade pas les performances en terme de latence, ni en terme de nombre de points mémoires. De même, la contrainte de temps d'accès sera respectée. En effet, les données de sortie seront alors donc ordonnancées selon la règle suivante :

Soient d_0 *une donnée et* C_{d_0} *le cycle d'émission de la donnée* d_0,

Soient d_1 *une donnée et* C_{d_1} *le cycle d'émission de la donnée* d_1,

Soit D *la contrainte de débit de sortie en nombre de cycle entre deux émissions,*

$$C_{d_1} - C_{d_0} \geq D$$

Donc,

$$C_{d_1} - D \geq C_{d_0}$$

Ainsi, la procédure de lissage ne peut violer ni les contraintes de débit, ni les contraintes de temps d'accès.

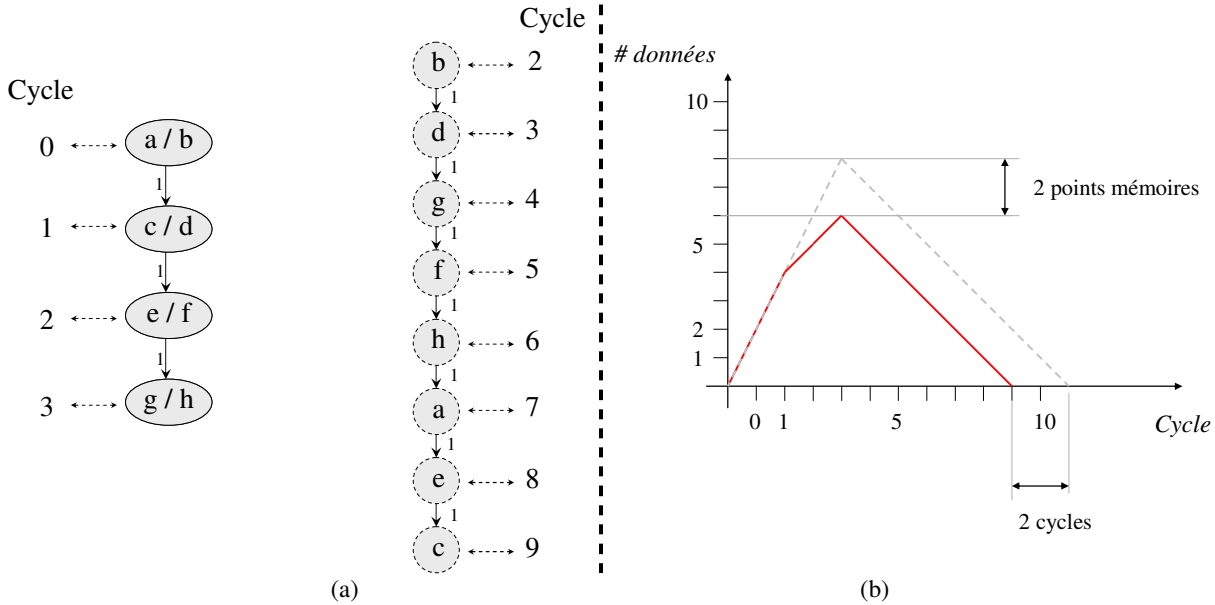


Figure 4.15. *Bilan de l'ordonnancement avec une contrainte de débit plus forte*

Au final, le fichier de contraintes de communication contient alors les informations relatives à la durée de vie des données, mais également les ports d'entrées et de sorties du composant STAR sur lesquels elles doivent transiter. Ces éléments sont nécessaires pour décrire chacun des modes de fonctionnement attendus de l'architecture (cf. Annexe D).
La Figure 4.16 présente l'algorithme permettant d'ordonnancer les échanges de données décrit par la fonction d'entrelacement.

Algorithme: Ordonnancement avec recouvrement (OR)

Entrée:

- Un ensemble de données d'entrée non ordonnancées : D_e ;
- Un ensemble de données de sortie non ordonnancées : D_s ;
- Un délai minimum entre l'acquisition et l'émission d'une donnée de T cycles ($T \in \mathbb{N}$) ;
- Une contrainte de débit d'entrée E ;
- Une contrainte de débit de sortie S ;

Sortie:

- Un ensemble de données d'entrée ordonnancées : Do_e ;
- Un ensemble de données de sortie ordonnancées : Do_s ;

Début

Ordonnancement des données de D_e par ordre d'arrivée dans le STAR sous contrainte de débit d'entrée E;

// Soit G_i *un ensemble de donnée de* D_s *devant être ordonnancée au même cycle* i*;*

// Soit $LastD_i$ *la donnée de* G_i *ordonnancée la plus tard pendant l'ordonnancement de* D_e*;*

Ordonnancement du premier ensemble de données G_0 au cycle $LastD_0+T$;

// Soit CyclePrec *un entier dans* $\mathbb{N}$

$CyclePrec = LastD_0+T$;

Pour tout G_i dans D_s

Si ($LastD_i+T < CyclePrec+S$) **alors**

Ordonnancement de l'ensemble de données G_i au cycle $LastD_i+T$;

$CyclePrec = LastD_i+T$;

Sinon

Ordonnancement de l'ensemble de données G_i au cycle $CyclePrec+S$;

$CyclePrec = CyclePrec+S$;

Fin Si

Fin Pour

// Si le concepteur souhaite lisser le débit des données en sortie

Si (*Lissage Demandé*) **alors**

Lissage(Do_s, S);

Fin Si

Fin

Figure 4.16. *Algorithme de transformation*

2.2. CDFG ordonnancé

L'outil ***StarDFG*** exploite des DFGs ayant été préalablement ordonnancés et assignés par un outil de synthèse de haut niveau. Un graphe flot de données issu d'un outil de synthèse de haut niveau contient, après ordonnancement et assignation des opérateurs de calcul, toutes les informations requises pour générer un fichier de contraintes de communication. En effet, après analyse d'une telle représentation, il est possible de déterminer comment sont produites puis consommées les données, par quels opérateurs, de déduire ainsi la durée de vie de ces données.

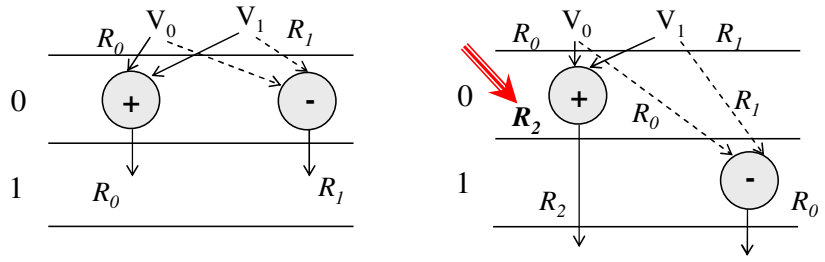


Figure 4.17. *Impact de l'ordonnancement sur la durée de vie d'un registre et sur le nombre de registre*

L'exemple présenté Figure 4.17 montre l'impact de l'ordonnancement sur la durée de vie des données et donc sur le nombre de registres qui devront être utilisés. En effet, sur cet exemple pédagogique, on considère un fragment de DFG dans lequel deux opérations (une addition et une soustraction) sont ordonnancées selon deux politiques différentes.
Nous pouvons remarquer que si l'addition et la soustraction sont ordonnancées dans le même cycle (*cycle 0*, cf. Figure 4.17.a), alors deux registres (R_0 et R_1) suffisent pour mémoriser les données V_0 et V_1 , et les résultats des opérations (réutilisation de R_0 et R_1). Si toutefois la soustraction était ordonnancée au cycle suivant (*cycle 1*, cf. Figure 4.17.b) les valeurs V_0 et V_1 devraient être maintenues un cycle de plus, augmentant donc la durée de vie de ces données. Il serait alors nécessaire d'utiliser un registre supplémentaire (R_2) pour mémoriser le résultat de l'additionneur.

Impact de l'assignation lors de la fusion de modes
Considérons l'exemple présenté dans la Figure 4.18. Soient DFG_3 et DFG_4 deux DFGs ordonnancés représentant deux modes de fonctionnent d'une architecture, supposons que l'assignation des opérateurs ait déjà été réalisée pour le cycle S0.

Si le concepteur souhaite partager les ressources de calcul (Opérateurs O_i et N_j), il faut assigner les opérations des deux modes aux mêmes opérateurs. Ainsi, supposons que les opérations (O_0, O_1, O_3) et (N_0, N_1, N_3) partagent les mêmes opérateurs. L'idée est d'assigner dans S1 les opérations (O_2, O_5) du DFG_3; et les opérations (N_2, N_4), du DFG_4 aux mêmes opérateurs.

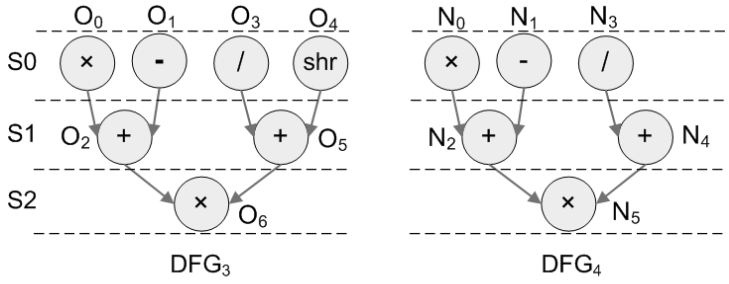


Figure 4.18. *Impact de l'assignation*

Si les opérations O_5 et N_2 sont assignées au même élément de calcul, Figure 4.19.a, alors nous pouvons constater qu'il faut alors utiliser deux multiplexeurs. En revanche, si le partage des opérateurs est fait pour les opérations O_2 et N_2 , il n'est dans ce cas plus nécessaire d'utiliser de multiplexeur (cf. Figure 4.19.b).

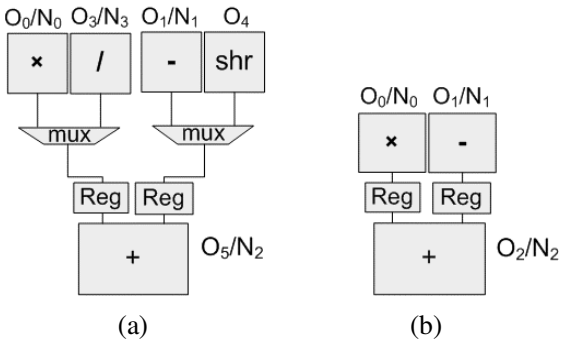


Figure 4.19. *Assignation des opération O_5 et N_2*

Comme le montrent ces exemples pédagogiques, les étapes d'ordonnancement et d'assignation ont donc un impact direct sur la complexité de l'architecture générée (mono ou multi-modes). Ainsi, à partir des informations contenues dans les CDFGs (après ordonnancement et assignation), l'outil StarDFG peut extraire les contraintes de communication exprimant les échanges de données du système.

Notre approche, corrélée à des travaux ciblant l'amélioration des algorithmes d'ordonnancement et d'assignation comme ceux menés par C. Andriamissaina, permet l'exploration architecturale de chemins de données optimisés, notamment dans le cadre de la génération d'architectures multi-modes.

3. Flot de conception de l'outil StarGene

Une fois que les contraintes de communication ont étés définies, l'outil STARGene peut alors les modéliser sous la forme d'un MMRCG et explorer l'espace des solutions architecturales par le biais du flot de synthèse présenté Figure 4.20. Cet outil représente à lui seul un peu plus de 20 000 lignes de code C.

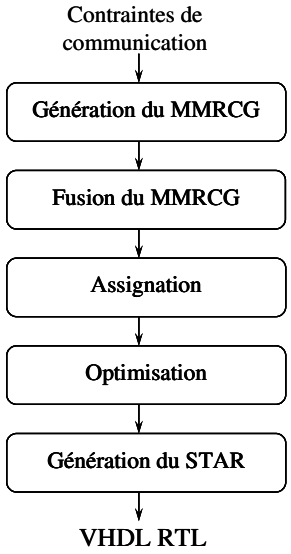


Figure 4.20. *Flot de synthèse de l'outil STARGene*

Ce flot de synthèse se décompose en cinq étapes :

- Construction d'un MMRCG.
- Fusion des différents modes de fonctionnement selon les principes énoncés dans le chapitre III (dans le cas des systèmes multi-modes).
- Assignation des données dans des éléments mémorisants.
- Optimisation de l'architecture obtenue.
- Génération de la description VHDL RTL et des scripts de synthèse pour l'outil Design Compiler Ultra (DCUltra) de Synopsis.

3.1. Décomposition du flot de synthèse

Dans cette partie, nous présentons en détails les différentes étapes du flot de synthèse de l'outil STARGene. Pour ce faire, nous utiliserons un exemple pédagogique de système multi-modes. Dans cet exemple, les contraintes de communication (ici les contraintes temporelles sont présentées sous la forme de chronogrammes, cf. Figure 4.21) supposent de modéliser deux comportements (cf. Figure 4.22).

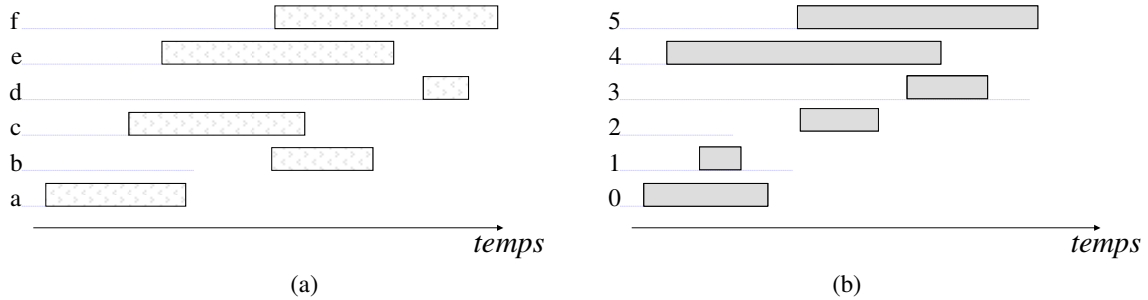


Figure 4.21. *Deux modes de fonctionnement différents à fusionner dans un composant STAR*

Pour clarifier la démonstration, nous supposerons que toutes ces données transitent par un seul et même port d'entrée et par un seul et même port de sortie, et que les ports sont communs pour les deux modes de fonctionnement.

3.1.1. Modélisation MMRCG

Les deux comportements de l'architecture sont modélisés sous la forme de deux RCGs qui constituent les éléments feuilles d'un modèle MMRCG (cf. Figure 4.22).

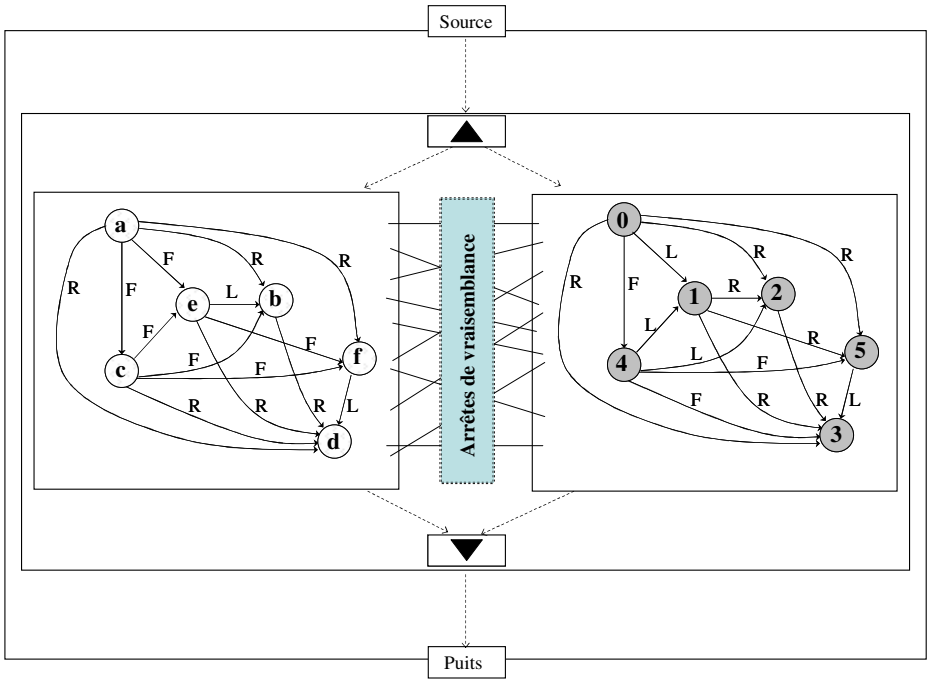


Figure 4.22. *MMRCG résultant des contraintes de communication*

Pour des raisons de lisibilité, nous n'avons pas représenté toutes les arrêtes de vraisemblance créées dans ce MMRCG.

3.1.2. Fusion des modes de fonctionnement

Comme exposé dans le chapitre III, les arrêtes de vraisemblance vont nous permettre de fusionner les différents modes (i.e. les différents RCGs). L'algorithme de fusion va ainsi regrouper des nœuds variables issus des différents RCGs dans des nœuds multi-variables/multi-configurations. Pour ce

faire, cet algorithme exploite les poids assignés aux arrêtes de vraisemblance. Ces poids sont calculés selon une heuristique qui a pour but d'optimiser le coût de la fusion selon deux critères :

- Un *coût local de fusion* : calculé en fonction de deux métriques, (1) le nombre arcs de compatibilité supprimés ou transformés par la fusion des nœuds reliés par l'arrête de vraisemblance (cf. chapitre III), et (2) l'impact en terme de multiplexage de la fusion de ces nœuds.
- Un *coût global de fusion* : calculé en fonction de l'impact de la fusion des nœuds sur l'espace total des solutions de fusion. En effet, nous avons montré, dans le chapitre III, que la fusion de nœuds variables entraînait la suppression d'arrêtes de vraisemblance non pertinentes.

Algorithme: Multi-mode binary graph merging algorithm (BGMA)

***Entrée**:*
- Un MMRCG : G;

***Sortie**:*
- Un MMRCG fusionné: G_F;

Début

Sélection de l'arc de vraisemblance (l_i) de plus faible poids dans G;
Fusion des noeuds reliés par (l_i) dans un nœud multi-variables/multi-modes mN_i;
Supprimer toutes les arêtes de vraisemblance non pertinentes dans G;
// Soit $pastGN_i$ le passé de mN_i dans G;
// Soit $nextGN_i$ le futur de mN_i dans G;
BGMA($pastGN_i$);
BGMA($nextGN_i$);

Fin

Figure 4.23. *Algorithme de fusion des modes d'un MMRCG*

Dans le pire cas, l'algorithme de fusion ne pourra exploiter le critère de pertinence des arêtes de vraisemblance pour réduire l'espace des solutions de fusion à explorer (Dans ce cas, $pastGN_i$ ou $nextGN_i$ sont vides). L'algorithme de fusion devra alors fusionner tous les noeuds. Le nombre d'appel récursif sera donc au pire cas de n, si n est le nombre maximum de données à fusionner.

Admettons que l'algorithme de fusion génère un MMRCG fusionné tel que présenté Figure 4.24. Il est alors possible d'explorer l'espace des solutions architecturales pour la génération d'un composant STAR à partir de ce modèle.

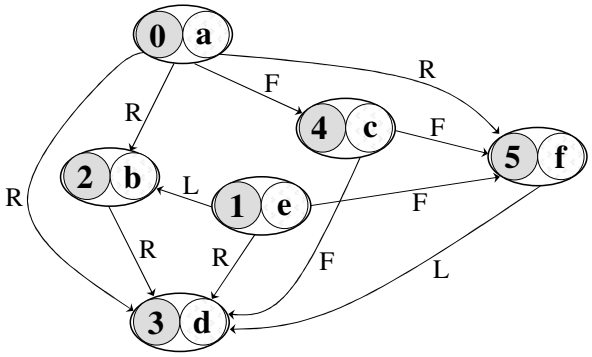


Figure 4.24. *Exemple de fusion des modes – Vision simplifiée du MMRCG résultant*

3.1.3. Assignation et optimisation

Après fusion des modes de fonctionnement, le modèle MMRCG est utilisé pour explorer l'espace des solutions. Le principe est le suivant : en premier lieu, l'algorithme cherche à assigner autant de FIFO et de LIFO que possible sur le MMRCG. Puis cette première architecture est ensuite optimisée en mutualisant les éléments mémorisants lorsque cela est possible (éléments de type compatibles et relié par des arcs de compatibilité Registre)

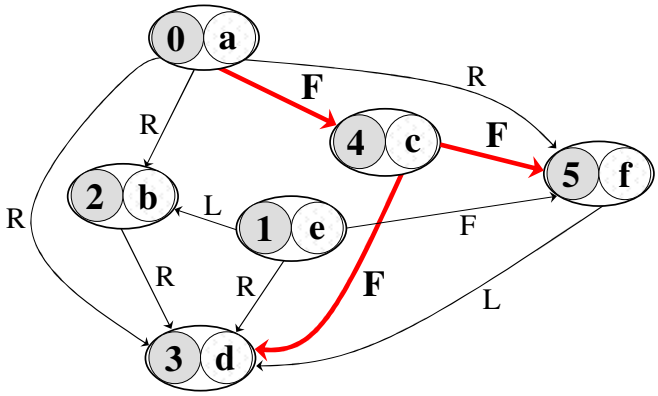


Figure 4.25. *Différentes solutions pour l'assignation d'un chemin FIFO*

L'observation du graphe de la Figure 4.25 monte qu'il est possible d'identifier par exemple deux chemins FIFO. Le premier suppose la création d'une FIFO regroupant les noeuds *0a*, *4c* et *3d* ; et le second regroupe les noeuds *0a*, *4c* et *5f*.

Admettons que notre heuristique indique que le meilleur chemin soit le chemin assignant les noeuds *0a*, *4c* et *5f* dans une même FIFO. Dans ce cas, comme le montre la Figure 4.26, une structure FIFO est ajoutée au graphe. Les arcs de compatibilité reliant cette structure au reste du MMRCG sont mis à jour de la façon suivante : la durée de vie de l'élément créé est définie pour chaque mode de fonctionnement. A partir de ces durées de vie, il est alors possible de créer des arcs de compatibilité entre la structure créée et les autres nœuds du MMRCG. Le cas échéant, ces arcs pourront être transformés selon les lois de transformation que nous avons présentées dans le chapitre précédent. Dans notre exemple (cf. Figure 4.26), l'insertion du nœud FIFO détruit tous les arcs le reliant aux autres nœuds du graphe.

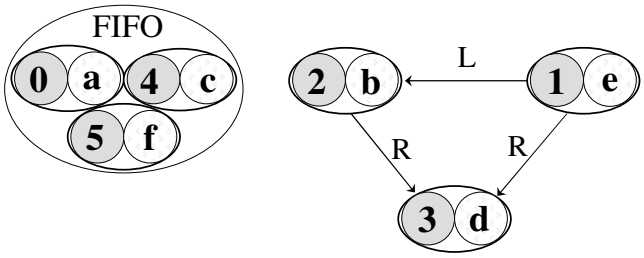


Figure 4.26. *Création et ajout d'une structure FIFO dans le MMRCG*

Tant qu'il reste des arcs de compatibilité FIFO ou LIFO dans le MMRCG, l'algorithme va continuer à assigner des données dans des éléments FIFO ou LIFO. Ainsi le MMRCG représenté dans la Figure 4.26 possède un arc de compatibilité LIFO reliant les nœuds *2b* et *1e*. Ces deux nœuds vont donc être assignés à une structure LIFO (cf. Figure 4.27).

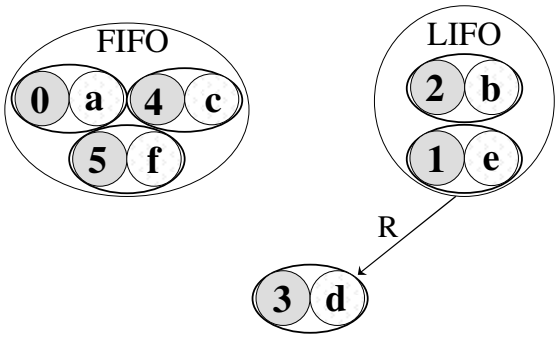


Figure 4.27. *Création et ajout d'une structure LIFO dans le MMRCG*

Les lois de transformation ont cette fois permis de conserver une relation de compatibilité Registre entre la LIFO et le nœud *3d* restant. Toutefois, il n'y a plus de compatibilité FIFO ou LIFO. L'étape d'assignation est donc terminée.
L'étape d'optimisation va chercher à mutualiser au maximum les ressources. Ainsi, l'algorithme va identifier les meilleurs chemins (Selon l'heuristique présentée section I.2.2) de compatibilité Registre sur le MMRCG. Sur cet exemple, il n'existe qu'un chemin de compatibilité Registre. Le nœud *3d* va ainsi être assigné dans la LIFO (cf. Figure 4.28).

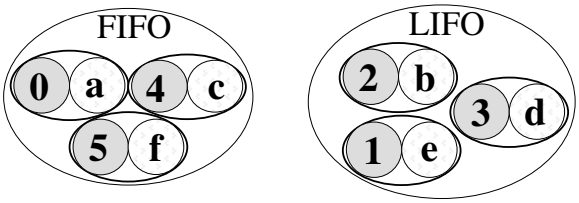


Figure 4.28. *MMRCG final après optimisation*

A partir de ce modèle MMRCG final (Figure 4.28), il est possible de générer un composant STAR. Dans le cas présent, l'architecture du STAR utilisera une FIFO de profondeur trois, mémorisant trois données, et une LIFO de profondeur trois, mémorisant trois données. De même, la complexité du contrôleur est réduite. En effet, ces deux éléments ont des comportements totalement déterministes, et deux signaux de commande (*Push*/*Pop*) sont suffisants pour les piloter. Ainsi, sur cet exemple, 4

signaux seront utilisés pour le contrôle au lieu de 6 signaux dans le cas où six registres seraient utilisés.

Remarques

L'algorithme d'optimisation que nous venons de présenter utilise, comme nous l'avons indiqué section I.2.2, une heuristique pour générer l'architecture finale. Toutefois, puisque cet algorithme exploite des relations de compatibilité Registre pour mutualiser les ressources, cela revient à dire qu'il analyse les durées de vie des éléments pour ensuite fusionner ceux dont ces durées de vie sont non recouvrantes

Ce principe est le même que celui exploité par l'algorithme d'optimisation dit "*Left-Edge*", [HAS71]. Nous avons donc choisi de proposer également cet algorithme d'optimisation à l'utilisateur. A titre de comparaison, l'algorithme Left-Edge que nous proposons permet, en général, de réduire le nombre de points mémoires utilisés, mais le coût en termes d'éléments de multiplexage dans l'architecture résultante est plus important, que le coût obtenu avec notre algorithme d'optimisation basé sur le calcul d'une heuristique.

Les solutions d'assignation et d'optimisation que nous proposons exploitent toutes des algorithmes gloutons. Elles s'exécutent donc en temps polynomial.

3.1.4. Génération du VHDL RTL

Une fois le MMRCG optimisé, la dernière étape du flot de synthèse consiste à projeter ce modèle sur une description architecturale en VHDL de niveau transferts de registres (RTL). Cette projection est facilitée par la structure même du modèle MMRCG. En effet, les éléments mémorisants constitutifs de l'architecture sont directement identifiés sur le modèle MMRCG. De même, la génération des parties contrôle intégrant les différents modes de fonctionnement est également rendu possible par la présence de nœuds variables colorés suivant les modes de fonctionnement auxquels les données appartiennent.

L'outil de synthèse logique que nous ciblons est DCUltra de Synopsys, et nous utilisons également les bibliothèques technologiques internes de STMicroelectronics. StarGene génère également les scripts destinés à synthétiser la composant STAR avec cet outil.

Remarque

Nous avons constaté que selon la syntaxe retenue pour implémenter les machines de contrôle, il était possible de réduire le coût en surface de l'architecture finale de 10 à 15 %. Ainsi, si l'on génère un contrôleur qui assigne un mot de contrôle dans une structure de type *if-then-else*, la surface totale sera inférieure à celle d'un contrôleur construit autour d'une FSM classique (Mot de commande modifié selon l'état courant). Toutefois, les temps de synthèse avec DCUltra augmentent alors de façon exponentielle, et deviennent rapidement inacceptables. Un exemple détaillé est proposé en annexe C.

3.2. Génération d'architectures pipelines

L'outil de synthèse comportementale GAUT cible les applications de traitement du signal et de l'image (TDSI). Les contraintes que supporte le flot de raffinement automatique sont de différente nature : temporelle (cadence), mémoire (mapping), date consommation/production des E/S (latence). Cet outil de synthèse génère des architectures potentiellement pipelines et optimisées en surface.

Principe des architectures pipelines

L'outil GAUT synthétise des applications TDSI sous contrainte de débit exprimée sous la forme d'une cadence d'itération. Cette valeur notée $T_{Cadence}$, correspond à la plage temporelle qui sépare l'arrivée des données d'une itération, de celle de l'itération suivante. Le temps d'exécution d'une itération de l'algorithme est définit par la phase d'ordonnancement et est noté $T_{Latence}$. En fonction du couple ($T_{Cadence}$, $T_{Latence}$), l'architecture générée sera ou non pipeline.

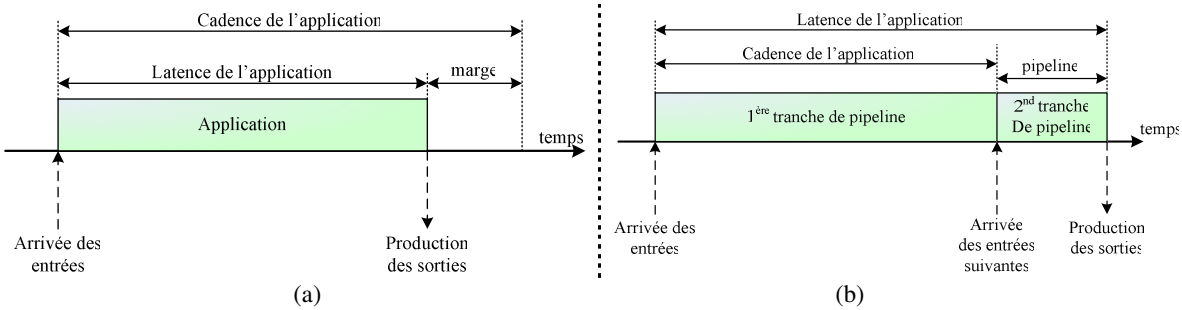


Figure 4.29. *Comparaison latence/cadence pour la création d'un pipeline*

Lorsque la contrainte de cadence liée à l'application est supérieure à la durée de la latence, cela entraîne la génération d'une architecture permettant la complétion de l'application (production des sorties) avant l'arrivé d'un nouveau jeu d'entrée (cf. Figure 4.29.a).

Lorsque la cadence de l'architecture est inférieure à la latence de l'architecture comme cela est représenté dans la Figure 4.29.b, l'architecture générée sera de type pipeline pour satisfaire la contrainte temporelle de débit spécifiée par le concepteur. A la fin de chaque itération, les données dont la fin de la durée de vie n'est pas atteinte sont vieillies, passant de la tranche *n* à la tranche *n+1*.

Génération d'un composant STAR avec gestion du pipeline

La génération d'une architecture STAR pipelinée suppose de disposer d'éléments mémorisants pouvant être utilisés à travers plusieurs tranches de pipeline. Ainsi, si l'on considère l'exemple d'une donnée a_i entrée dans le STAR pendant la $i^{ième}$ tranche de pipeline. Si, à l'arrivée de cette donnée, la donnée a_{i-1} est toujours présente dans le STAR, alors il sera nécessaire d'utiliser deux points mémoire pour mémoriser a_i et a_{i-1} (cf. Figure 4.30).

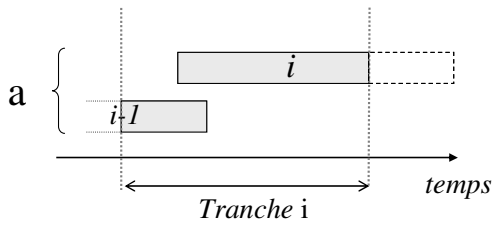


Figure 4.30. *Durée de vie d'une donnée recouvrant deux tranches de pipeline*

L'utilisation de notre approche dans le cadre de la génération de chemins de données pipeline nous a conduit à définir une première approche permettant de proposer une solution pour ce type d'architecture. L'idée est de modéliser et de considérer une donnée *a* dont la durée de vie s'étend sur *n* tranches de pipeline (e.g. a_{i-1} et a_i de la figure précédente) comme des données distinctes dans le modèle MMRCG.

Bien que cette solution ne permette pas d'utiliser une FIFO, elle n'entraîne pas de surcoût en nombre de points mémoire. Dans tous les cas, le STAR doit de toute façon mémoriser deux données dans deux points mémoires distincts. Cette approche suppose que la durée de vie de la donnée *a* soit modifiée : on passe d'une donnée *a* avec une durée de vie propre, à un ensemble $\{a_i\}$ de données avec *n* durées de vie (*n* étant défini comme la partie entière supérieure de la durée de vie de *a* sur la $T_{Cadence}$).
L'idée est d'extraire la donnée *a* du STAR au temps $T_{Cadence}$ et de considérer ensuite dans l'architecture finale que la donnée sortant sur ce port est ensuite directement réinjectée dans la composant STAR (cf. Figure 4.31).

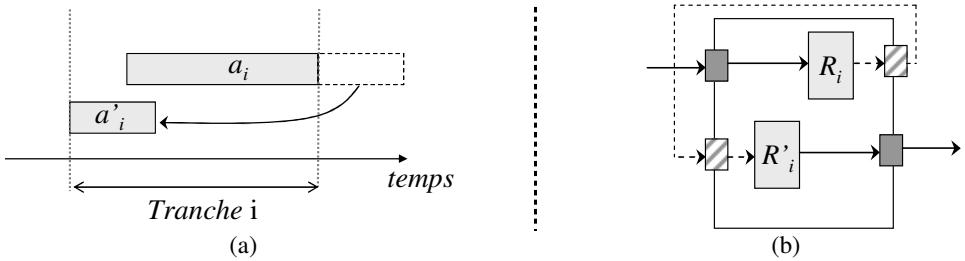


Figure 4.31. *Transfert de donnée dans le cas d'une architecture pipeline*

On impose ainsi des transferts de données supplémentaires qui peuvent entraîner un surcoût en termes de contrôle et de consommation en comparaison à une architecture pipeline utilisant une FIFO. En effet, comme le montre la Figure 4.31.b, l'architecture finale contiendra deux registres (R_i et R'_i). Nous pouvons également constater que nous avons ajouté deux ports (ports hachurés) pour transférer la donnée ; c'est deux ports sont directement reliés l'un à l'autre dans l'architecture finale.

Il ne s'agit toutefois là que d'une solution architecturale au problème de la génération d'une architecture pipeline. Une approche théorique reste à formaliser : le principe est de considérer la contrainte de cadence dans le modèle MMRCG a priori de la synthèse et non a posteriori.

Remarques

Cette problématique n'a toutefois pas été explorée sur le plan théorique dans le cadre de cette thèse. Les travaux de Caaliph Andriamisaina (doctorant LESTER) permettront, à court terme, d'étendre l'architecture STAR et le flot associé pour la génération d'architectures pipelinées.

4. Validation de l'architecture STAR

La validation de l'architecture se fait par simulation. Pour ce faire, l'outil StarBench génère automatiquement un banc de test en considérant les contraintes de communication qui ont été utilisées pour la génération du composant STAR.
Ce banc de test permet après simulation de récupérer les résultats de la simulation du composant STAR généré (les sorties) et de les analyser selon deux critères :

- L'ordre de sortie des données.
- Les dates de sortie des données.

Ces éléments sont ensuite comparés avec le comportement théorique de l'application tel que défini dans le DFG, dans le chronogramme ou dans la description algorithmique.
Cette analyse permet de valider le comportement temporel, ainsi que le comportement fonctionnel de l'architecture STAR générée.

5. Extension du modèle architecturale : SAGE

Dans le chapitre II, nous avons présenté un état de l'art des méthodologies de conception des architectures pour des entrelaceurs. Comme nous l'avons évoqué, ces solutions peuvent se classer en trois catégories :

- *Entrelaceurs non standards* : le concepteur doit définir sa propre règle d'entrelacement pour pouvoir ensuite exploiter une architecture d'interconnexion simplifiée et sans conflit [GNA03a].
- *Entrelaceurs standards avec résolution des conflits à l'exécution* : le concepteur peut exploiter des entrelaceurs standardisés mais la résolution des conflits suppose d'ajouter de la mémoire dans le réseau d'interconnexion [WHE04].
- *Entrelaceurs standards avec résolution des conflits à la conception* : les conflits d'accès sont cette fois résolus lors de la conception de l'architecture, par le biais d'un algorithme de mapping des données en mémoire relativement complexe [TAR04].

Dans le cadre de ces travaux de thèse, le besoin de pouvoir générer des entrelaceurs basés sur des architectures classiques à base de mémoire RAM (en lieu et place d'un composant STAR) est apparu. Pour ce faire, nous avons cherché à développer une solution permettant de générer un entrelaceur exploitant un réseau d'interconnexion simplifié (comme dans [GNA03a]), en se basant une technique de placement des données en mémoire permettant de ne pas ajouter de mémoire supplémentaire dans l'architecture, et ce pour tous standards de communication (comme dans [TAR04]) et avec un algorithme simple (comme dans [WHE04]).

La solution que nous proposons (***S**tatic **A**ddress **G**eneration **E**asing*, SAGE) répond à ces attentes et a fait l'objet d'un dépôt de demande de brevet auprès de l'INPI, [CHA07a]. Le principe est de réaliser un placement des données en mémoire, avec pour objectif la génération d'un réseau d'interconnexion régulier le plus simple possible.

Pour des raisons de confidentialité, cette approche n'a pour l'instant pas fait l'objet d'une publication. Ainsi, nous ne développerons pas plus avant cette solution dans ce manuscrit.

SAGE a toutefois déjà été appliqué dans le cadre d'une collaboration entre l'équipe HLS de STMicroelectronics dont je faisais partie, et une équipe de développement de l'une des divisions clients de la société. Le prototype logiciel développé pour l'occasion à permit de valider les concepts clefs de notre solution.

6. Bilan

Dans ce chapitre nous avons présenté l'architecture retenue pour notre composant d'adaptation des communications. Le flot de synthèse associé, que nous proposons, permet de réaliser l'exploration de l'espace des solutions par le biais de métriques pouvant être paramétrées par l'utilisateur. Nous avons vu qu'en fonction de la provenance des informations temporelles à la disposition du concepteur (description algorithmique, CDFG ou chronogramme), notre flot propose différentes solutions méthodologiques et différents outils pour l'exploration et la synthèse de système mono ou multi-modes.

Nous avons également évoqué une autre approche développée dans le cadre de ces travaux de thèse pour la génération de composants d'entrelacement. Cette approche faisant l'objet d'une demande de brevet, elle n'a pas été détaillée dans le manuscrit.

Dans le chapitre suivant, nous démontrerons la validité du flot de synthèse proposé en analysant un ensemble de résultats expérimentaux.

Chapitre 5

EXPERIMENTATIONS

Ce chapitre est composé de trois expériences réalisées sur des applications du domaine du traitement du Signal, de l'image et des Télécommunications. La première montre l'intérêt de notre approche dans le cadre de l'intégration de composants virtuels. Nous montrons également l'impact des métriques d'exploration sur l'architecture obtenue. La deuxième expérience compare les résultats obtenus en appliquant notre approche pour la synthèse d'un composant de brassage de données, avec le même composant généré par un outil de synthèse de haut niveau du commerce. La dernière expérience présente les résultats de la synthèse de chemins de données multi-modes avec notre flot.

1. Introduction

Dans ce chapitre, nous exposons les résultats de synthèse d'adaptateurs de communication obtenus à l'aide du flot de conception et d'exploration proposé dans le cadre de ces travaux. Le choix des applications à synthétiser a été guidé par les expériences conjuguées de l'équipe FTM-HLS de la société STMicroelectronics et du laboratoire LESTER dans les domaines du traitement du signal, de l'image et des télécommunications.

Au travers d'un premier ensemble d'expérimentations, nous montrons l'importance des métriques pour l'exploration de l'espace des solutions. Ces expériences permettent également de démontrer l'intérêt de notre approche dans le cadre de l'intégration de composants virtuels comparée à des solutions classiques basées sur l'utilisation de blocs RAM.

Dans un second ensemble d'expériences, nous présentons un algorithme d'entrelacement utilisé dans le cadre d'un système Ultra-Wide Band (802.15.3a, [IEE07]). Ce type d'application constitue un parfait exemple de composant de brassage de données comme on en retrouve dans de très nombreuses applications de télécommunications. Cet exemple nous permet également de comparer l'architecture que nous générons avec une architecture de référence obtenue par l'équipe FTM-HLS à l'aide d'un outil de synthèse de haut niveau du commerce [CHA07c]. Cette expérience illustre la capacité que présentent notre approche et nos outils, à traiter des problèmes industriels de grande complexité.

Enfin le dernier ensemble d'expériences présente les résultats obtenus dans le cadre de la génération de chemins de données multi-modes. Nous utilisons pour cela un ensemble de fonctions devant être intégrées au sein d'une même architecture : une Transformée en Cosinus Discrète *(DCT)*, une Transformée de Fourier Rapide (*FFT*) et sa Transformée inverse (*IFFT*), un Filtre à Réponse Impulsionnelle finie (*FIR*). Les architectures obtenues à l'aide de notre approche ([CHA07b]) sont comparées avec les architectures multi-modes générées de façon naïve (simple agrégat des différents modes) et celles générées par une approche de synthèse dédiée à ce type de système, *SPACT-MR* [CHI05].

Toutes les expériences présentées dans ce chapitre ont été réalisées à l'aide de la suite StarSystem et de son interface graphique. Cette dernière [DUR07] répond à deux objectifs : proposer à l'utilisateur une interface permettant une prise en main rapide de la suite logicielle StarSystem (StarTor, StarDFG, StarGene et StarBench) et permettre l'analyse et la comparaison rapide des résultats de synthèse obtenus dans le cadre des expérimentations. Cette interface s'articule autour du flot de synthèse de la suite StarSystem (cf. chapitre IV) et est présentée en Annexe F.

2. Intégration de composants virtuels

La première série d'expériences que nous proposons se situe dans le cadre de l'intégration de composants virtuels. Considérons l'exemple pédagogique suivant : un concepteur dispose d'une chaîne de traitement du signal pour une architecture de type MPEG-2 [ISO94b]. Pour le développement d'une architecture de type JPEG2000 [ISO00], il souhaite pouvoir réutiliser une partie de son architecture, à savoir le bloc de quantification qui est commun aux deux applications.

2.1. Présentation des applications

Présentation d'une application MPEG2

MPEG-2 est la norme de seconde génération (1994) du Moving Picture Experts Group qui fait suite à MPEG-1. Elle définit les aspects de compression de l'image, et du son et le transport à travers des réseaux pour la télévision numérique. Ce format vidéo est utilisé pour les DVD et SVCD avec différentes résolutions d'image. Il est également utilisé dans la diffusion de télévision numérique par satellite, câble, réseau de télécommunications ou hertzienne.

L'algorithme de compression vidéo MPEG-2 atteint des taux de compression élevés en exploitant la redondance d'information dans une séquence d'images. La Figure 5.1 représente le synoptique de l'architecture d'un encodeur MPEG-2. Cette norme supprime les redondances temporelles et fréquentielle en codant les différences entre les images (cf. Figure 5.2.a).

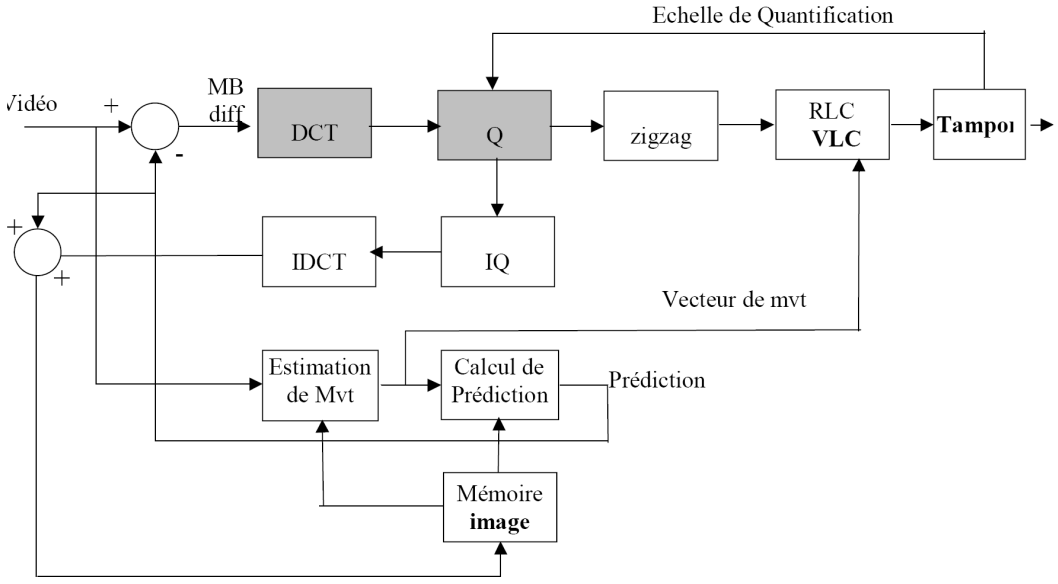


Figure 5.1. *Schéma synoptique d'un encodeur MPEG-2*

Une image est composée de trois matrices rectangulaires représentant les valeurs de luminance (*Y*) et de chrominances (*Cb* and *Cr*). Les images I (Intra) sont codées indépendamment (comme en JPEG). Les images *P* et *B* (Inter) sont quant à elles compressés par référence aux autres images I et P d'une même séquence.

L'unité de donnée dans une image est le macro-bloc. Un macro-bloc est composé d'une section de luminance de 16 pixels par 16 lignes, et des deux sections de chrominances 8 pixels par 8 lignes respectives (Figure 5.2.b). Un macro-bloc est décomposé en bloc 8x8 pixels sur lesquels s'opèrent tous les calculs dans la chaîne de compression.

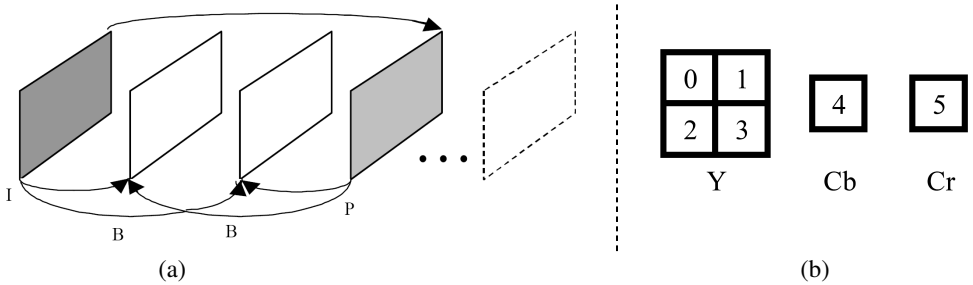


Figure 5.2. *Images MPEG-2*

Dans une architecture MPEG-2, chaque bloc sera traité par la DCT. Puis, ce bloc subira une étape de quantification permettant de réduire la quantité d'information à transmettre, et enfin les coefficients sont ensuite réorganisés (Bloc de zigzag) et encodés par le RLC (*Run-Length Encoding*).

Présentation de l'application JPEG2000

La chaîne de codage de JPEG-2000 (cf. Figure 5.3) commence par un prétraitement de l'image originelle. Ce prétraitement permet de changer la représentation de l'image (facultatif), tel qu'une transformation de couleur RGB à YUV. Ensuite, cette image est décomposée en sous blocs (ou *tile*). Chacune de ces tuiles subit une transformée en ondelettes [DAU98].

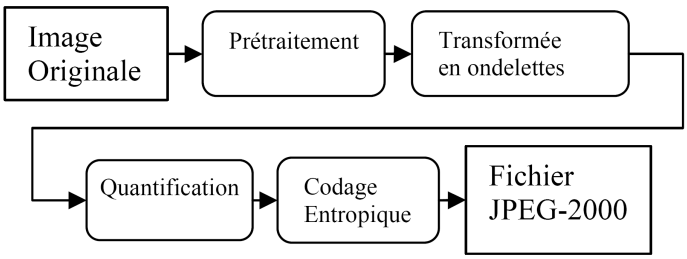


Figure 5.3. *Chaîne de codage JPEG-2000*

Une quantification est par la suite appliquée sur chaque sous bande obtenue suite à la transformée en ondelettes (TO). Les coefficients obtenus sont alors traités par le codeur entropique qui permet d'obtenir les données compressées. Ces données sont finalement mises en forme pour aboutir à un fichier JPEG-2000.

2.2. Expérimentations

Problématique

Le bloc de quantification utilisé dans l'architecture MPEG-2 est conçu pour recevoir et traiter les données issues de la DCT. Cette dernière traite des données matricielles ligne par lignes (cf. Figure 5.4).

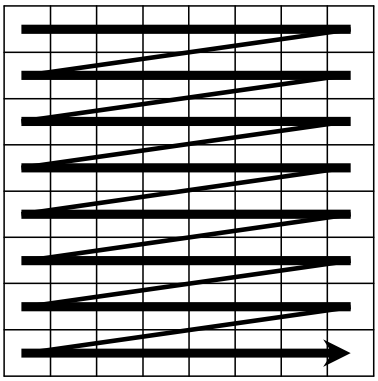


Figure 5.4. *Parcours d'une matrice par la DCT*

Le bloc de quantification a donc été construit et optimisé pour traiter les données d'une matrice transmise ligne par ligne.

Le composant virtuel utilisé pour implémenter la transformée réalise une ondelette 9/7 implémentant le "Lifting scheme" [DAU98]. Ce composant travaille sur des blocs 8x8 et calcule progressivement la TO en consommant les entrées deux pixels par deux pixels suivant un ordre Z fractal (Figure 5.5). Les résultats sont produits pixel par pixel dans l'ordre décrit dans la Table 5-1.

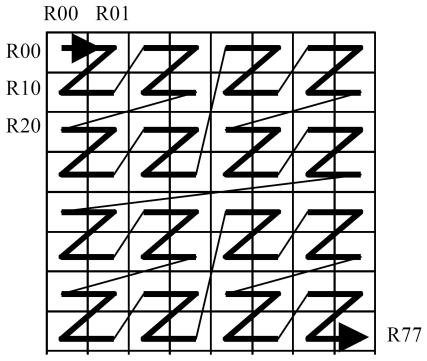


Figure 5.5. *Lecture Z fractal*

Table 5-1

Séquence de résultats produite par la TO

R72	R02	R43	R05	R74	R16	R50	R21
R03	R01	R06	R04	R17	R13	R22	R20
R47	R10	R50	R14	R65	R24	R66	R27
R11	R07	R15	R12	R25	R23	R30	R26
R77	R32	R67	R35	R77	R51	R73	R54
R33	R31	R36	R34	R52	R44	R55	R53
R70	R40	R71	R45	R75	R57	R76	R63
R41	R37	R46	R42	R60	R56	R64	R61

Si l'on réutilise ce bloc de quantification dans une architecture JPEG2000 où les données issues de la TO sont transmises selon une matrice parcourue différemment (cf. Table 5-1), dans ce cas il est nécessaire d'adjoindre un composant réalisant l'adaptation des échanges de données entre la TO et la quantification.

Nous souhaitons mettre en exergue l'exploration de l'espace des solutions architecturales, rendue possible dans notre approche par l'exploration des métriques. De fait, en fonction des paramétrages définis par l'utilisateur (réglage des métriques, choix des algorithmes d'optimisation), les architectures résultantes sont différentes les unes des autres.
Le flot de conception utilisé (cf. Figure 5.6) exploite les chronogrammes de fonctionnement (Modèle IPERM) des blocs *TO* et *quantification* pour générer les contraintes de communications du système.

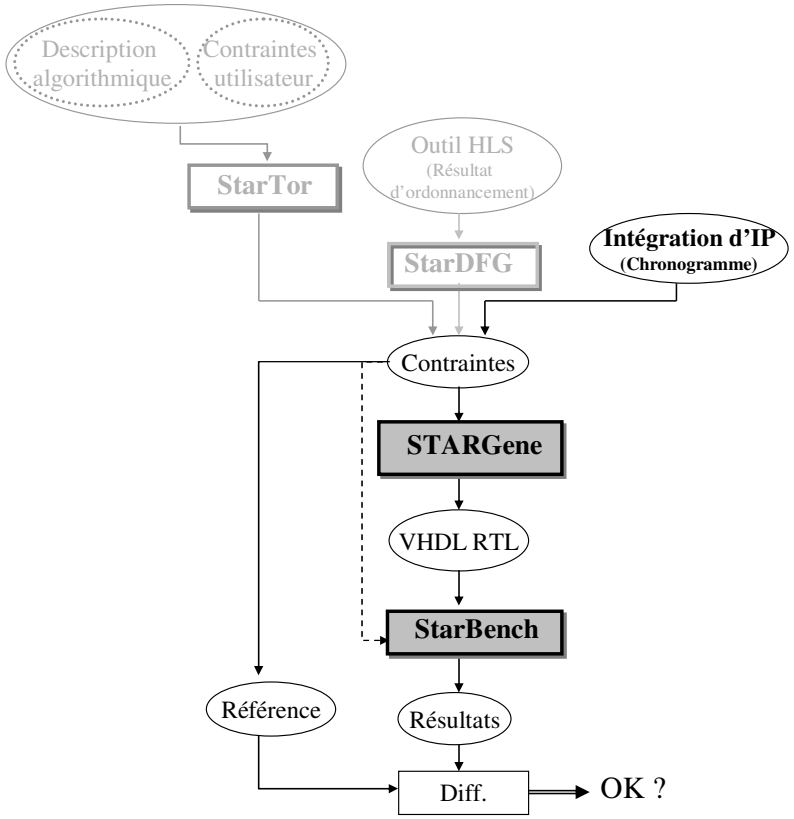


Figure 5.6. *Flot de conception utilisé*

Le système, tel qu'il est présenté plus haut, suppose que la quantification accepte ses entrées pixel par pixel. Nous avons donc généré un adaptateur transmettant les données une par une à la quantification. Toutefois, dans le cadre de ces expérimentations, nous avons souhaité explorer l'espace des solutions pour différents débits applicatifs (de 4,7 à 9,4 Mb./s.), en incluant un cas plus général où la quantification accepterait ses données deux pixels par deux pixels.
Rappelons que du point de vue du bloc de quantification, le composant STAR ne doit pas introduire de latence supplémentaire, mais s'adapter au comportement attendu des entrées /sorties.

Table 5-2

Synthèse d'un adaptateur de communication STAR

Paramètres			Registres			FIFO/LIFO/Registres				
IN	OUT	Nb Cycles	Nbre Reg	Nbre Mux	Nbre Pts Mem	Nbre FIFO	Nbre LIFO	Nbre Reg	Nbre Mux	Nbre Pts Mem
1	1	123	60	0	60	7	4	1	0	62
2	1	123	64	0	64	7	4	0	8	63
2	1	92	62	0	62	8	3	3	8	62
2	2	61	60	2	60	6	7	3	10	60

La première série de résultats (cf. Table V.2), présente les architectures obtenues lors de la synthèse d'un composant STAR pour différentes contraintes de débit. Ce tableau synthétise plusieurs informations relatives à l'architecture générée : le nombre de données transmises en parallèle dans l'architecture en entrée (*IN*) et en sortie (*OUT*) et le nombre de cycles nécessaire pour la temporisation et la transmission des données échangées.
Les architectures obtenues sont ensuite détaillées pour deux types d'architectures STAR : (1) une architecture STAR composée uniquement de registres, et (2) une architecture STAR à base de FIFOs, de LIFOs et de Registres. Le nombre de multiplexeurs (*MUX*) est donné en nombre de mux2 :1 équivalent. Enfin nous reportons également le nombre de points mémoire total de l'architecture.

Nous pouvons constater que le nombre de structures à piloter est plus important dans le cas d'une architecture à base de registre. L'architecture STAR classique (à base de FIFOs, LIFOs et Registres) permet donc de réduire la complexité du contrôleur pour un nombre de points mémoire équivalent (sur cet exemple). Notons que lors d'une utilisation pertinente des FIFOs/LIFOs, à nombre de points mémoire équivalent, le chemin de données est plus petit.

Nous pouvons également constater qu'il est possible d'utiliser des critères pour augmenter le débit de l'application : (1) le parallélisme des entrées/sorties, et (2) la latence en nombre de cycles de l'application. Ainsi, nous pouvons constater que pour un même parallélisme d'entrée/sortie (2 données/cycle en entrée, 1 données/cycle en sortie) il est possible de réduire le nombre de cycles nécessaires (de 123 à 92 cycles) pour réaliser la temporisation et le réordonnancement des données.

Si l'on compare les architectures obtenues avec une architecture à base de mémoires RAM, les gains sont très importants. En effet, si nous considérons la première architecture (débit de 1 donnée/cycle en entrée et en sortie), alors une solution à base de RAM comme celle présentée dans le chapitre I, utilisera deux RAMs (en entrée et en sortie) de 64 cases mémoire, soient au total 128 points mémoire, ce qui représente plus du double des architectures STAR proposées.
Toujours avec un architecture à base de RAMs, si le concepteur souhaite augmenter le débit, nous avons constaté (cf. Chapitre I) qu'il devait augmenter le nombre de bancs. Ainsi, pour une architecture avec un débit de 2 données/cycle en entrée et en sortie, il faudra utiliser quatre blocs RAM de 64 cases, soient 256 points mémoire, ce qui représente plus de 4 fois le nombre de points mémoire utilisés avec l'une ou l'autre des architectures STAR.

Table 5-3
Architecture STAR générées en modifiant la métrique de *Multiplexage* et le *Taux de remplissage*

Paramètres			FIFO/LIFO/Registres				
IN	OUT	Nb Cycles	Nbre FIFO	Nbre LIFO	Nbre Reg	Nbre Mux	Nbre Pts Mem
1	1	123	6	3	7	0	64
2	1	123	6	3	4	6	64
2	1	92	8	2	10	6	62
2	2	61	5	5	11	7	63

Les métriques d'exploration et d'optimisation que nous avons présentées dans le chapitre IV permettent de générer des architectures qui seront optimisées selon différents critères. Ainsi, si un poids plus important est accordé au coût des multiplexeurs, les algorithmes d'exploration optimiseront les architectures obtenues en fonction de ce critère. Ainsi, la Table V.3 synthétise les informations concernant les architectures STAR générées en accordant un poids plus important au coût des multiplexeurs. Nous pouvons constater qu'en comparaison avec les précédentes expérimentations (cf. Table V.2), les architectures obtenues contiennent moins de multiplexeurs. Toutefois le nombre de points mémoire utilisés augmente.
De même, la contrainte de remplissage plus forte pour ces nouvelles expériences a un impact important (en moyenne +30%) sur le nombre de structures utilisées.

Nous pouvons également constater que les résultats obtenus montrent une prédominance des FIFOs sur les LIFOs. Il est possible d'accorder un poids plus important pour l'affectation d'un type d'éléments mémorisants. Nous allons donc analyser les architectures générées si l'on privilégie l'assignation d'éléments LIFOs (cf. Table V.4).

Table 5-4
Architectures STAR obtenues en privilégiant les éléments LIFOs

Paramètres			FIFO/LIFO/Registres				
IN	OUT	Nb Cycles	Nbre FIFO	Nbre LIFO	Nbre Reg	Nbre Mux	Nbre Pts Mem
1	1	123	4	5	3	0	64
2	1	123	2	7	4	7	64
2	1	92	3	8	6	9	62
2	2	61	2	5	4	8	63

L'analyse des résultats nous montre que l'utilisation de LIFOs ne permet pas de réduire le nombre de points mémoire utilisés, ni la complexité de l'architecture en terme de multiplexage.

Table 5-5
Architectures STAR à base de registre et optimisation *Left-Edge*

Paramètres			Registres		
IN	OUT	Nb Cycles	Nbre Reg	Nbre Mux	Nbre Pts Mem
1	1	123	60	0	60
2	1	123	62	4	62
2	1	92	61	8	61
2	2	61	58	8	58

Comme nous l'indiquions dans le chapitre IV, le concepteur peut décider d'utiliser un autre algorithme pour l'optimisation de l'architecture. Ainsi, l'utilisation d'un algorithme dit "*Left-Edge*" sur une architecture STAR à base de registre (cf. Table V.4), permet de réduire le nombre total de points mémoire utilisés (-2,5% en moyenne). Toutefois, nous pouvons constater que l'impact sur le nombre de composants de multiplexage est toutefois important (cf. Table V.2).

2.3. Conclusion

Ces expérimentations menées dans un cadre pédagogique d'intégration de composant pré conçus nous ont permis de démontrer la pertinence de la solution retenue pour l'adaptation des échanges de données. En comparaison d'une architecture à base de blocs RAM, les architectures obtenues permettent de réduire le coût en termes d'éléments mémorisants.

De plus, ces expérimentations nous ont permis de mettre en avant l'impact des différentes options d'optimisation sur l'architecture finale du composant STAR. L'architecture finale dépend donc fortement de l'importance relative accordée aux métriques et du choix des algorithmes d'optimisation. Ainsi, accorder une importance prépondérante à la métrique de complexité du multiplexage va permettre d'optimiser le nombre et la complexité de composants de multiplexage utilisés, au détriment du nombre total de points mémoire de l'architecture. Inversement, l'utilisation d'un algorithme d'optimisation *Left-Edge* permet de réduire le nombre de points mémoire utilisés, mais au détriment de la complexité de l'architecture de multiplexage.

L'exploration architecturale se fait donc bien en exploitant les métriques d'exploration proposées. La complexité de cette exploration nous amène (1) à envisager l'automatisation de l'exploration des métriques par des méthodes de programmation linéaire en nombre entiers par exemple, et (2) à mettre au point un algorithme *Left-Edge* prenant en compte la complexité de l'architecture de multiplexage.

Les expérimentations suivantes, menées sur des cas d'utilisation plus complexes, permettent également de mettre en avant l'impact des métriques sur l'architecture finale.

3. Entrelaceur pour l'Ultra-Wide Band

La seconde série d'expériences que nous présentons se situe dans le cadre de la génération de composants de brassage de données. Nous considérons l'exemple d'un entrelaceur pour l'Ultra Wide Band [IEE07] pour lequel nous disposons d'architectures de référence, issues de l'équipe FTM-HLS de STMicroelectronics, pour comparer nos résultats.

3.1. Présentation de l'application

Ces dernières années, les systèmes de communications large bande (Ultra-Wide Band, UWB) ont fait l'objet d'une grande attention de la part de l'industrie et de la recherche académique. En effet, ces technologies peuvent permettre d'atteindre des débits très importants (cf. Figure 5.7), à un coût réduit en termes de consommation et de surface. Plusieurs systèmes OFDM (pour ***O****rthogonal* ***F****requency* ***D****ivision* ***M****ultiplexing*, [FLO95]) ont ainsi été proposés pour implémenter des systèmes UWB par exemple [IEE07].

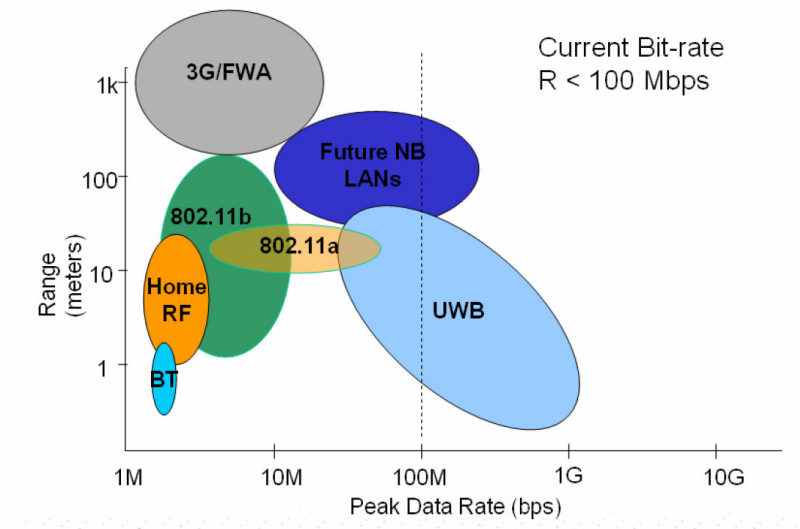


Figure 5.7. *Les différents systèmes WLAN /PLAN existant*

L'OFDM est une modulation de signaux numériques par répartition en fréquences orthogonales. Le principe de l'OFDM consiste à diviser sur un grand nombre de porteuses le signal numérique que l'on veut transmettre. Pour que les fréquences des porteuses soient les plus proches possibles et ainsi transmettre le maximum d'information sur une portion de fréquences données, l'OFDM utilise des porteuses orthogonales entre elles. Les signaux des différentes porteuses se chevauchent, mais grâce à l'orthogonalité, n'interfèrent pas.

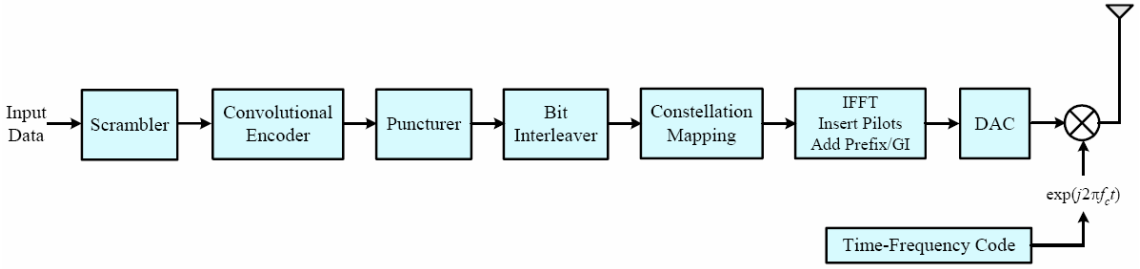


Figure 5.8. *Schéma bloc d'une architecture OFDM pour un émetteur*

C'est ce type de système qui est exploité dans le cadre de la norme WPAN-802.14.3a (*Wireless Personnal Area Network*, [IEE07]). Comme nous pouvons le constater (cf. Figure 5.8), cette architecture requiert l'utilisation d'un entrelaceur. La règle d'entrelacement utilisée est définie par les relations mathématiques suivantes :

$$S(i) = U\left\{\text{Floor}\left(\frac{i}{N_{CBPS}}\right) + (6/TSF) * \text{Mod}(i, N_{CBPS})\right\}$$

$$T(i) = S\left\{\text{Floor}\left(\frac{i}{N_{T\,\text{int}}}\right) + 10\text{Mod}(i, N_{Tint})\right\}$$

Les symboles N_{CBPS}, N_{Tint}, et *TSF* font respectivement référence au nombre de bits codés par symbole, à la taille d'un symbole et au facteur de dispersion temporel (*Time Spreading Factor*).

Le principe de l'entrelacement est le suivant :

- Un entrelacement des symboles entre eux
- Un entrelacement des bits au sein d'un symbole.

Au final, en sortie de l'entrelaceur, les bits ont été entrelacés dans les symboles et entre les différents symboles.
L'entrelaceur que nous ciblons doit pouvoir être utilisé dans trois modes de fonctionnement différents en fonction de la longueur de la trame à entrelacer (300, 600 ou 1200 bits). L'architecture générée doit respecter des contraintes de débit et de latence pour pouvoir s'intégrer dans le système cible. Pour des raisons de confidentialité, ces informations ne peuvent toutefois pas être divulguées.

3.2. Résultats expérimentaux

Un tel système UWB l'objet de développements au sein de l'équipe FTM-HLS de STMicroelectronics. Ainsi, nous disposons de résultats de synthèse de l'entrelaceur UWB, dans le cadre d'un projet industriel, pour comparer nos résultats. Ces synthèses ont été réalisées à l'aide d'un outil de synthèse de haut niveau du commerce. Suite à des accords de non divulgation, nous ne pouvons révéler dans ce manuscrit le nom de cet outil, ni les informations concernant les bibliothèques technologiques utilisées pour la synthèse, ni les surfaces précises obtenues.

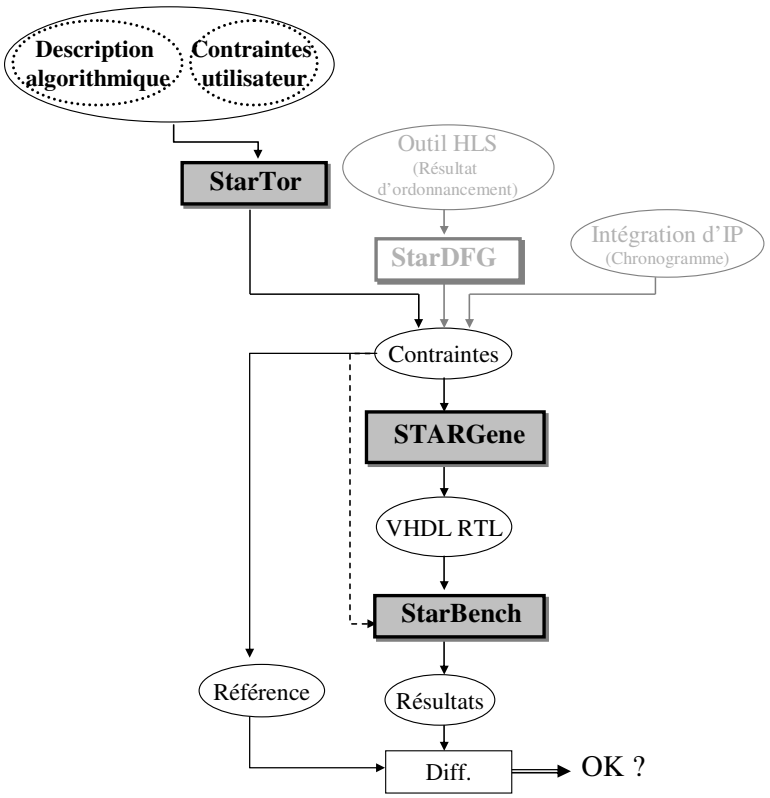


Figure 5.9. *Flot de synthèse utilisé*

Dans le cadre de ces expérimentations, la règle d'entrelacement nous a été fournie sous la forme d'un algorithme C. Nous avons donc exploité le flot de conception présenté Figure 5.9 : à partir du modèle

C et de contraintes utilisateur (débit, cadence) l'outil StarTor génère les contraintes de communications pour l'outil de synthèse StarGene. Nous avons ensuite procédé à l'exploration de l'espace des solutions pour chacun des modes de fonctionnement de l'entrelaceur. L'exploration des métriques et leurs impacts sur l'espace des solutions architecturales ayant été exposé, nous ne présenterons ici que les résultats finaux des expérimentations menées pour cet entrelaceur. Le lecteur intéressé trouvera un ensemble de résultat concernant l'exploration des métriques dans [CHA07d].

Les expériences suivantes ont étés menées en générant séparément les trois modes de fonctionnement de l'entrelaceur. Les sources VHDL obtenus ont ensuite été synthétisés avec l'outil DC-Ultra (Synopsis). L'étape de génération du code VHDL dans notre flot de synthèse multi-modes est en cours de développement. Nous ne présenterons donc pas ici les architectures d'entrelacement multi-modes générées.

Table 5-6

Comparaison des résultats de synthèse entre l'architecture de référence et trois composants STAR

Mode	Référence		F/L (Min 7, Tx 95%)		F/L (Min 15, Tx 90%)		Registres		Débit
	Gain mémoire	Ctrl	Gain mémoire	Ctrl	Gain mémoire	Ctrl	Gain mémoire	Ctrl	
300	0	300	56	77	60	240	60	240	434,8
600	0	600	83	101	130	470	130	470	438
1200	0	1200	96	117	120	609	168	1032	412,4

La Table 5-6 synthétise les résultats que nous avons obtenus au cours de cette expérience. L'architecture de référence a été générée par l'un des membres de l'équipe FTM-HLS à l'aide d'un outil de synthèse de haut niveau du commerce. Nous savons que cette architecture n'est pas optimisée en termes de points mémoire (aucun point mémoire économisé), et qu'elle est composée d'un ensemble de registres.

Cette architecture référence est comparée à trois architectures STARs. Les deux premières sont des architectures STARs classiques exploitant des composants FIFOs, LIFOs et Registres, générés avec des métriques d'exploration différentes (Longueur minimum d'un chemin typé, Taux d'utilisation moyen d'une structure) : dans le premier cas, la longueur minimum d'un chemin typé est fixé à 7 données et le taux d'utilisation moyen est fixé à 95% ; pour le second cas, la longueur minimum d'un chemin typé est fixé à 15 données et le taux d'utilisation moyen est fixé à 90% . La dernière architecture STAR générée n'utilise que des Registres (Génération de FIFO et de LIFO désactivées). Enfin, les débits obtenus (Exprimés en Mb./s.) pour les architectures STAR ont étés indiqués. Nous pouvons remarquer que pour un mode donné, quelle que soit la solution architecturale du composant STAR, le débit reste inchangé. Notre approche permet donc d'explorer efficacement l'espace des solutions tout en garantissant le respect de la contrainte de débit.

L'analyse de ces résultats montre que notre approche permet de réduire le nombre de points mémoire et le nombre d'éléments à piloter dans tous les cas, par rapport à l'architecture de référence.

Ainsi, pour la première architecture STAR avec FIFOs/LIFOs, le gain moyen en termes de signaux de contrôle est de 77% et en termes de points mémoire, le gain moyen est de 86%.

Dans le cas de la seconde architecture STAR avec FIFOs/LIFOs, le gain moyen en termes de signaux de contrôle est de 30% et en termes de points mémoire, le gain moyen est de 82%.

Enfin, dans le cas du composant ne comportant que des registres, le gain moyen en termes de signaux de contrôle est de 19% et en termes de points mémoire, le gain moyen est de 81%.

De même, si l'on considère les modes 300 et 600 données, nous pouvons constater que les architectures générées pour le second STAR à base de FIFOs/LIFOs, et le STAR n'utilisant que des registres, ont les mêmes caractéristiques. De fait, ces architectures sont identiques car les contraintes utilisées ne permettent pas d'assigner d'éléments FIFOs ou LIFOs, seul le mode 1200 données offre des chemins de compatibilités permettant d'assigner ce type de structures. Cela signifie que les éléments FIFOs et LIFOs assignés dans le cas de la première architecture STAR avec FIFOs/LIFOs pour les modes 300 et 600 données sont des éléments de petite taille.

Toutefois, si l'on compare la surface de l'architecture de référence intégrant les trois modes de fonctionnement, et la surface cumulée des composants STAR à base de registres générés pour chacun des modes, alors la surface totale de l'architecture de référence est de 14% supérieure à la surface cumulée des architectures STAR [CHA07c].

3.3. Conclusion

Ces expérimentations menées dans le cadre de la génération d'architectures d'entrelacement mono-mode ont démontré la pertinence de l'approche STAR en comparant les composants générés avec une architecture de référence connue, et obtenue à l'aide d'un outil de synthèse de haut niveau du commerce.
Les résultats obtenus démontrent que notre approche permet de réduire de façon non négligeable (i.e. 14%) la surface totale de l'architecture par rapport à cette architecture de référence. Lorsque que l'étape de génération du VHDL multi-modes sera terminée, nous pourrons vérifier que le partage des ressources de mémorisation entre les différents modes de fonctionnement permet encore d'améliorer ces résultats.

4. Chemin données multi-configurations

Dans le cadre de la génération d'architectures multi-modes, nous avons cherché à intégrer dans un même chemin de données différents algorithmes, et ce pour des débits applicatifs différents. Pour ce faire, nous avons généré des chemins de données combinant des filtres élémentaires : FIR, FFT et DCT.
Nous avons ensuite comparé les architectures obtenues avec des architectures similaires générées (1) par une approche naïve consistant à agréger les différentes architectures en une seule, et (2) à l'aide de l'approche SPACT-MR proposée dans [CHI05].

4.1. Présentations des algorithmes

Filtre à Réponse Impulsionnelle finie
Un filtre numérique est un élément qui effectue un filtrage à l'aide d'une succession d'opérations mathématiques sur un signal discret. C'est-à-dire qu'il modifie le contenu spectral du signal d'entrée en

atténuant ou éliminant certaines composantes non désirées. Un filtre numérique peut-être défini par une équation aux différences, c'est-à-dire l'opération mathématique du filtre dans le domaine temporel (discret). La forme générale du filtre d'ordre M est la suivante:

$$y[n] = \sum_{k=0}^{N} b_k \cdot x[n-k] - \sum_{k=1}^{M} a_k \cdot y[n-k]$$

$$y[n] = b_0 \cdot x[n] + b_1 \cdot x[n-1] + b_2 \cdot x[n-2] + ... + b_N \cdot x[n-N] - a_1 \cdot y[n-1] - a_2 \cdot y[n-2] + ... + a_M \cdot y[n-M]$$

La valeur des coefficients *a* et *b* fixera le type de filtre (passe-bas, passe-haut…).

Les filtres à réponse impulsionnelle finie (FIR) sont les exemples les plus communs de filtres numériques et se retrouvent dans de nombreuses applications de traitement du signal. Un filtre FIR est non récursif, c'est-à-dire que la sortie dépend uniquement de l'entrée du signal, il n'y a pas de contre-réaction. Ainsi, les coefficients *a* de la forme générale des filtres numériques sont tous égaux à zéro.
Une propriété importante des filtres FIR est que les coefficients du filtre *b* sont égaux à la réponse impulsionnelle *h* du filtre. D'autre part, la forme temporelle du filtre est tout simplement la convolution du signal d'entrée *x* avec les coefficients (ou réponse impulsionnelle) *b* (ou *h*).
Un exemple de filtre FIR simple est une moyenne. En effet, effectuer la moyenne sur une série de données est équivalent à appliquer un filtre FIR à coefficient constant 1/N.

Transformée de Fourrier Rapide
La Transformée de Fourier Rapide (FFT*)* est dérivée de la Transformée de Fourrier Discrète *(*DFT*)* qui est l'une des primitives les plus importantes dans le traitement du signal. Une FFT peut calculer le spectre fréquentiel d'un signal, la réponse en fréquence d'un système à partir de sa réponse à une impulsion ou peut être utilisée comme étape intermédiaire dans des traitements plus élaborés tels que l'OFDM ou l'annulation d'écho acoustique. La FFT est issue de la réorganisation des calculs de la DFT dont la définition est la suivante :

$$X(k) \Leftrightarrow x(n)$$

$$X(k) = \sum_{n=0}^{N-1} x(n) * e^{-2j\Pi \frac{kn}{N}} \quad \forall k = 0,1,...,N-1$$

$$x(n) = \frac{1}{N} \sum_{n=0}^{N-1} X(k) * e^{-2j\Pi \frac{-kn}{N}} \quad \forall n = 0,1,...,N-1$$

Où *x(n)* et *X(k)* sont, dans le cas général, des nombres complexes.

Les fondements théoriques de la FFT reposent sur la décomposition du calcul sur *N* échantillons en deux FFT réalisées sur la moitié des échantillons (échantillons pairs et impairs) :

$$X(k) = \sum_{n\ pair} x(n) * e^{-2j\Pi \frac{kn}{N}} + \sum_{n\ impair} x(n) * e^{-2j\Pi \frac{kn}{N}}$$

La décomposition est réitérée jusqu'à l'obtention d'un calcul élémentaire appelé *papillon* [COO65]. Cette décomposition est réalisée dans le domaine temporel *(DIT Decimation in Time)* ou fréquentiel *(DIF, Decimation in Frequency)* (voir Figure 5.10).

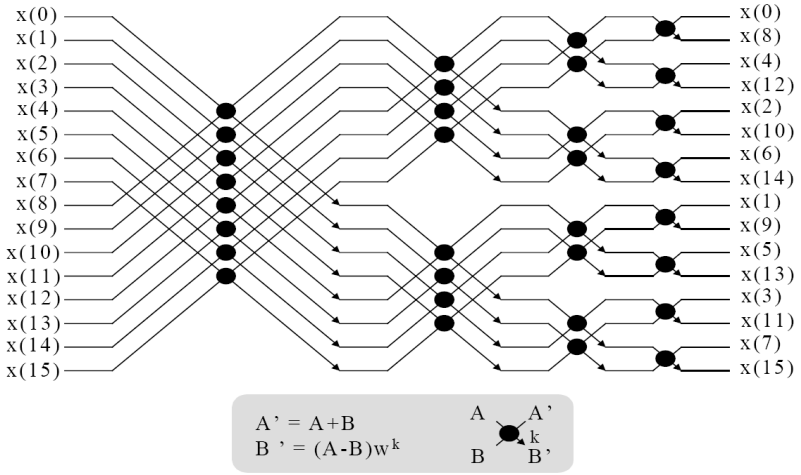


Figure 5.10. *FFT radix 2 DIF*

Transformée en Cosinus Discrète

La Transformée en Cosinus Discrète (*Discrete Cosine Transform DCT*) est un cas particulier de la Transformée de Fourier qui décompose un signal périodique en une série de fonction sinus et cosinus harmoniques. Sous certaines conditions, la DCT décompose le signal en une série de fonctions uniquement cosinus harmoniques en phase avec le signal d'origine, ce qui réduit de moitié le nombre de coefficients nécessaires par rapport à une transformée de Fourrier.

Dans le cas d'une image, le signal échantillonné est bidimensionnel et la DCT à deux dimensions (horizontale et verticale) transforme les valeurs de luminance (ou chrominance) discrètes d'un bloc de *NxN* pixels en un autre bloc *NxN* coefficients correspondant à l'amplitude de chacune des fonctions cosinus harmoniques. Ce type d'algorithme est utilisé dans les normes de codage d'images et de vidéo tel que JPEG [ISO94a] et MPEG-2 [ISO94b]. Afin de réduire la complexité et le temps de traitement des circuits intégrés réalisant ces applications, les images sont découpées en blocs de taille 8x8 pixels que la DCT transforme en blocs de coefficients également de taille 8x8.

La matrice d'entrée est définie dans le domaine spatial alors que la matrice de sortie est définie dans le domaine fréquentiel. Ainsi, le premier coefficient (en haut à gauche) représente la composante continue représentant l'intensité moyenne du bloc, et le dernier (en bas à droite), la composante de fréquence spatiale la plus élevée selon les deux axes.

La définition de l'algorithme de DCT 8x8 bidimensionnel défini dans la norme de codage vidéo MPEG-2 est la suivante :

$$G(u,v) = \frac{c(u)c(v)}{4} \sum_{m=0}^{7} \sum_{n=0}^{7} g(m,n) * \cos\left[\frac{(2m+1)u\Pi}{16}\right] \cos\left[\frac{(2n+1)v\Pi}{16}\right]$$

$$\text{avec} \quad \begin{cases} u = 0,...,7 \\ v = 0,...,7 \\ c(k) = \frac{1}{\sqrt{2}} \quad \text{si } k = 0 \\ c(k) = 1 \quad \text{sinon} \end{cases}$$

Où u et v sont les coordonnées dans le domaine de transformation.

Cet algorithme de DCT-2D 8x8 est facilement réalisable matériellement lorsqu'il est découpé en deux DCT 1D. Ceci résulte dans la formulation suivante :

$$H(m,v) = \frac{1}{2} c(v) \sum_{n=0}^{7} g(m,n) * \cos\left[\frac{(2n+1)v\Pi}{16}\right]$$

$$G(u,v) = \frac{1}{2} c(u) \sum_{m=0}^{7} H(m,v) * \cos\left[\frac{(2m+1)u\Pi}{16}\right]$$

Une représentation matricielle du calcul est [G] = [C][g][C]t est représentée graphiquement dans la Figure 5.11.

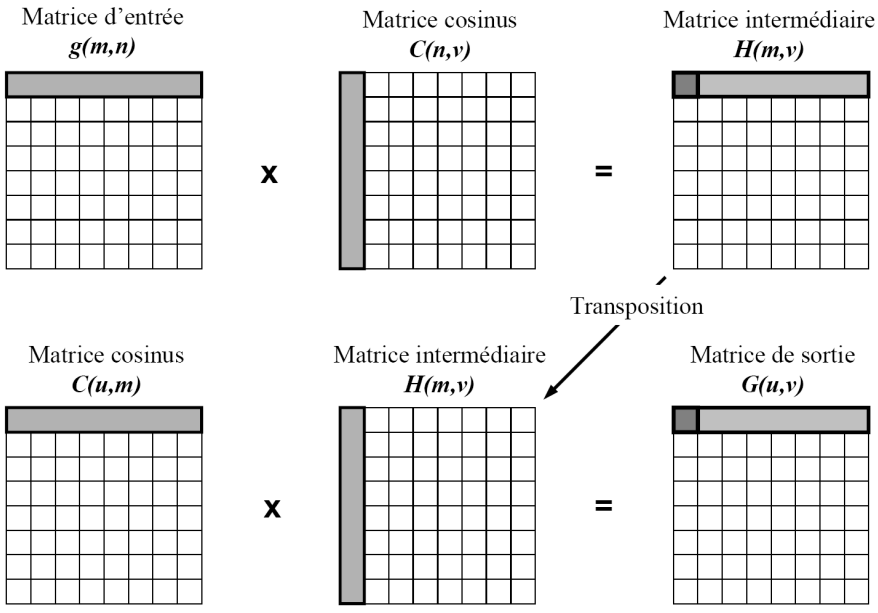


Figure 5.11. *Représentation matricielle d'une DTC-2D*

4.2. Présentations de l'approche SPACT-MR

Dans [CHI05], les auteurs présentent une méthodologie permettant de synthétiser des chemins de données à configurations multiples (plusieurs modes de fonctionnement). L'approche proposée consiste à construire les DFGs de chaque mode de fonctionnement, puis à les ordonnancer individuellement sous contraintes de latence ou de ressources (cf. Figure 5.12).

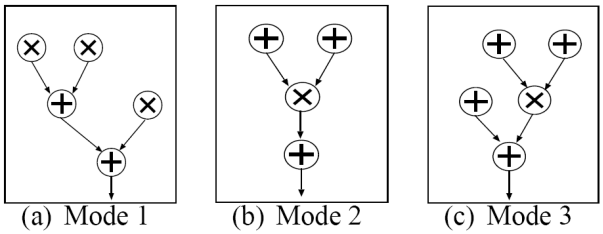


Figure 5.12. *Trois modes de fonctionnement différents*

Après ordonnancement, ces DFGs sont concaténés (cf. Figure 5.13). Cette transformation est appelée ***SPA****tially* ***C****hained* ***T****ransformation* (*SPACT*). L'étape d'assignation permet de partager les ressources de calcul entre les DFGs concaténés et de partager ainsi les ressources entre les modes de fonctionnement.

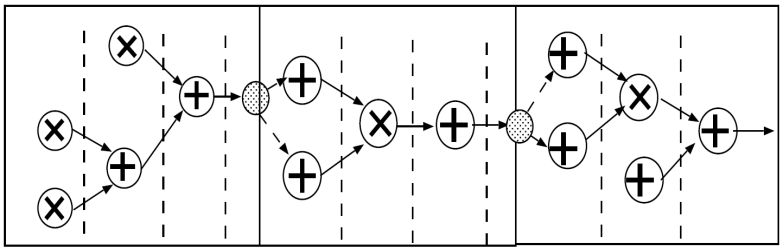


Figure 5.13. *Chaînage des DFGs des différents modes*

L'approche de référence que nous avons retenue pour ces expérimentations est l'approche SPACT-MR (pour ***M****inimum* ***R****esource*) qui vise à minimiser le nombre de ressources utilisées dans le système final, tout en garantissant une contrainte de latence. Cette approche exploite un algorithme d'ordonnancement dirigé par les forces FDS (***F****orce* ***D****irected* ***S****cheduling*, [PAU89]), et un algorithme d'assignation basé sur une approche dite *Maximal Bipartite Weighted Maching* [CHE96]. D'autres approches sont proposées dans [CHI05] basées sur différents algorithmes d'ordonnancement (contraints par les ressources ou visant à favoriser la similarité et donc la réutilisation des opérateurs entre les modes). Toutefois notre objectif est ici de comparer des architectures minimisant le nombre de ressources utilisées à travers les différents modes de fonctionnement (surface). C'est pourquoi nous avons sélectionné l'approche SPACT-MR.

Cette solution permet de générer des architectures multi-modes très optimisées en termes de ressources de calcul et de mémorisation (opérateurs et registres). Toutefois les auteurs ne tiennent pas compte de la complexité du contrôleur, ni de la logique d'aiguillage (multiplexeurs) dans leur approche. Comme nous allons le montrer, l'analyse de l'architecture dans sa globalité (opérateurs, registres, contrôleur et logique d'aiguillage) tout au long du flot de synthèse permet d'améliorer grandement ces résultats [CHA07b].

4.3. Résultats de synthèse

L'approche que nous proposons repose sur les travaux de thèse de C. Andriamisaina pour l'ordonnancement et l'assignation de DFGs multi-modes dans l'outil GAUT, et sur l'approche STAR, que nous proposons, pour la génération et l'optimisation du chemin de données en termes de contrôleur, de registres et de logique d'aiguillage (cf. chapitre IV).

Les travaux menés par C. Andriamisaina permettent de prendre en compte l'impact sur la complexité du contrôleur du partage des ressources de calcul entre les modes, lors des phases d'ordonnancement et d'assignation pour des systèmes multi-modes [CHA07b]. Ces algorithmes ont étés implémentés dans l'outil GAUT. Le flot StarSystem a été utilisé pour générer l'architecture de mémorisation, d'aiguillage et de contrôle, du chemin de données multi-modes généré par ces algorithmes d'ordonnancement et d'assignation.

Le flot de synthèse que nous avons mis en œuvre dans le cadre de ces expérimentations utilise l'outil StarDFG (cf. Figure 5.14) pour extraire, à partir des CDFG multi-modes générés par l'outil GAUT, les contraintes de communications du système. A partir de ces contraintes, l'outil StarGene est à même de générer l'architecture de mémorisation, de contrôle et d'aiguillage du chemin de données. La fusion des différents modes de fonctionnement est réalisée selon l'approche que nous avons présentée dans les chapitres III et IV.

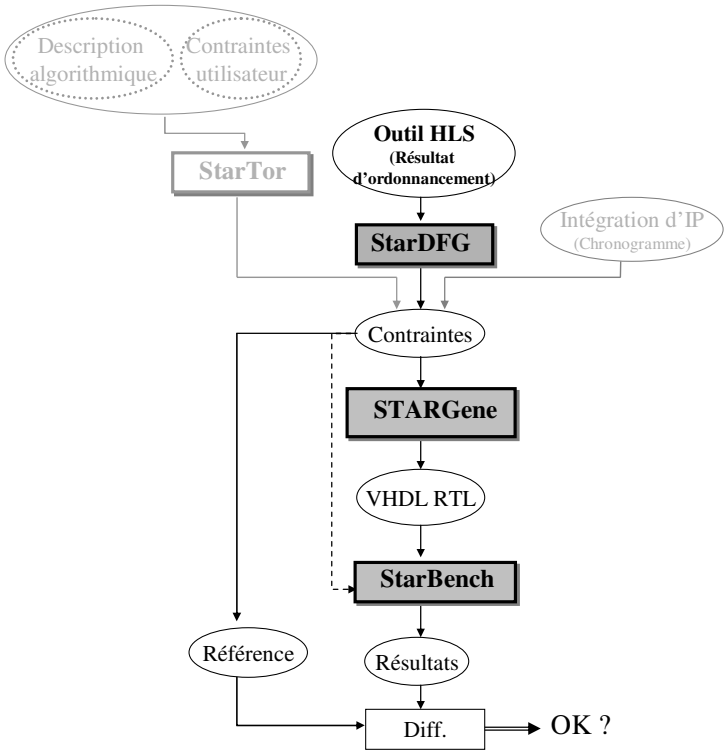


Figure 5.14. *Flot de synthèse utilisé*

4.3.1. Débits variable pour une même fonction

Pour démontrer l'efficacité de notre approche de synthèse, nous avons procédé à plusieurs expériences avec les applications connues du traitement du signal que nous venons de présenter. Ces applications sont de complexités différentes (nombre d'opérations) et doivent respecter différentes contraintes de débit.

Nous avons d'abord exploré différentes configurations d'une même application. Ainsi, en premier lieu nous considérons un ensemble de FFT 64, 32, 16 et 8 points (i.e. FFT64, FFT32, FFT16 et FFT8 dans la Table 5-7) avec des débits variant de 64 à 32 MégaEchantillons/Seconde (Me/s). Puis, nous avons expérimenté notre approche pour la génération d'un filtre FIR intégrant un FIR 64, un FIR 32 et un FIR 16 avec des débits variant de 8 à 1.4 Me/s. Enfin, le dernier filtre multi-modes (FIR19, FIR15, FIR11 et FIR7) est directement tiré des expérimentations menées dans [CHI05], avec des débits variant de 3.5 à 4.7 Me/s.

Nous avons, dans la seconde série d'expériences, cherché à générer des architectures multi-modes regroupant des applications complètement différentes avec leurs débits propres. Dans cette optique, nous avons combiné des FFT 16 et 8 points (FFT16, FFT8) et des FFT Inverse 16 et 8 points (IFFT16, IFFT8) ; un filtre LMS 16 avec un FIR 16 ; une DCT (DCT-II) 8×8 et un produit de matrice 8×8 ; et enfin une DCT 8×8 un FIR 64.

Résultats expérimentaux

Dans le cadre de ces expérimentations, nous souhaitons réduire la complexité de l'architecture. Le flot StarSystem doit ainsi générer une architecture limitant le nombre de multiplexeurs (et leur complexité) et le nombre d'éléments mémorisants. Ainsi, l'architecture obtenue n'exploitera pas de mécanisme de gel dans le cadre de ces expériences.

Pour chaque système multi-modes, nous utilisons un chemin de données 16 bits. Les surfaces des composants sont données en équivalent portes-NAND :

ADD = SUB = 212, MULT = 2032, MUX 2:1 = 100, REGISTRE = 139.

La Table 5-7 regroupe les résultats en termes d'opérateurs, de registres, de multiplexeurs (calculé en équivalent *mux 2:1* pour les multiplexeurs complexes). Nous avons expérimenté trois approches : (1) chaque application est d'abord synthétisée séparément puis ajoutée aux autres (agrégat ou architecture cumulative CA), (2) les architectures multi-modes sont synthétisées en exploitant l'approche SPACT-MR et (3) nous utilisons notre approche (chemin de données utilisant exclusivement des Registres).

Table 5-7
Surface du chemin de données (Opérateurs, Registres et Multiplexeurs)

Système Multi-Modes	CA					SPACT-MR					Notre approache					Gain (%)	
	Op		Reg	Mux[1]	Area[2]	Op		Reg	Mux[1]	Area[2]	Op		Reg	Mux[1]	Area[2]	CA	SPACT-MR
	+/ -	*				+/ -	*				+/ -	*					
FFT64, FFT32, FFT16, FFT8	14	12	306	2184	288286	6	6	163	1720	208121	6	6	163	1757	211821	26,5	-1,8
FIR64, FIR32, FIR16	3	9	40	59	30384	1	3	22	32	12566	1	3	22	33	12666	58,3	-0,8
FIR19, FIR15, FIR11, FIR7	4	12	21	39	32001	1	3	7	13	8531	1	3	7	13	8581	73,2	-0,6
FFT16, IFFT16	4	4	92	406	62314	2	2	92	326	49826	2	2	93	334	50765	18,5	-1,9
FFT8, IFFT8	4	2	39	146	24933	2	1	39	118	19677	2	1	39	118	19677	21,1	0
LMS16, FIR16	9	11	22	22	29468	8	8	16	19	22076	8	8	16	20	22176	24,7	-0,5
DCT-II 8x8, PRODMAT8x8	6	8	223	1476	196075	3	6	133	1129	144215	3	6	133	1320	163265	16,7	-13,2
DCT-II 8x8, FIR64	4	9	155	1047	145331	3	8	133	1017	137029	3	6	133	1044	135665	6,7	1

[1]*Equivalent Mux 2:1* [2]*Equivalent portes NAND*

Comparé à l'architecture cumulative, notre approche permet de réduire la surface totale de 44% en moyenne. Si l'on considère l'approche de SPACT-MR, on observe un surcoût moyen de 2.5% en surface pour le chemin de données généré avec notre approche. Ceci est provoqué par la complexité croissante des multiplexeurs générés, qui sont induits par l'étape de partage des ressources.

Table 5-8

Surface totale du chemin de données (Opérateurs, Registres, Multiplexeurs et Contrôleur)

Système Multi-Modes	CA Area^2	SPACT-MR Area^2	Notre approche Area^2	Gain (%)	
				CA	SPACT-MR
FFT64, FFT32, FFT16, FFT8	781082	524737	350821	55,1	33,1
FIR64, FIR32, FIR16	54132	26634	18786	65,3	29,5
FIR19, FIR15, FIR11, FIR7	37701	11103	9249	75,5	16,7
FFT16, IFFT16	118538	109238	81017	31,7	25,8
FFT8, IFFT8	36033	31545	25561	29,1	19
LMS16, FIR16	51396	38884	36016	29,9	7,4
DCT-II 8x8, PRODMAT8x8	774351	530143	324809	58,1	38,7
DCT-II 8x8, FIR64	370115	345813	335817	9,3	2,9

Cependant, si nous considérons la surface totale de l'architecture (cf. Table 5-8), les systèmes générés par notre approche se révèlent bien meilleurs. De fait, notre approche permet de réduire de 21% en moyenne la surface du système comparé à l'approche SPACT-MR.

Notons que notre approche permet de réduire de 75.5% la surface du système FIR19…FIR7. Dans [CHI05] les auteurs ne parviennent qu'à exhiber un gain de surface de 44.6% pour le même cas.

4.4. Conclusion

Le flot de conception présenté dans [CHA07b] exploite l'approche proposée dans nos travaux pour explorer l'espace des solutions et générer l'architecture de mémorisation et de contrôle des systèmes multi-modes que nous avons présentés.

Les résultats obtenus démontrent la pertinence du modèle et du flot de conception StarSystem pour la génération de chemin de données multi-modes. De plus, nous avons également démontré la nécessité de disposer de flot de synthèse de haut niveau prenant en compte la complexité du contrôleur et de la logique d'aiguillage dans le cadre de la génération de systèmes multi-modes. En effet, la mutualisation des ressources de mémorisation et de multiplexage, et l'optimisation du contrôleur ne sont possibles que si les étapes d'ordonnancement et d'assignation sont réalisées de façon à permettre ces optimisations.

Enfin, cette expérience prouve que notre approche est à même d'être utilisée de façon complémentaire à d'autres outils de synthèse de haut niveau. Ici, notre flot a été exploité avec l'outil GAUT grâce à l'outil StarDFG permettant d'extraire d'un CDFG ordonnancé, un ensemble de contraintes de communication.

5. Bilan

L'ensemble des expériences que nous avons présentées dans ce chapitre a permis de mettre en avant l'intérêt de notre modèle de représentation MMRCG ainsi que la pertinence de l'approche de conception associée.

Nous avons montré que l'exploration de l'espace des solutions architecturales, à l'aide de métriques paramétrables par l'utilisateur (grâce à l'interface proposée), permettait de générer des architectures permettant d'optimiser différents aspects du circuit : le nombre de points mémoire, la complexité du contrôleur, la logique d'aiguillage...

Les expérimentations menées sur un exemple industriel de composant d'entrelacement nous ont permis de démontrer la pertinence de notre solution architecturale, notamment en la comparant avec une architecture générée à l'aide d'un outil de synthèse de haut niveau du commerce (gain en surface de 14%)

Enfin, les recherches menées conjointement avec C. Andriamissaina sur une approche novatrice pour la synthèse d'architectures multi-modes ont confirmées la validité de nos choix en exhibant des performances, en terme de surface, bien supérieures aux solutions existantes. Toutefois, dans ce cas comme dans les expériences précédentes, une analyse des performances en termes de consommation (travaux en cours) permettra de confirmer, nous l'espérons, ces résultats.

Conclusion et perspectives

Nous avons proposé dans ce document un modèle formel et un flot de conception pour la spécification et la synthèse automatique de composants d'adaptation des communications pour des applications orientées traitement du signal et de l'image, et télécommunications. Nous avons montré que cette problématique d'adaptation des communications apparaissait dans trois domaines d'application que sont la synthèse de composant de brassage de données, l'intégration de composants virtuels et la génération des parties mémorisation, contrôle et aiguillage dans le cadre de la génération de chemins de données.

Notre approche est ainsi basée sur un modèle de représentation original des échanges de données noté *MMRCG* (*Multi-Modes Resource Compatibility Graph*). Cette représentation permet d'associer une sémantique à des relations temporelles entre des données : sémantique *FIFO* (*First In First Out*), sémantique *LIFO* (*Last In First Out*) ou bien sémantique *Registre*. Ce modèle permet également de formaliser différents schémas d'échanges de données, ce qui nous permet d'explorer et de générer des architectures intégrant plusieurs modes de fonctionnement. Le flot de conception associé, que nous proposons, s'inscrit dans la démarche Adéquation Algorithme Architecture et est basé sur l'utilisation de techniques de synthèse de haut niveau.

Nous avons développé une analyse formelle de ces relations temporelles en exploitant les propriétés de notre modèle. Ce dernier supporte, entre autre, la modélisation des modes de fonctionnement, permettant ainsi l'exploration et la synthèse de composant dit multi-modes.
Ce modèle est exploité par une suite logicielle, que nous avons développée, permettant (1) l'expression des contraintes de communication en fonction du contexte d'utilisation de notre approche (Adaptation des communications, synthèse de composants de brassage de données à partir d'un fichier C ou synthèse de chemin de données à partir d'un CDFG), (2) l'exploration de l'espace des solutions architecturales à partir d'une formalisation des contraintes de communication à l'aide d'un MMRCG et génération de la description en VHDL niveau transfert de registres (*RTL*) du composant, et (3) la validation de l'architecture obtenue.
En fonction du contexte d'utilisation de notre flot, les contraintes de communication qui sont utilisées pour construire le modèle formel MMRCG peuvent être extraites : par l'outil *StarTor* à partir d'un fichier C ou par l'outil *StarDFG* à partir d'un CDFG ordonnancé et assigné. Le composant d'adaptation des communications appelé *STAR* (*Space-Time AdapteR*) est généré par l'outil *StarGene* après exploration de l'espace des solutions, à l'aide d'une heuristique exploitant un ensemble de métriques (complexité du contrôleur, complexité de la logique d'aiguillage, taux d'utilisation des éléments mémorisants, taille minimale des éléments mémorisants…). Des optimisations pour l'implémentation sont également proposées à partir de transformations formelles du graphe. Enfin le composant STAR généré peut être validé par simulation à l'aide d'un banc de test généré automatiquement par l'outil *StarBench.*

Nous avons également évoqué une autre approche développée dans le cadre de ces travaux de thèse pour la génération de composants d'entrelacement. Cette approche faisant l'objet d'une demande de brevet n'a pas été détaillée dans le manuscrit.

Nous avons présenté un ensemble de résultats obtenus en appliquant notre approche à des algorithmes des domaines du Traitement du Signal et de l'Image (*TDSI*) et des Télécommunications. Nous en avons montré l'intérêt, et la validité de la mise en oeuvre et de l'utilisation, pour implémenter l'adaptation des échanges de données dans le cadre l'intégration de blocs de traitements préconçus. Ainsi, la première expérience a été réalisée sur un exemple d'adaptation des communications entre une Transformée en Ondelette (*TO*) provenant d'une architecture implémentant une application JPEG2000 et un bloc de Quantification provenant d'une architecture implémentant une application MPEG-2. Ces expérimentations ont permis de montrer l'impact des métriques d'exploration sur l'architecture générée.

La seconde expérience a utilisé un entrelaceur pour un système *UltraWide Band* (*UWB*) de type *Wireless Personnal Area Network* (*WPAN*). Afin d'évaluer les résultats, obtenus en appliquant l'approche proposée dans ce manuscrit, nous avons utilisé comme référence une architecture du même entrelaceur générée par un outil de synthèse de haut niveau du commerce au sein de l'équipe FTM HLS de STMicroelectronics. Le gain sur la surface totale du composant est de 14% en faveur de notre approche.

La dernière série d'expériences démontre l'applicabilité de notre méthodologie pour la génération de composants intégrant plusieurs modes de fonctionnement. Dans ce cadre, nous avons utilisé différentes applications du domaine du traitement du signal (*FFT, IFFT, FIR, DCT* et *produit de matrices*) devant chacune satisfaire un débit différent. Une version particulière de l'outil *GAUT* a été utilisée pour réaliser les étapes d'ordonnancement et d'assignation dans le cas de système multi-modes. Cette version de GAUT est le fruit des travaux de thèse de C. Andriamisaina (Doctorant LESTER). Les architectures obtenues sont comparées à celles générées avec l'approche *SPACT-MR*. Les résultats montrent des gains en surface totale (Opérateurs, mémorisation, aiguillage et contrôle) allant de 3% à 39% selon les systèmes étudiés.

L'ensemble de ces travaux a fait l'objet de plusieurs publications qui sont présentées dans la partie bibliographique de ce document.

Perspectives

Bien que la méthodologie proposée conduise à des résultats de bonne qualité sur les exemples traités, plusieurs améliorations et extensions peuvent être apportées comme nous l'avons mentionné à plusieurs reprises dans ce document.

Ainsi, un premier ensemble de travaux visent à permettre la modélisation et l'exploration d'architecture intégrant un mécanisme de pipeline. En effet, à ce jour nous avons proposé et exploité une solution architecturale pour la génération d'architectures pipelines. Une analyse formelle des transformations induites sur le MMRCG par une telle architecture reste à définir.

De même, des expérimentations concernant les problématiques de consommation sont actuellement en cours, notamment pour la validation des hypothèses formulées concernant l'impact du mécanisme de gel sur la consommation.
Un second ensemble de travaux concerne l'amélioration de l'exploration de l'espace des solutions. Différentes approches semblent pouvoir être exploitées dans cette optique : l'exploration des métriques à l'aide de la programmation linéaire en nombre entiers (*Integer Linear Programming, ILP*) ou encore d'autres techniques comme par exemple la génération de colonne. L'exploration architecturale à l'aide de méthodes de type ILP est également utilisée par K. Trabelsi (Doctorante LESTER) pour définir des politiques d'ordonnancement. Ces travaux utilisent l'approche STAR pour générer les chemins de données et pour évaluer la qualité des architectures obtenues (multiplexeurs et registres).

Un troisième ensemble de travaux concerne la prise en compte de la taille du chemin de données (*Bitwidth*) pendant l'exploration de l'architecture STAR. Ceci suppose d'intégrer ces aspects au sein du modèle MMRCG et de définir les transformations formelles qui en découleront. Ces travaux seront menés par C. Andriamisaina et font suite aux travaux de G. Lhairech concernant la prise en compte du bitwidth dans les algorithmes d'ordonnancement et d'assignation dans l'outil GAUT.

Un quatrième ensemble de travaux concerne la synthèse conjointe de chemins de données et d'interface d'adaptation des communications au sein d'une même architecture optimisée.

Enfin, un cinquième ensemble de travaux concerne l'application de l'approche proposée dans le brevet [CHA07a], dans le cadre de la génération de la mémorisation des données pour des applications de traitement du signal. Ainsi, certaines problématiques soulevées dans [CHA06] peuvent être résolues en appliquant un flot de synthèse de haut niveau exploitant l'approche *SAGE* (*Static Address Generation Easing*) pour la génération du chemin de données. Les performances relatives de SAGE et de STAR peuvent également être comparées dans ce cadre.

Bibliographie personnelle

CHAPITRE DE LIVRE

[COU08] COUSSY Philippe, CHAVET Cyrille, BOMEL Pierre, HELLER Dominique, SENN Eric, MARTIN Eric, "GAUT : A High-Level Synthesis Tool for DSP Applications", *dans "High-Level Synthesis: From Algorithm to Digital Circuit", en cours de publication, Springer,* premier trimestre 2008.

BREVET

[CHA07a] CHAVET Cyrille, COUSSY Philippe, URARD Pascal et MARTIN Eric, "Apparatus for data interleaving algorithm", brevet en co-propriété STMicroelectronics/CNRS, N° *0754793* (en cours d'examen à l'INPI). .

COMMUNICATIONS AVEC ACTES

Conférences Internationales

[CHA07b] CHAVET Cyrille, ANDRIAMISAINA Caaliph, COUSSY Philippe, JUIN Emmanuel, URARD Pascal, CASSEAU Emmanuel et MARTIN Eric, "A design flow dedicated to multi-mode architecture for DSP applications", *IEEE International Conference on Computer-Aided Design*, *ICCAD 2007.*

[CHA07c] CHAVET Cyrille, COUSSY Philippe, URARD Pascal et MARTIN Eric, "A Methodology for Efficient Space-Time Adapter Design Space Exploration: A Case Study of an Ultra Wide Band Interleaver"*, IEEE International Symposium on Circuits and Systems, ISCAS 2007.*

[CHA07d] CHAVET Cyrille, COUSSY Philippe, URARD Pascal et MARTIN Eric, "Application of a design space exploration tool to enhance interleaver generation", *European Signal Processing Conference*, *EUSIPCO 2007*

[CHA07e] CHAVET Cyrille, COUSSY Philippe, URARD Pascal et MARTIN Eric, "A Design Methodology for Space-Time Adapter", *ACM Great Lakes Symposium on VLSI*, *GLSVLSI 2007.*

Conférences Nationales

[CHA07f] CHAVET Cyrille, COUSSY Philippe, URARD Pascal et MARTIN Eric, "Méthodologie de modélisation et d'implémentation d'adaptateurs spatio-temporels", colloque *GRETSI 2007.*

[CHA05] CHAVET Cyrille, COUSSY Philippe, URARD Pascal et MARTIN Eric, "Approche formelle pour la Conception d'Adaptateurs Spatio-Temporel", *Manifestation des Jeunes Chercheurs en STIC, MajecSTIC 2005.*

COMMUNICATIONS SANS ACTES

Conférences Internationales

[CHA06] CHAVET Cyrille, MICHEL Thierry, GUIZZETTI Roberto et URARD Pascal, "Moving to System Level Design – Requirements & Challenges", *First High Level Synthesis workshop in Forum on specification & Design Languages, FDL 2006.*

Conférences Nationales

[CHA07g] CHAVET Cyrille, COUSSY Philippe, URARD Pascal et MARTIN Eric, "Méthodologie de synthèse d'adaptateurs spatio-temporels", colloque National du *GDR SOC-SiP 2007.*

Bibliographie

[ALO94] M. Aloqeely, C. Y. Roger Chen, “Sequencer-Based Data Path Synthesis of Regular Iterative Algorithms”, *Proceedings IEEE International Design Automation Conference (DAC), p.155-160*, 1994.

[AND97] K. Andrews, C. Heegard, D. Kozen, “A Theory of Interleavers”, *Technical report 97-1634, Computer Science Department, Cornell University*, juin 1997.

[ARM07] ARM. Advanced microcontroller bus architecture specification, http://www.arm.com/armtech/AMBA_spec.

[ART07] ARTERIS Network-on-Chip company, http://www.arteris.com.

[BAC94] D.F. Bacon, S.L. Graham, et O.J. Sharp, “Compiler transformations for high performance computing”, *ACM Comput. Surv., 26(4), p.345–420,* 1994.

[BAG98] A. Baganne, J-L. Philippe, E. Martin, "A Formal Technique for Hardware Interface design", *IEEE Trans. On Circuits And Systems, Vol.45, N°5*, 1998.

[BAL97] F. Balarin, M. Chiodob, P. Giusto, H. Hsieh, A, Jurecska, L. Lavagno, C.Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki, B.Tabbara, *"*Hardware-Software Co-Design of Embedded Systems: The Polis Approach*". Kluwer Academic Press*, 1997.

[BEN65] V. E. Benes, “Mathematical Theory of Connecting Network and Telephone Trafic”, New York, NY: Academic, 1965.

[BER93] C. Berrou, A. Glavieux, P. Thitimajshima, “Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Code*”. Proc. ICC’93, Geneva, Switzerland, p.1064-1070*, Mai 1993.

[BER99] C. Berrou, C. Douillard, M. Jézéquel. “Multiple parallel concatenation of circular recursive systematic codes”, *Annales des Télécommunications, tome 54, n°3-4, p 166-172*, 1999.

[BLU07] Bluespec ESL Synthesis Extension (ESE) to SystemC, http://www.bluespec.com/index.htm

[BOM05] P. Bomel, E. Martin, E. Boutillon, “Synchronization Processor Synthesis for Latency Insensitive Systems”, *International Conference on Design Automation and Test in Europe (DATE)*, 2005.

[BRO96] S. Brown and J. Rose. “FPGA and CPLD Architectures: A Tutorial”. *IEEE Design & Test of Computers, p. 42-57*, 1996.

[BRU99] J-Y. Brunel, E.A. De Kock, W. Kruijtzer, H. Kenter, W. Smits, “Communication refinement in Video Systems On Chip*”*, *Proceedings of IEEE International Workshop on Hardware/Software Co-Design (CODES), p.142-146*, 1999.

[BRU00] J-Y. Brunel, W. Kruijtzer, H. Kenter, F. Petrot, L. Pasquier, E.A. DeKock, W. Smits, *"*COSY Communication IP’s*"*, *Proceedings IEEE International Design Automation Conférence (DAC), p.406-410*, 2000.

[CAD03] “Cadence Virtual Component Co-Design Vcc”, http://www.cadence.com/datasheets/vcc.html.

[CAD07] "Cadence NC-VHDL simulator", http://www.cadence.com/products/functional_ver/nc-vhdl/index.aspx.

[CAF05] B.McCaffrey, "Exploring the Challenges in Creating a High-quality Mainstream Design Solution for System-in-Package (SiP) Design," *IEEE Sixth International Symposium on Quality Electronic Design (ISQED), p.556- 561*, 2005.

[CAI03] L. Cai, D. Gajski, "Transaction Level Modeling in system level design", *CECS technical report 03-10, Mar 28,* 2003.

[CAL07] "SLEC system-User manual", 2007, http://www.calypto.com.

[CAR01] L. P. Carloni, K. L. McMillan, and A. L. Sangiovanni-Vincentelli, ''Theory of Latency-Insensitive Design,'' *in. IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, 20(9) :18*, Sept. 2001

[CAR02] L. P. Carloni and A.L. Sangiovanni-Vincentelli, ''Coping with Latency in SoC Design,'' *in IEEE Micro, Special Issue on Systems on Chip, 22(5) :12,* Oct. 2002.

[CAT99] K.Danckaert, K.Masselos, F.Catthoor, H.De Man, "Strategy for Power Efficient Combined Task and Data Parallelism Exploration Illustrated on a QSDPCM Video Codec", *Journal of Systems Architecture*, Elsevier, 1999.

[CAV03] P. Cavalloro, C. Gendarme, K. Kronlof, J. Mermet, J.V. Sas, K. Tiensyrja, and N. Voros. "System Level Design Model with Reuse of System IP", *Springer, 1 edition*, September 2003.

[CES01] W.O. Cesario, G.Nicolescu, L.Gauthier, D.Lyonnard, A.A.Jerraya, "Colif: a Multilevel Design Representation for Application-Specific Multiprocessor System-on-Chip Design", *IEEE Design & Test of Computers, Vol. 18 no. 5, p. 8-20*, Sept/Oct 2001.

[CHA84] D. M. Chapiro, ''Globally-Asynchronous Locally- Synchronous Systems,'' *PhD Thesis, Stanford University*, Oct. 1984

[CHA92] V. Chaiyakul et D.D. Gajski. "Assignement decision diagram for highlevel synthesis", *Technical Report ICS-TR-92-103, University of California, Irvine,* 1992.

[CHA04] H. Charlery, A. Greiner, E. Encrenaz, L. Mortiez, A. Andriahantenaina, "Using VCI in a on-chip system around SPIN network", *Proceedings of the 11th International Conference Mixed Design of Integrated Circuits and Systems*, juin 2004

[CHE96] Y. Cheng, V. Wu, R. Collins, A. Hanson, et E. Riseman. "Maximum Weight Bipartite matching technique and its application in image feature matching", *In SPIE Conference on Visual Communication and Image Processing,* 1996.

[CHE04] D. Chen, J.Cong, "Register binding and port assignment for multiplexer optimization", *proceedings of Asia and South Pacific Design Automation Conference (ASP-DAC), p.68-73*, 2004.

[CHE06] D. Chen, J.Cong, P. Pan "FPGA Design Automation: A survey", *Foundations and trends in Electronic Design Automation, Vol.1, Issue 3, ISSN 1551-3939,* 2006.

[CHI05] L. Chiou, S. Bhunia and K. Roy, "Synthesis of Application-Specific Highly Efficient Multi-mode Cores for Embedded Systems", *ACM Transactions on Embedded Computing Systems, February* 2005

[CLA86] E.M. Clarke, E.A. Emerson et A.P. Sistla, “Automatic verification of finite-state concurrent systems using temporal logic specifications”, *ACM Transaction on programming language and systems, 8(2):244-263,* avril 1986.

[CLO03] A.Clouard, K.Jain, F.Ghenassia, L.Maillet-Contoz, J-P Strassen, “Using transactional level models in a SoC design flow, SystemC: methodologies and applications”, *Kluwer Academic Publishers, Norwell, MA*, 2003.

[COF07] CoFluentDesign. Cofluent studio. http ://www.cofluentdesign.com.

[CON06a] J. Cong, Y. Fan, G. Han, W. Jiang,, Z. Zhang, “Platform-Based Behavior-Level and System-Level Synthesis”, *Proceedings of IEEE International SOC Conference, p. 199-202*, Austin, Texas, septembre 2006.

[CON06b] J. Cong, Y. Fan, G. Han, W. Jiang, Z. Zhang, “Behavior and Communication Co-Optimization for Systems with Sequential Communication Media”, *Proceedings of the 2006 Design Automation Conference, p.675-678*, San Francisco, CA, juillet 2006.

[CON06c] J. Cong, Y. Fan, W. Jiang, “Platform-Based Resource Binding Using a Distributed Register-File Microarchitecture”, *Proceedings of the IEEE international Conference on Computer Aided Design (ICCAD), p.709-715*, novembre 2006.

[COR05] G. Corre, ‘Gestion des unités de mémorisation pour la synthèse d’architecture’, *mémoire de thèse, Université de Bretagne Sud*, 2005.

[COU02a] P. Coussy, A. Baganne, E. Martin, "A design methodology for IP integration", *Proceedings of the IEEE international Conference on Computer Aided Design (ICCAD), p.709-715*, 2002.

[COU02b] P. Coussy, A. Baganne, E. Martin, "Analyse Fonctionnelle des Moyens de communication Proposés dans les Systèmes sur Silicium", *IEEE International Symposium on Circuits and Systems, ISCAS,* 2002

[COU05] P. Coussy, E. Casseau, P. Bomel, A. Baganne, E. Martin, "A Formal Method for Hardware IP Design and Integration under I/O and Timing Constraints", *ACM Transaction on Embedded Computing Systems,* 2005.

[COO65] J.W. Cooley et J.W. Tuckey, "An algorithm for the machine calculation of complex Fourier series", *Mathematics of computation, pp. 297-301*, April, 1965

[COW07] CoWare ESL design, http://wwwcoware.com.

[CYR01] G. Cyr, G. Bois, M. Aboulhamid, J. Baillairge, "Synthesis of communication interface for SoC using VSIA recommendations", *International Conference on Design Automation and Test in Europe (DATE), 2001.*

[DAU98] I. Daubechies, W. Sweldens, "Factoring Wavelet Transforms into Lifting Steps", *in Journal of Fourier Analysis and Applications, Vol 4, N°3*, 1998.

[DEK00] E.A. De Kock, G. Essink, W.J.M. Smits, P. Van Der Wolf, J.-Y. Brunel, W.M. Kruijtzer, P. Lieverse, K.A. Vissers, ″YAPI : Application modelling for signal processing systems″, *Proceedings IEEE International Design Automation Conference (DAC), p. 402–405,* 2000.

[DEM02] G. De Micheli, R. Ernst, and W. Wolf, editors, “Readings In Hardware/Software Co-Design”*, Number ISBN:1-55860-702-1 in The Morgan Kaufmann Series in Systems on Silicon. Elsevier*, 2002.

[DIN05] L.Dinoi, S.Benedetto, "Variable-Size Interleaver Design for Parallel Turbo Decoder Architectures", *IEEE Transactions On Communications, Vol. 53, No. 11*, novembre 2005.

[DOB03] R. Dobkin, M. Peleg, and R. Ginosar, "Parallel VLSI architectures and parallel interleaving design for low-latencyMAP turbo decoders", *Tech.Rep. CCIT-TR436*, 2003.

[DON04] A.Donlin, "Transaction Level Modeling : flows and use models", *CODES+ISSS'04,* 2004.

[DUC07] L. Ducousso, "Challenges pour la conception et la vérification d'un SoC pour la HDTV", *colloque national du GDR SoC-SiP,* 2007

[DUM02] C.Dumoulin et J-L.Dekeyser, "UML framework for intensive signal processing embedded applications", *Research Report 02-07, Laboratoire d'Informatique Fondamentale de Lille, Université de Lille 1, France*, juillet 2002.

[DUR07] J. Deruere, "Développement d'une interface utilisateur pour une suite logicielle dédiée à la synthèse de haut niveau", *Rapport de stage de DUT Informatique (IUT de Vannes), France*, juillet 2007

[EFP07] Embedded Field Programmable Gate Arrays, M2000 Inc, "FLEXOStm Configurable IP Core", http://www.m2000.fr, 2007.

[EVA04] S.Evain, J.-P.Diguet, D.Houzet, "µSpider: A CAD Tool for Efficient NoC Design", dans les proceedings de IEEE Norchip Conference, p.218-221, 2004.

[EDW00] C. Edwards, "Panel weighs hardware, software design option", *EETimes, http://www.eetimes.com/story/OEG20000607S0043*, Juin 2000

[FAN04] K.Fan, M.Kudlur, H.Park, S.Mahlke, **"**Cost Sensitive Modulo Scheduling in a Loop Accelerator Synthesis System", *Proceedings of the 38th annual IEEE/ACM International Symposium on Microarchitecture, p. 219 – 232*, 2005.

[FAN06] K.Fan, M.Kudlu,r H.Park, S.Mahlke, "Increasing Hardware Efficiency with Multifunction Loop Accelerators", *proceedings of IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2006.

[FCU99] "Affirma FormalCheck User's manual", Août 1999.

[FLE99] J.Fleischmann, K.Buchenrieder, R.Kress. "Java driven codesign and prototyping of networked embedded systems with java", In *Proceedings of the 36th Design Automation Conference*, *p.794–797,* New Orleans, juin 1999.

[FLO95] B.Le Floch, M.Alard, C.Berrou, "Coded orthogonal frequency division multiplex", *Proceedings of the IEEE Volume 83, Issue 6, p. 982 - 996*, June 1995.

[FOR07] FORTE design systems, http://www.forteds.com.

[FRA04] A. Fraboulet, T. Risset, "Efficiently on-chip communications for data-flow IPs", dans *ASAP, p. 293-303*, Galveston, Texas, September 2004.

[GAJ91] D. Gajski et al. "High-Level Synthesis: Introduction to Chip and System Design"*, Kluwer Academic Publishers*, 1991.

[GAR01] R.Garello, G.Montorsi, S.Benedetto, G.Cancellieri, "Interleaver properties and their applications to the trellis complexity analysis of turbocodes**",** *IEEE Transactions on Communications Volume 49, Issue 5*, *p.793 – 807*, mai 2001.

[GAU07] GAUT, http://web.univ-ubs.fr/gaut.

[GIR91] M.B. Girkar, C.Polychronopoulos, **"**The HTG: An Intermediate Representation for Programs Based on Control and Data Dependences", *CSRD Report 1046*, May 1991

[GIU02] A**.** Giulietti, L. van der Perre, M. Strum, “Parallel turbo coding interleavers: avoiding collisions in accesses to storage elements”, *Electronics Letters 28th, Vol. 38 No. 5*, février 2002.

[GNA03a] D. Gnaedig, E. Boutillon, M. Jezequel, V. C. Gaudet, P. G. Gulak, “Les turbo-codes à roulettes”, *colloque GRETSI sur le traitement du signal et des images*, 2003.

[GNA03b] D. Gnaedig, E. Boutillon, M. Jezequel, V. C. Gaudet, P. G. Gulak, “On multiple slice turbo codes”, in *Proc. 3rd Int. Symp. Turbo Codes,Related Topics*, *p. 343–346*, Brest, France, 2003.

[GRA00] T.Grandpierre. “Modélisation d’architectures parallèle hétérogènes pour la génération automatique d’executifs distribués temps réel optimisés”, *PhD thesis, Université de Paris Sud, UFR Scientifique d’Orsay,* Novembre 2000.

[GRE06] A.Greiner, M.Benabdenbi, F.Petrot, M. Carrier, R. Chotin-Avot, R. Labayrade, “Mapping an obstacles detection, stereo vision-based, software application on a multi-processor system-on-chip”, I*ntelligent Vehicles Symposium 2006*, 2006,

[GUP04] S.Gupta, R.Gupta, N.Dutt, A.Nicolau, “Spark: A parallelizing approach to the high-level synthesis of digital circuits”*, Kluwer Academic Publishers*, 2004.

[HAN06] S-Il Han, X. Guerin, S-Ik Chae, A.A. Jerraya, “Buffer memory optimization for video codec application modeled in Simulink”, *Proceedings IEEE International Design Automation Conférence (DAC), p.689-694*, 2006.

[HAS71] A. Hashimoto et J. Stevens, “Wire routing by optimizing channel assignment within large apertures*,” in Proceedings of Design Automation Workshop*, *p.155-169*, 1971.

[HOM01a] D. Hommais, *"*Une méthode d'évaluation et de synthèse des communications dans les systèmes intégrés matériel-logiciel*", Thèse de l'Université Pierre et Marie Curie*, septembre 2001.

[HOM01b] D. Hommais, F. Petrot, I. Auge, *"*A Practical Toolbox for System Level Communication Synthesis*", Proceedings IEEE International Workshop on Hardware/Software Co-Design (CODES), p. 48-53*, 2001.

[IBM07] IBM, “32-bit processor local bus architecture specifications”, http://www-3.ibm.com/chips/products/coreconnect.

[IEE07] IEEE 802.15 Working Group for WPAN, http://www.ieee802.org/15

[ISO94a] ISO/IEC 10918-1 Digital Compression and Coding of Continuous tone Still Images (JPEG), 1994

[ISO94b] ISO/IEC JTC1/SC29/WG11 N0702(revised), Rec. H.262, 1994

[ISO00] ISO/IEC JTC1/SC 29/WG1 N1978, Décembre-2000

[JAN03] A. Jantsch and H. Tenhunen (Eds.), "Networks on Chip," *Kluwer Academic Publishers*, 2003

[JUA94] H. P. Juan, V.l Chaiyakul, et D.D. Gajski. “Condition graphs for high-quality behavioral synthesis”, *In Proceedings of the 1994 IEEE/ACM international conference on Computer-aided design, pages 170–174. IEEE Computer Society Press*, 1994.

[KAH74] G. Kahn, *"*The semantics of a simple language for parallel programming*",* In *Information Processing*, *p. 471–475*, 1974.

[KAO03] L. Kaouane, M. Akil, T. Grandpierre, and Y. Sorel. "A methodology to implement realtime applications on reconfigurable circuits", *In the Proceedings of the InternationalConference on Engineering of Reconfigurable Systems and Algorithms, p. 188–200*, 2003.

[KAT02] V. Kathail, S.l Aditya, R. Schreiber, B.R. Rau, D.C. Cronquist, and M. Sivaraman. "Pico : Automatically designing custom computers". *Computer, 35(9) :39–47*, 2002.

[KAU00] M.Kaufmann, P.Manolios, J Strother Moore (eds.), "Computer-Aided Reasoning: ACL2 Case Studies*"*, *Kluwer Academic Publishers*, *(ISBN 0-7923-7849-0)*, juin 2000.

[KEN99] H. Kenter, C. Passerone, W.J.M. Smits, Y. Watanabe, A. SangiovanniVincentelli, *"*Designing Digital Video Systems: Modeling and Scheduling*", Proceedings IEEE International Workshop on Hardware/Software Co-Design(CODES), p. 64-68*, 1999.

[KIE97] B. Kienhuis, E. Deprettere, K. Vissers And P. Van Der Wolf, "An Approach for Quantitative Analysis of Application-Specific Dataflow Architectures*"*, *Proceedings International Conference. On Application-specific Systems, Architectures and Processors, p.338-349*, 1997.

[KIE00] B.Kienhuis, E.Rypkema, Ed Deprettere, "Compaan: Deriving process networks from matlab for embedded signal processing architectures," in *Proceedings of the 8th International Workshop on Hardware/Software Codesign (CODES)*, San Diego, USA, mai 2000.

[KIR83] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, "Optimization by Simulated Annealing, Science", *Vol 220, Number 4598, p. 671-680*, 1983.

[KOL93] T.Kolks, B.Lin, H. de Man, "Sizing and verification of communication buffers for communicating processes". *In Proceedings of the International Conference on Computer Aided Design (ICCAD), pages 660–664*, Santa Clara, CA, USA, novembre 1993.

[KUH99] T.Kuhn, W.Rosenstiel, U.Kebschull. "Description and simulation of hardware/software systems with java", In *Proceedings of the 36th Design Automation Conference*, *p.790–793*, New Orleans, juin 1999.

[KUN82] H.T.Kung, "Why systolic architectures", *computer, vol.15, p.37-46, Jan,* 1982.

[LAG00] X.Lagrange, P.Godlewski, S.Tabbane, "Réseaux GSM : des principes à la norme", *Éditions Hermès Sciences*, 2000, *ISBN 2-7462-0153-4.*

[LAV99] L.Lavagno, E.Sentovich, "ECL : A specification environment for system level design", In *Proceedings of the 36th Design Automation Conference, p.511–516*, 1999.

[LEE87] E.A Lee, D.G Messerschmitt, "Synchronous Dataflow", *Proceedings of IEEE*,1987

[LEE95] E.Lee, T. Park, "Dataflow Process Networks", *In Proceedings of the IEEE, volume 83, p.773-799*, 1995.

[MAL06] Kevin Fan, Manjunath Kudlur, Hyunchul Park, Scott Mahlke, "Increasing Hardware Efficiency with Multifunction Loop Accelerators"**,** *Proc. 2006 Intl. Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), p. 276-281*, 2006.

[MAR93] E. Martin, O. Santieys, Jl. Philippe, "GAUT, An Architecture Synthesis Tool for Dedicated Signal Processors*", Proceedings IEEE International European Design Automation Conference (Euro DAC), pp.14-19*, 1993.

[MEN07] Catapult synthesis for ASIC and FPGA, http://www.mentor.com

[MOD07] ModelSim simulation and debug environment, http://www.model.com

[MOU05] Y.Le Moullec, J-Ph.Diguet, T.Gourdeaux, J-L.Philippe, “Design Trotter: System-Level Dynamic Estimation Task a 1st step towards platform architecture selection”, *Journal of embedded computing (JEC), Cambridge Int. Science Pub, issue 4*, 2005.

[MOU07] H.Moussa, O.Muller, A.Baghdadi, M.Jézéquel, “Butterfly and benes-based on-chip communication networks for multiprocessor turbo decoding”, *Proceedings of the conference on Design, automation and test in Europe*, 2007.

[OBE99] J.Oberg, “ProGram: A Grammar-Based Method for Specification and Hardware Synthesis of Communication Protocols”, *Doctoral Thesis, Royal Institute of Technology, Department of Electronoics, Electronic System Design, Stockholm Sweden. TRITA-ESD-99-03, ISSN 1104-8697, ISRN KTH/ESD/AVH--99/3-SE.*

[OCP07] Open-Core Protocol Int. Partnership Association Inc. Open-core protocol specification, http://www.ocpip.org

[ORA96] A. Orailoglu et D.D. Gajski, “Flow graph representation”, *In the Proceedings of the 23rd ACM/IEEE conference on Design automation (DAC’86), p. 503–509, Piscataway, NJ, USA*, 1986.

[PAU89] P.Paulin, J. Knight "Force-Directed Scheduling for the Behavioral Synthesis of ASIC’s”, *in IEEE Trans. On Computer-Aided Design, Vol.8, No.6, June*, 1989

[PAU07] P.Paulin, "Défis et Solutions pour la Programmation de Plateformes Multi-Processeur Hétérogènes”, *Ecole d'hiver Francophone sur les Technologies de Conception des systèmes embarqués Hétérogènes (FETCH07),* 2007.

[PET77] J.L.Peterson, "Petri Nets", *ACM Computing Surveys 9, p.223–252,* 1977.

[PET03] F. Petrot, P. Gomez, *"*Lightweight Implementation of the POSIX Threads API for an On-Chip MIPS Multiprocessor with VCI Interconnect*"*, *Proceedings IEEE International Conference on Design Automation and Test in Europe (DATE), p. 51-56*, 2003.

[PSL07] IEEE P1850 - Standard for PSL - Property Specification Language.

[QUE82] J.P. Queille et J. Sifakis, “Specification and verification of concurent systems in CESAR”, *Procedings Int. Symp. On programming, Turin Italy, avril 1982 Vol. 137 de Lectures notes in computer science, pages 337-351,* 1982.

[RBF99] IBM Haifa Research Laboratorie, “RuleBase Formal Verification Tool”, *Version 1.4.3 Verification Technologies Group,* 2003.

[SAN00] A.Sangiovanni-Vincentelli, M.Sgroi, L.Lavagno. "Formal Models for Communication-based Design". *IEEE Design & Test, Volume 17, Issue 2, p. 14 – 27,* Avril 2000.

[SAN04] I.Sander, A.Jantsch, “System modeling and transformational design refinement in ForSyDe”, *dans IEEE transaction on CAD ICS, vol. 23, n°1*, janvier 2004

[SCH86] A.Schrijver, "Theory of Linear and Integer Programming", *John Wiley and Sons*. 1986.

[SEE00] R. Seepold et N.M. Madrid, “Virtual Components Design and Reuse”. *Springer, 1 edition*, 2000.

[SLI00] K. Slind. Another look at nested recursion. In Mark Aagaard and John Harrison, editors, “Theorem Proving in Higher Order Logics"*, 13th International Conference, TPHOLs'00, Portland Oregon, August 2000, volume 1869, p. 498-518. Springer-Verlag*, 2000.

[SOC07] http://soclib.lip6.fr/Home.html.

[SOW04] A. Sowmya V. D'silva, S. Ramesh, "Synchronous Protocol Automata: A Framework for Modelling and Verification of SoC CommunicationArchitectures". In Design, Automation and Test in Europe Conference and Exhibition, volume I, p. 10390, 2004.

[SPI07] SPIRIT consortium, http://www.spiritconsortium.org

[STO94] L.Stok, "Datapath synthesis", *the VLSI journal (18) ,p.1-71* , 1994.

[SYN07] "Formality, functional equivalence checking", 2007, http://www.synopsys.com

[TAG00] H.Tago, K.Hashimoto, N.Ikumi, M.Nagamatsu, "CPU for Playstation2", *proceedings of Workshop on Synthesis And System Integration of Mixed Information technologies (SASIMI), p.153–157*, 2000.

[TAM03] L. Tambour, "Méthodologie et flot semi-automatique d'aide à la conception et à la validation des macro-cellules ASIC dédiées au traitement numérique du signal", Thèse de l'Institut National Polytechnique de Grenoble, 2003.

[TAR04] A. Tarable, S. Benedetto, G. Montorsi, "Mapping interleaving laws to parallel turbo and LDPC decoder architectures", *IEEE Transactions on Information Theory, vol.50, p.2002 - 2009*, septembre 2004.

[THA06] Thabet F., Coussy P., Heller D., Martin E. "Design Space Exploration of DSP Applications Based on Behavioral Description Models", *dans IEEE Workshop on Signal Processing Systems Design and Implementation SIPS,* 2006.

[THU02] M. J. Thul, F. Gilbert, N.Wehn, "Optimized concurrent interleaving architecture for high-throughput turbo-decoding", in *Proc. 9th Int. Conf. Electron., Circuits, Syst.*, *vol. 3, p. 1099–1102,* 2002.

[TUR02] A. Turjan, B. Kienhuis and E. Deprettere, "A compile time based approach for solving out-of-order communication in Kahn Process Networks", *In Proceeding of IEEE 13th International Conference on Aplication-specific Systems, Architectures and Processors (ASAP)*, 2002.

[URA07] P.Urard, "Discussion: les systèmes hétérogènes discret/continu", *Ecole d'hiver Francophone sur les Technologies de Conception des systèmes embarqués Hétérogènes (FETCH),* France, 2007

[VER02] F. Vermeulen, *"*Reuse of System-Level Design Components in Data-Dominated Digital System*",* Thèse de l'université Catholique de Louvain, décembre 2002.

[VIA06] E.Viaud, F.Pêcheux, A.Greiner, "An Efficient TLM/T Modeling and Simulation Environment Based on Conservative Parallel Discrete Event Principles", *International Conference on Design Automation and Test in Europe (DATE)*, 2006.

[VSI] Virtual Socket Interface Alliance, http://www.vsi.org

[WEH04] N. Wehn, "SoC-Network for Interleaving in Wireless Communications", *In Proceeding of MPSoC,* Saint-Maximin la Sainte Baume, France, 2004.

[WEN93] A. S. Wenban, J. W. O'Leary, "Codesign of communication protocol", *IEEE Computer*, *26(12), p.46–52*, décembre 1993.

[WOD03] P.Wodey, G. Camarroque, F. Baray, R. Hersemeule, J.-P. Cousin, "LOTOS Code Generation for Model Checking of STBus Based SoC : the STBus interconnect",

Proceedings of the First ACM and IEEE International Conference on Formal Methods and Models for Co-Design (MEMOCODE'03), 2003

[XIL07] http://www.xilinx.com, "High-Performance 1024-point Complex FFT/IFFT", 2007.

[ZER04] N. Zergainoh, K. Popovici, A.A. Jerraya, P. Urard, "Matlab based Environment for designing DSP Systems using IP blocks", *12th Workshop on Synthesis And System Integration of Mixed Information technologies, SASIMI, p. 296 – 302*, 2004.

[ZER06] N.E. Zergainoh., L. Tambour, P. Urard, A.A. Jerraya, "Macrocell builder: IP block-based design Environment for high-throughput VLSI dedicated digital signal processing systems", *EURASIP Journal on Applied Signal Processing Volume 2006, Article ID 28636, Pages 1–11*, 2006.

[ZHE03] Y. Zheng, G. Bartsch, "Transaction level modeling", *Seminar : Algorithms and tools for computer aided circuit design, 2003-11-13*

[ZIS02] C. Zissulescu-Ianculescu, A. Turjan, B. Kienhuis and E. Deprettere, "Solving Out of Order communication using CAM memory: an implementation", *ProRisc02*, Veldhoven, 2002.

ANNEXE A
OUTILS DE SYNTHESE DE HAUT NIVEAU

1. L'outil de synthèse DTSE

Ces travaux menés à l'IMEC s'inscrivent dans la démarche méthodologique *DTSE (Data Transfer and Storage Exploration)* de conception de systèmes embarqués pour les applications orientées télécommunications et multimédia dans lesquelles les transferts de données sont prédominants. Ainsi, dans [VER02] l'auteur développe une approche d'intégration des composants évitant l'utilisation d'un module d'interface pour adapter les séquences d'E/S. Il utilise pour cela une modélisation hiérarchique, en trois niveaux, des composants virtuels. Dans cette optique, le niveau le plus bas représente les opérations de type scalaire *(motif de calcul)* spécifiées à l'aide de sous-programme, le niveau intermédiaire décrit les boucles et les structures d'indexation dans lesquelles sont appelés les sous-programmes et le plus haut niveau hiérarchique spécifient le contrôle du processus (cf. Figure 0.1).

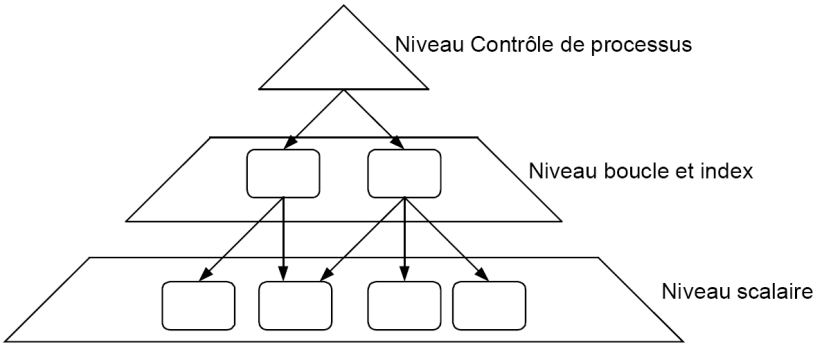


Figure 0.1 . *Décomposition en niveaux hiérarchiques*

L'adaptation des séquences d'acquisition et de production est réalisée dans le niveau intermédiaire de la description hiérarchique des composants virtuels. Pour cela les auteurs modifient la hiérarchie des boucles (cf. Figure 0.2.a) et/ou réordonnent les accès aux structures multidimensionnelles en réalisant des calculs sur les indices de boucles. (cf. Figure 0.2.b).

Avant	**for i = 0 to 127**
	  for j = 0 to 31
	  ligne[i,j] = operation_ligne(ligne[i,j])
		for i = 0 to 127
		  for j = 0 to 31
		  (a[i,j*4] = operation_ligne(z[i,j*4])

Après	for j = 0 to 31
	  for i = 0 to 127
	  ligne[i,j] = operation_ligne(ligne[i,j])
		for i = 0 to 31
		  for j = 0 to 31
		  for k = 0 to 3
		  a[i*4+k,j*4] = operation_ligne(z[i*4+k,j*4])

	(a)	**(b)**

Figure 0.2 . *Exemple de modifications appliquées*

Cette méthodologie, bien qu'efficace, oblige le concepteur à réécrire en partie le code de la spécification initiale. De plus, la modification des modes d'accès aux tableaux requiert des calculs sur les indices de boucles ayant une forte complexité et une réorganisation de la hiérarchie de boucle non triviale. Cette méthode ne permet pas l'intégration des composants virtuels dans le cadre général de l'adaptation de débit ou de protocole et ne fait aucune référence aux contraintes de communication tels que les modes de synchronisation. Enfin, les auteurs n'abordent pas la problématique de la synthèse d'architectures devant intégrer plusieurs modes de fonctionnement.

2. L'Outil de Synthèse SPARK

SPARK est un outil de synthèse comportementale développé à l'Université de Californie à San Diego dans le groupe "*Microelectronic Embedded Systems*" dirigé par R. Gupta. L'outil de synthèse de haut niveau SPARK possède comme point d'entrée une description comportementale écrite en C-ANSI. Le C-ANSI accepté possède quelques restrictions sémantiques : il n'est pas possible de synthétiser une description contenant des pointeurs, il ne supporte pas non plus les fonctions récursives, les sauts conditionnels et inconditionnels.

SPARK repose sur un flot de synthèse qui privilégie les algorithmes de calcul intensif. Une présentation de l'outil est disponible dans [GUP04]. Le flot de synthèse employé par SPARK est présenté dans la Figure 0.3. La description comportementale initiale est transformée par la phase de compilation en une représentation interne de type CDFG (cf. Chapitre III).

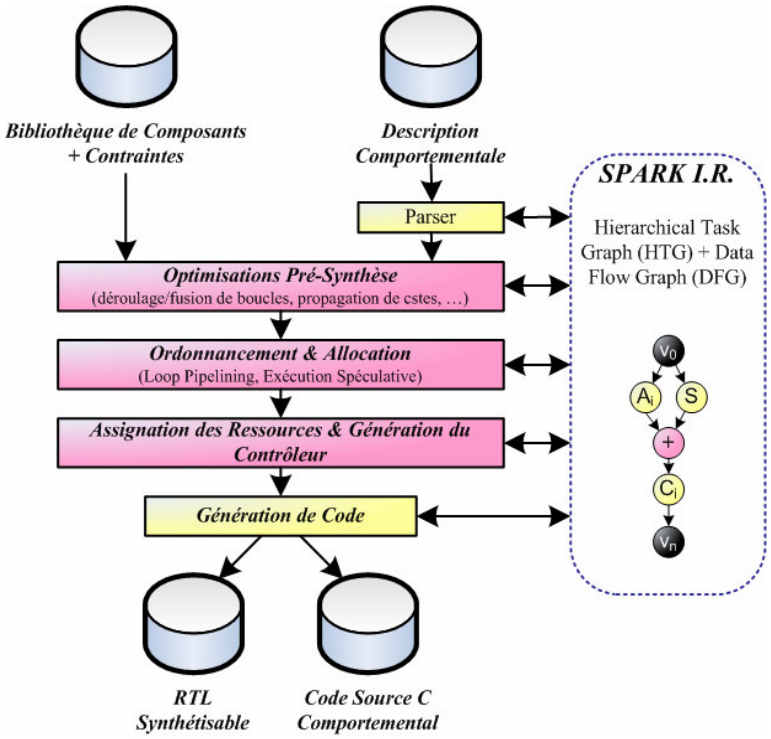


Figure 0.3. *Le flot de synthèse SPARK*

Afin de réaliser la synthèse, le concepteur doit fournir à l'outil une bibliothèque contenant les opérateurs qui pourront être utilisés durant les étapes de la synthèse ainsi que les contraintes devant guider la synthèse. Les contraintes acceptées par l'outil SPARK sont des contraintes matérielles (nombre et type des opérateurs disponibles). En fonction de ces paramètres, l'outil de synthèse va générer une architecture en optimisant sa latence. Les tâches de sélection et d'allocation des opérateurs sont à la charge du concepteur.

La première étape du flot de synthèse consiste à appliquer une suite d'optimisations sur la description fournie en entrée. Les optimisations appliquées sur le code source de la description sont les suivantes :

- Mise en ligne des procédures (*Function Inlining*),
- Déroulage partiel ou total des boucles suivant les directives spécifiées par le concepteur (*Loop Unrolling*),
- Fusion de boucles (*Loop Fusion*),
- Elimination des calculs communs (*Common Sub-Expression Elimination, CSE*),
- Elimination du code mort (*Dead Code Elimination*),
- Extraction des invariants de boucle (*Loop-Invariant Code Motion*),
- Réduction de la complexité des opérations (*Operation Strength Reduction*).

La gestion du déroulage partiel ou total de chacune des boucles présentes dans la description comportementale fournie à l'entrée de l'outil se fait sous contrôle exclusif de l'utilisateur. Cela permet au concepteur de pouvoir expérimenter différents choix et ainsi choisir de manière manuelle le compromis qui lui semble optimal.

Méthode d'ordonnancement sous ressources contraintes

L'ordonnancement au sein de l'outil se fait en plusieurs étapes. Avant l'ordonnancement, des algorithmes de transformation vont modifier les structures conditionnelles (Basic Bloc) présentes dans le graphe afin d'accroître le parallélisme présent en déplaçant des opérations entre les différentes structures. Les algorithmes de "Code Motions" qui réalisent ces mouvements entre les différents blocs ont plusieurs objectifs :

(1) Augmentation du partage des ressources matérielles : plus on regroupera d'opérations au sein d'une même structure conditionnelle, plus on pourra exploiter le parallélisme existant entre les opérations et ainsi lisser le taux d'utilisation des ressources matérielles,

(2) Diminution de la durée du chemin critique (latence de l'architecture). La réduction de la latence de l'architecture est une conséquence directe de l'augmentation du partage des ressources matérielles, réduisant ainsi la durée totale d'exécution des "basic blocs".

Une fois les transformations appliquées sur le graphe, chaque structure conditionnelle est ordonnancée de manière indépendante. Le CDFG une fois ordonnancé va servir à la génération de la machine à états finis (FSM) qui va piloter l'architecture décrite au niveau transfert de registre (RTL).

Déroulement de l'ordonnancement - Au niveau de l'ordonnancement, différentes optimisations sont apportées à l'aide d'heuristiques : "*Speculative*, *Percolation* et *Trailblazing Code Motion*" [GUP04], "*Dynamic Renaming of Variables*" afin là aussi d'augmenter le parallélisme entre les opérations.

L'algorithme d'ordonnancement utilisé est de type "list-scheduling" [GUP04]. Suite à cet ordonnancement, l'assignation des opérations sur les opérateurs matériels est réalisée. Durant cette opération, la logique nécessaire à la communication entre les composants et les registres est générée. La dernière étape consiste à réaliser la génération de l'automate de contrôle qui va piloter l'architecture. Il est généré à partir du mapping des opérations qui vient d'être réalisé.

L'automate de contrôle qui est généré en sortie de l'outil SPARK correspond à une FSM de Moore dans laquelle on trouve des conditions sur les transitions permettant la gestion des boucles et des branches conditionnelles. En sortie, l'outil génère une architecture décrite au niveau VHDL-RTL synthétisable.

L'outil propose également en sortie du VHDL de niveau comportemental (correspondant à la description d'entrée), ainsi que du C niveau RTL permettant de visualiser plus facilement les optimisations réalisées par l'outil et de réaliser une simulation afin de vérifier l'intégrité de la solution.

3. L'outil de synthèse Streamroller

Streamroller est un outil de synthèse comportementale développé à l'Université du Michigan dans le groupe "*Advanced Computer Architecture Laboratory*" dirigé par S.Mahlke. Dans [MAL06] les auteurs présentent une méthodologie permettant de concevoir des architectures pipelinées pour des accélérateurs multifonctions.

A partir d'un code C et de contraintes fournies par l'utilisateur (débit du pipeline, période d'horloge, …), le flot modélise les communications sous la forme d'un graphe à partir de l'analyse des dépendances de données. Ce graphe, appelé *Loop Graph* (cf Figure 0.4), est similaire aux réseaux de processus (e.g. KPN) mais sans l'aspect parallélisation. Il permet de modéliser la structure de communication entre les processus. L'approche consiste ensuite à synthétiser des accélérateurs de boucles et des tampons à base de mémoires ping-pong pour assurer les transferts de communication.

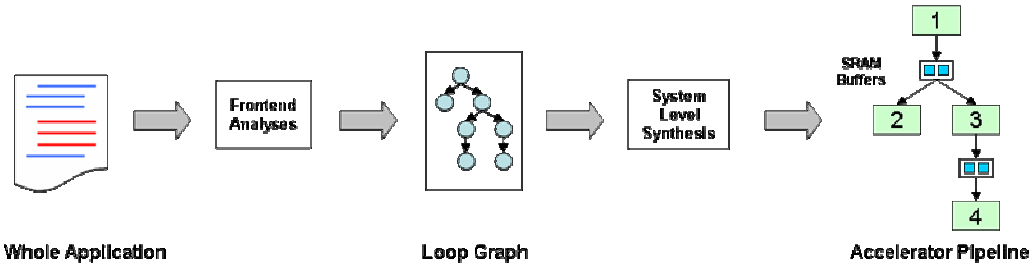


Figure 0.4. *Flot de conception Streamroller*

L'architecture interne des accélérateurs de boucles (cf. Figure 0.5) est elle générée en utilisant le principe du *modulo scheduling* [FAN04]. L'approche permet de générer des architectures multifonctions.

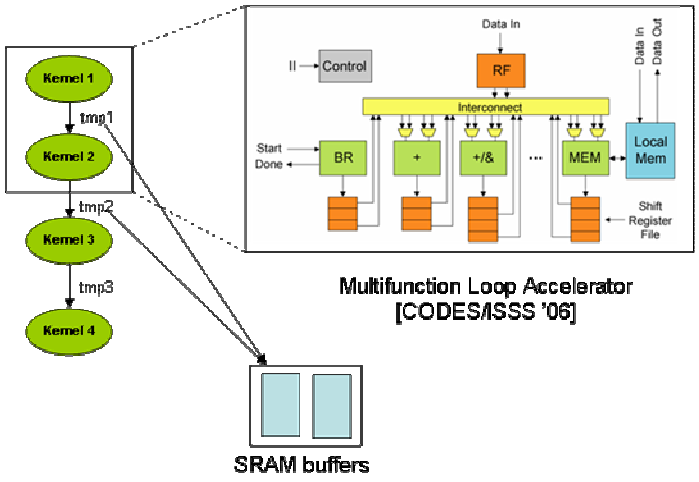


Figure 0.5. *Détails de l'architecture interne des accélérateurs de boucle*

A partir de descriptions en langage C des fonctions, l'outil génère une description de niveau transferts de registre. La première étape consiste à construire une architecture VLIW abstraite sur laquelle l'application sera mappée. Cette architecture a pour paramètres le nombre d'unités fonctionnelles (FU) et leur fonctionnalité propre. L'analyse des opérations présentes dans les boucles et des débits désirés permet ensuite à l'algorithme d'allocation, à l'aide d'une librairie de cellules, d'allouer les unités fonctionnelles. Le but est de minimiser le coût de l'architecture tout en garantissant que les contraintes de performances seront respectées.

L'étape suivante consiste à appliquer un algorithme d'ordonnancement modulo pour implémenter l'application dans l'architecture définie à l'étape précédente. L'ordonnancement assigne les opérations des boucles sur des unités fonctionnelles, en respectant les dépendances de données et les contraintes de modulo. L'architecture du chemin de données et le contrôleur associé sont alors générés.
L'approche proposée permet également de générer des architectures multifonctions (cf. Figure 0.6). Ici, cela consistera à fusionner dans une seule et même architecture plusieurs boucles, c'est-à-dire plusieurs graphes. Pour ce faire, les auteurs proposent un algorithme d'ordonnancement conjoint basé sur une approche de programmation linéaire en nombres entiers (ILP, pour Integer Linear Programming) [SCH86].

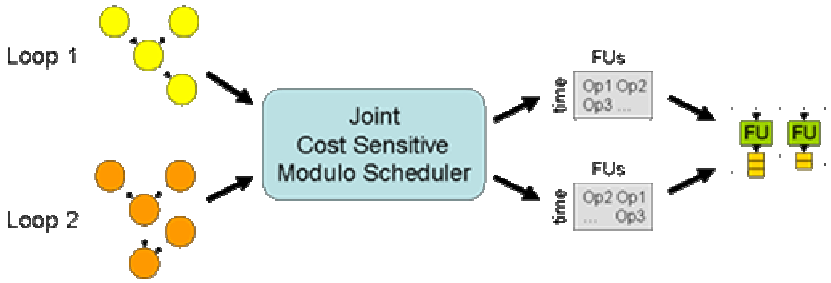


Figure 0.6. *Flot de synthèse pour des architectures multi-modes*

Toutefois, on notera que l'approche Streamroller est limitée sur deux points :

- les mémoires tampons générées entre les accélérateurs de boucles utilisent le principe des mémoires ping-pong. Ce qui signifie que l'approche ne cherche pas à optimiser ces mémoires en faisant par exemple du recouvrement de communications comme [BAG98].
- l'architecture mémoire générée lors de la synthèse du chemin de données des accélérateurs, n'exploite que des registres ou des fichiers de registres à sémantique FIFO. L'exploration est donc limitée par ces contraintes architecturales.

4. L'outil de synthèse xPilot

xPilot est un outil de synthèse développé à l'Université de Californie dans le "*Computer Science Department*" dirigé par J.Cong. Dans [CON06c] les auteurs présentent une approche de synthèse des communications qui utilise des *SCM* (acronyme de Sequential Communication Media). Il s'agit de l'une des étapes du flot de synthèse (cf. Figure 0.7) associé à l'outil xPilot [CON06b].

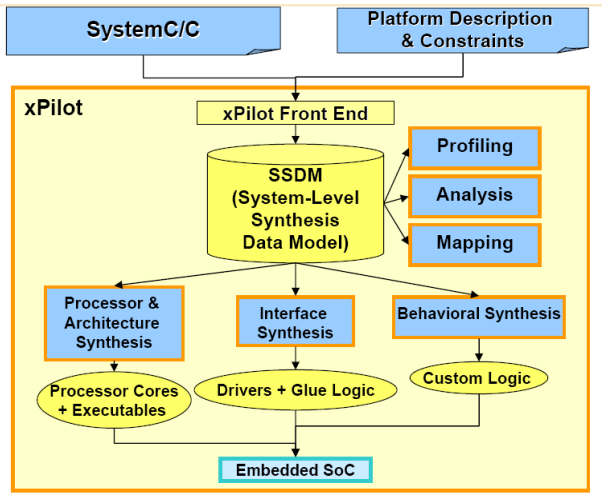


Figure 0.7. *Le flot de synthèse xPilot*

L'objectif de ces travaux d'analyser le comportement aux entrées sorties des éléments de calcul, de déterminer les ordres de transfert permettant d'optimiser la latence, de transformer automatiquement les descriptions comportementale des algorithmes et enfin de générer les pilotes et la logique de contrôle nécessaire au bon fonctionnement du système.

Pour ce faire, le flot prend en entrée du code SystemC et génère une architecture de niveau transfert de registres. La synthèse des interfaces de communication est réalisée par l'outil SCOOP (cf Figure 0.8).

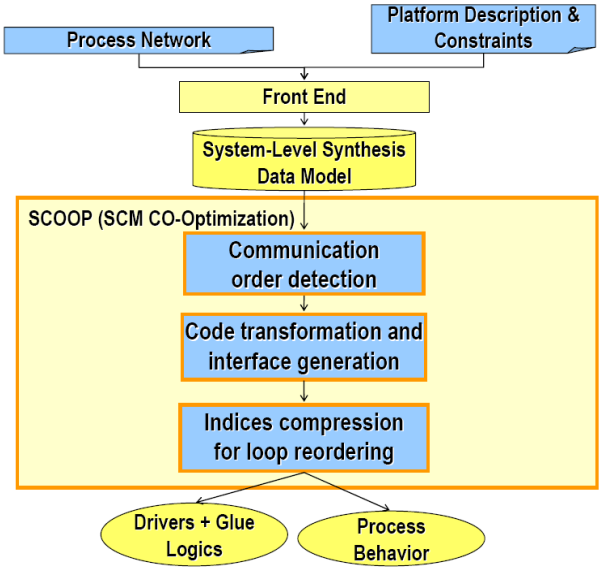


Figure 0.8. *Flot de conception de l'outil SCOOP*

Cet outil modélise le comportement de chaque processus sous la forme d'un graphe de flot de contrôle et de données (ou CDFG, pour Control & Data Flow Graph, [GUP04]). Puis, ces CDFG sont ensuite analysés afin de déterminer un ordre de transfert de données optimal qui minimise la latence globale du système et respecte une sémantique de communication de type FIFO. L'algorithme qui analyse les communications se base sur le fait que ce problème est équivalent à un problème d'ordonnancement

sous contrainte de ressource. Le principe est de considérer les données qui seront transmises sur le SCM, comme des opérations. L'ordonnancement sous contrainte de ressources étant un problème NP-complet, les auteurs utilisent un algorithme basé sur le principe des algorithmes d'ordonnancement par listes ([GAJ91]).

Dans la seconde étape (cf. Figure 0.9), le code source des différents processus est modifié pour prendre en compte les optimisations détectées lors l'étape précédente, sans pour autant violer les dépendances de données.

Enfin, l'interface proprement dite est générée (pilotes et logique de contrôle).

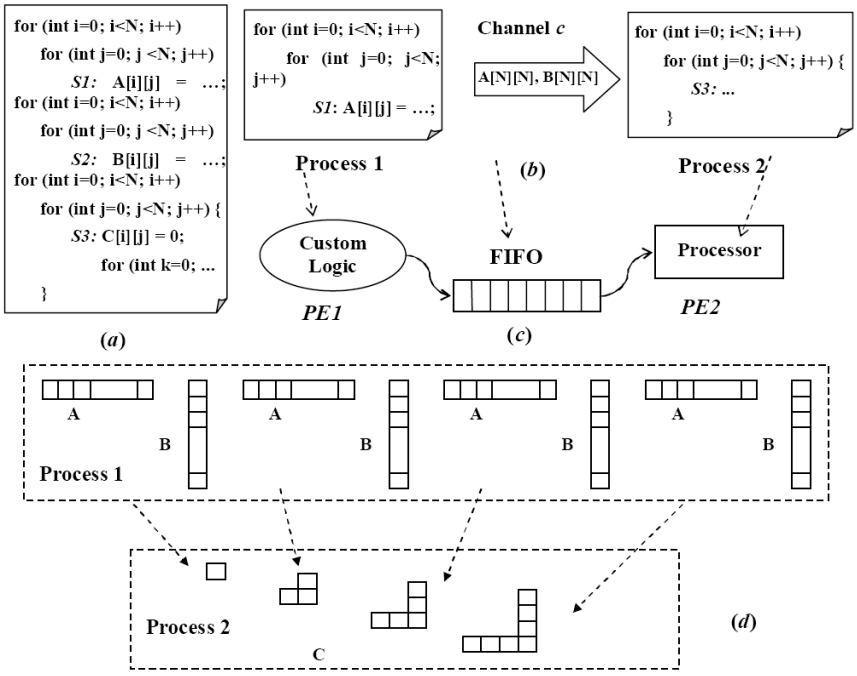


Figure 0.9. *Exemple d'application du pour la génération d'un SCM*

Cette approche propose donc un flot de synthèse sous contrainte de communication, au même titre que les travaux présentés dans [COU05]. Toutefois, les auteurs supposent une contrainte forte sur les échanges de communication : les données sont échangées (écrites et lues) dans le même ordre. Ainsi, ils analysent le code SystemC pour calculer l'ordre d'échange des données optimal et transforment le code originel en conséquence. Ils cherchent à orienter leur architecture vers un système de communication à base de FIFO, limitant ainsi le champ de l'exploration architecturale, comme nous l'avons vu précédemment dans les approches proposées par [VER02] ou [KIE00]. Enfin, les auteurs ne présentent pas de solution pour générer des architectures multi-configurations à l'aide de leur méthodologie.

5. L'outil de synthèse PICO

PICO est un outil développé et distribué par Synfora. Le système PICO complet est un outil de Synthèse/Co-Design. Il propose une méthodologie capable, à partir d'une description algorithmique en langage C, de produire une architecture implémentant la-dite application. L'architecture est basée sur un ensemble d'accélérateurs matériels élémentaires nommés NPA (Non- Programmable Accelerator) qui permet d'accélérer le traitement des nids de boucles dans les applications TDSI [KAT02]. L'architecture cible est présentée en Figure 0.10. Elle se compose d'un processeur VLIW qui va réaliser les opérations extérieures aux nids de boucles et qui va piloter un réseau systolique de processeurs.

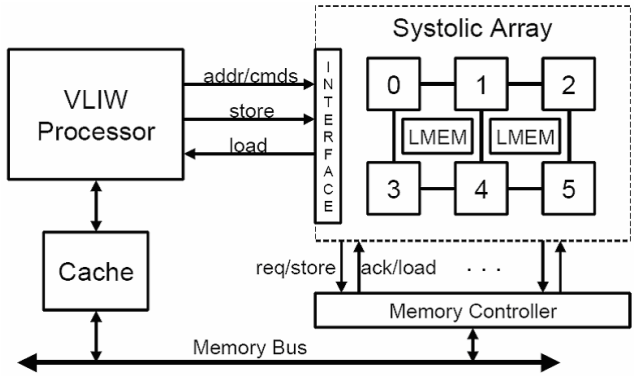


Figure 0.10 . *Architecture cible de l'outil Pico NPA*

La première étape consiste à isoler la partie calcul intensif du code et de la dériver en un NPA. La synthèse du NPA est réalisée sous contrainte de latence, l'objectif étant de respecter la contrainte temporelle tout en minimisant le coût de l'architecture. Le concepteur peut aussi contraindre la synthèse en jouant sur la bande passante maximum allouée à la mémoire (limitation du nombre de transferts simultanés).

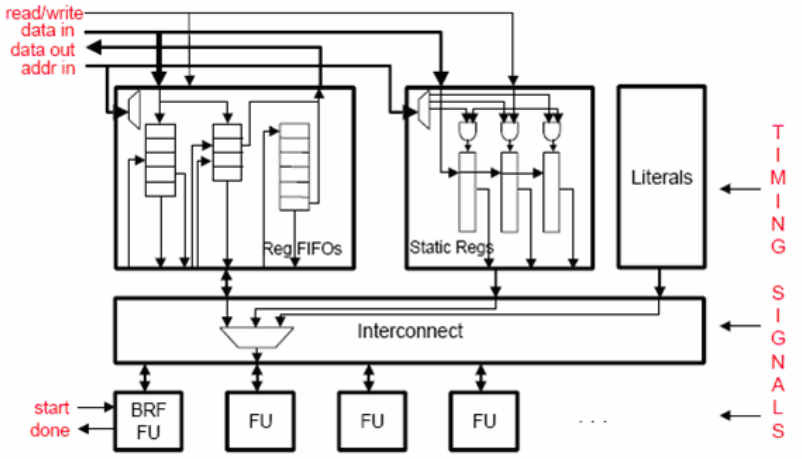


Figure 0.11 . *Architecture d'un processeur (Pico)*

Lors de la synthèse des processeurs du réseau, l'outil va chercher les unités fonctionnelles dans des bibliothèques où les composants sont déjà caractérisés (surface, délai). L'outil exploite les différentes largeurs de bits entre les opérations et les données lors de la sélection du matériel à mettre en oeuvre. L'architecture générée après synthèse est décrite au niveau RTL. Chaque processeur composant le réseau systolique est composé de manière identique à ses voisins. Il est composé de files de registres et d'unités fonctionnelles comme cela est décrit dans la Figure 0.11.

6. L'outil Catapult-C

L'outil Catapult-C est un outil de synthèse de haut niveau, développé par Mentor Graphics [MEN07]. Le point d'entrée de l'outil est du C++ ANSI écrit sans notion temporelle. Cela rend l'outil compatible SystemC. Le point de sortie de l'outil Catapult-C est une description de niveau RTL ciblant indépendamment les technologies ASIC et FPGA. Les langages cibles de niveau RTL sont VHDL et Verilog, les scripts générés ciblent les outils ModelSim, Design Compiler et Précision RTL.
L'outil autorise le concepteur à utiliser les formats de données présents dans le langage C/C++ à condition que ces formats soient entiers. Durant la phase de synthèse, en fonction des formats des données, un nombre de bits différents sera utilisé pour réaliser les calculs au sein du chemin de données (bool(1), char(8), short (16), int(32), long(32)). Il est aussi possible d'utiliser nativement les formats proposés par SystemC et qui sont plus flexibles (sc_int). Dans ce cas, l'utilisateur peut définir de manière exacte les largeurs de bits qu'il veut employer pour chacune de ses variables. Ces informations seront prises en considération au niveau de la synthèse.
Lors de la génération de la représentation interne, les techniques habituelles de propagation de constantes et d'analyse des bornes des boucles sont employées afin d'optimiser la description de manière automatique.

Les optimisations mémoire
Il existe plusieurs voies permettant de gérer la mémoire dans l'outil Catapult-C. Dans un premier temps, l'outil va détecter la présence de vecteurs ou de tableaux au sein de la description algorithmique. En fonction de leur taille, il va les convertir soit en files de registres, soit en mémoires internes ou mémoires externes. Une fois que ces données ont été mappées dans des bancs mémoires séparés, l'utilisateur peut décider d'appliquer un certain nombre de transformations pour optimiser l'architecture :

- *Sélection du type de mémoire* : l'utilisateur peut spécifier à l'outil avant synthèse quel est le type de mémoire qu'il faut mettre en oeuvre. Les différentes possibilités sont : file de registre multiplexée, mémoire mono-port ou multi-ports.
- *Fusion des bancs mémoire alloués* : une fois le type des mémoires spécifié, il est possible de fusionner différents tableaux au sein d'une même mémoire afin de réduire la taille des structures de contrôle ainsi que les éléments alloués et non utilisés.
- *Augmentation de la taille des bus* : afin d'augmenter le parallélisme au sein du chemin, l'outil va augmenter la taille du bus de données afin de transmettre n données contiguës en parallèle.

Les blocs de données de taille n qui sont maintenant les blocs de transfert élémentaires sont statiques en fonction du positionnement des données au sein du banc mémoire. Cette technique est intéressante dans les applications réalisant des accès contiguës tout au long de leur exécution (mais donc inutilisable pour les accès poinçonnés).

Interfaçage avec le reste du système

En ce qui concerne les interfaces de communication, l'outil utilise des méthodes permettant d'interfacer un grand nombre de ports d'entrées / sorties : streaming, mémoire simple ou double port, FIFO, bus AMBA ou tout autre composant de communication généré par le concepteur à l'aide de l'outil CatapultC library builder tool. Il est aussi possible de préciser à l'outil où sont mappées les données lorsque l'on souhaite utiliser des mémoires partagées afin de communiquer avec le système.

Utilisation des composants caractérisés

Afin de réaliser la sélection, l'outil va rechercher des composants dans des librairies. Ces derniers ayant déjà été caractérisés, des informations fiables sur leurs coûts en surface, délais, consommation, etc. leur sont associés. La synthèse se fait sous contraintes de ressources et d'interfaces. Lors de la synthèse, l'outil est capable de gérer les différentes largeurs de bits des données et opérations manipulées. Durant les différentes étapes de la synthèse, des conseils sont prodigués au concepteur sur les goulots d'étranglement présents dans le circuit. Cela doit lui permettre de modifier la description algorithmique manuellement afin de réduire les problèmes de limitation des accès mémoire (bande passante) et de dépendances dans les coeurs de boucle (empêchant l'exploitation du parallélisme).

7. Intégration d'IP dans une architecture ASIC : Mat2Colif/Mat-2-RTL

Dans [ZER06], les auteurs proposent un environnement conception basé sur l'intégration de blocs IP appelé : DSP Macrocell builder (cf. Figure 0.12). Ce flot est basé sur un flot de validation, un flot de conception matériel (incluant chemin de données et machines d'état finis pour le contrôle) et un flot de correction des délais ([TAM03]). Il est directement inspiré du flot Matlab-to-RTL ([ZER04]) définit et utilisé au sein de STMicroelectronics. Ce flot de conception est basé sur un modèle unifié pour la simulation et la synthèse d'un Système-sur-Puce, appelées « Colif », [CES01].

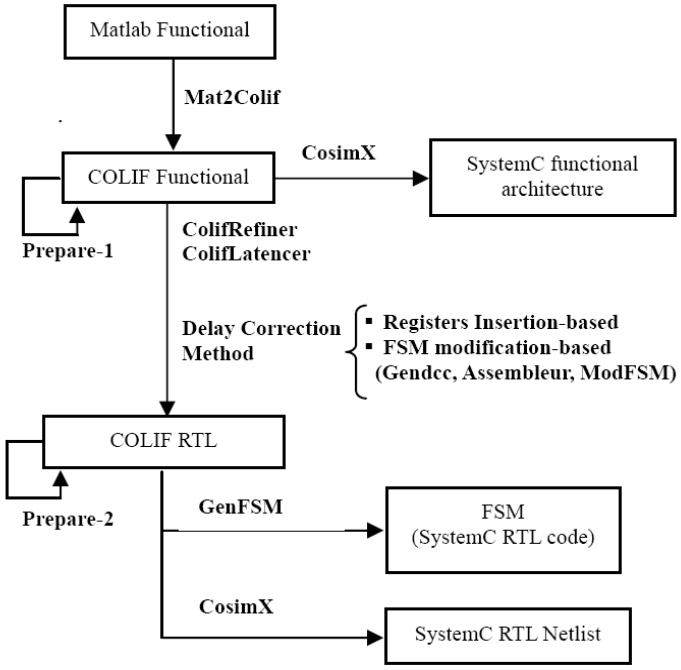


Figure 0.12 . *Flot de raffinement de DSP Macrocell builder*

En premier lieu, le concepteur utilise une bibliothèque d'IP de niveau fonctionnel pour décrire un modèle Matlab de son système. Il peut ainsi explorer efficacement l'espace de conception pour mettre au point son algorithme.
Puis, l'outil *Mat2Colif* transforme la description Matlab en une description Colif du système. Les valeurs de paramètres des IP sont extraites à partir du modèle fonctionnel validé, puis sont utilisées par *Prepare1* et *CosimX* afin de produire l'architecture en SystemC.

Cette représentation est alors raffinée à travers un flot de correction des délais (incluant ColifRefiner, ColifLatencer et ADCM), qui transforme le modèle fonctionnel Colif en un modèle Colif RTL corrigé. Les paramètres architecturaux sont employés pour instancier des IP RTL préconçus, décris en langage matériel (c.-à-d. VHDL, SystemC). L'outil génère automatiquement l'architecture RTL finale (incluant chemin de données et contrôle). Après simulation, l'architecture produite peut être implémentée à l'aide d'outil de synthèse logique et outils de placement/routage classiques.

Correction des délais dans l'approche Mat2Colif

Cet outil repose sur l'approche de correction des délais proposés dans [TAM03]. L'auteur propose une méthodologie et un flot semi-automatique (cf. Figure 0.13) d'aide à la conception et à la validation des macro-cellules ASIC dédié au traitement numérique du signal.

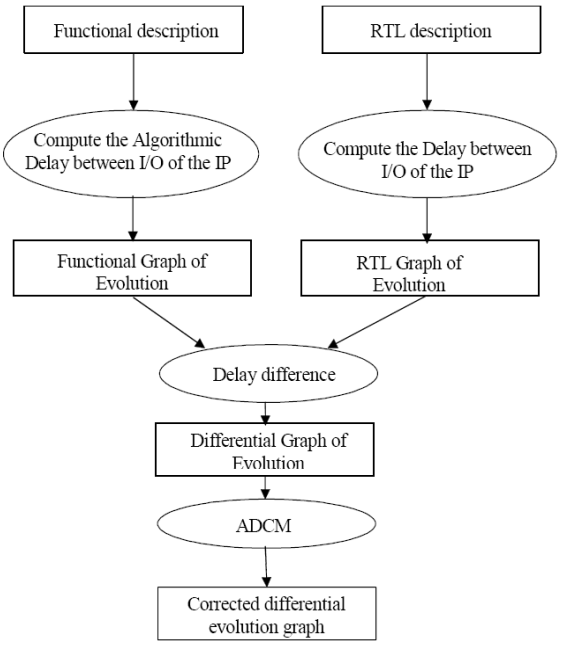
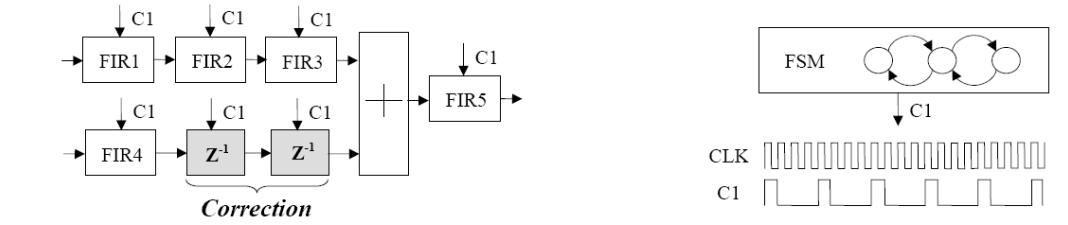


Figure 0.13 . *Flot de correction des délais*

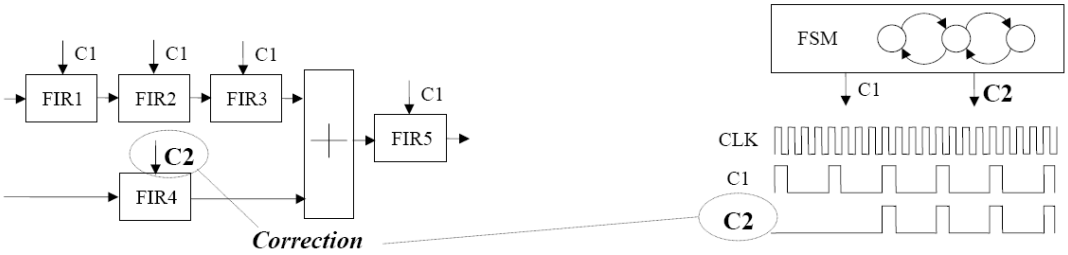
Ces travaux s'attachent à la correction de délais pouvant apparaître lors de la traduction au niveau transfert de registre d'une description fonctionnelle. Ces délais résultent des contraintes d'implémentation (registres de pipeline, temporisation des sorties, etc.). Ces erreurs comportementales sont provoquées par l'existence de délais induits dans sur le modèle RTL qui ne sont pas présents dans le modèle fonctionnel. Ceux-ci se produisent quand une application contient : des chaînes d'IPs convergeant vers une autre IP, des boucles de rétroaction, et/ou des IP à latences variables. Ce problème est plus généralement connu comme un problème de *retiming*.

On peut envisager trois approches pour résoudre ce problème :

- La première méthode implique l'insertion de protocole de synchronisation (par exemple poignée de main). L'avantage de ceci est que les problèmes de délais sont résolus avant l'assemblage des IPs RTL. L'inconvénient principal à cette technique est le coût en surface et en latence pour l'application. Toutefois, le problème de communication dans le cas de boucles de rétroaction, même avec un protocole « poignée de mains».
- La seconde solution repose sur l'insertion de registres entre les IPs pour temporiser les données (cf. Figure 0.14.a). Cette technique a l'avantage d'être non intrusive. La difficulté est alors de parvenir à déterminer les endroits où il y a problème, puis à déterminer exactement combien de registres doivent être insérés.
- La troisième méthode implique une modification du contrôleur initial en générant des signaux additionnels pour piloter les IPs (cf. Figure 0.14.b). Ces signaux sont décalés dans le temps par rapport aux signaux initiaux du contrôleur. Ils sont donc en mesure d'avancer ou de différer l'activation d'une IP. Cette technique repose sur les mêmes étapes d'analyse que la seconde solution (localiser les points conflictuels, déterminer combien de signaux doivent être décalés et de combien de cycles) et exige que le contrôleur initial soit modifié.



(a) Correction des délais par insertion de registres



b) Modification du controleur

Figure 0.14 . *Deux solutions pour la correction des délais induits*

Dans son approche, l'auteur mis au point une approche systématique appelée *Automatic Delay Correction Method* (ADCM) pour résoudre ces problèmes. L'ADCM exploite les deux dernières solutions proposées (insertion de registres et décalage temporel de signaux de contrôle).

Toutefois, on peut reprocher à cette approche la complexité de sa mise en œuvre (identification des points conflictuels, déterminer le coût de la correction). De plus cette solution ne permet que de temporiser des données. Ni les problèmes de ré-ordonnancement des données, ni les aspects multi-configurations ne sont pris en compte.

ANNEXE B
MODELES FORMELS POUR LA SYNTHESE DE HAUT NIVEAU

Un langage tel que le C ou le VHDL permet d'exprimer un nombre (trop) important de propriétés d'une spécification. Ces propriétés peuvent être ou non acceptées par un outil de synthèse de haut niveau (HLS) et suivant l'outil aboutir à des interprétations et donc des optimisations de l'implémentation différentes.
Les modèles couramment utilisés dans les outils de haut niveau (GAUT, Catapult-C, SPARK, MAHA, etc.) sont les modèles de type DFG, CFG et CDFG. Ces formalismes ayant été décrit dans le chapitre III, nous ne reviendrons pas dessus dans cette partie.
Les modèles calculatoires permettent idéalement l'expression de une ou plusieurs propriétés. Certains modèles sont plutôt appropriés à des applications dominées par les traitements (Data-Dominated) ou dominées par le contrôle (Control-Dominated).

1. Assignment Decision Diagram (ADD)

Le modèle de représentation nommé ADD (Assignement Decision Diagram) a été développé par D.Gajski en 1992 [CHA92]. Afin de réduire l'impact de l'écriture de la description sur le modèle de représentation, les structures conditionnelles sont représentées de manière équivalente à ce qui a été proposé pour étendre les sémantiques des DFG. Cela à pour effet de mettre à plat les différents calculs à effectuer dans chacune des branches conditionnelles entraînant une exécution spéculative puis un choix de résultat (Figure 0.1).

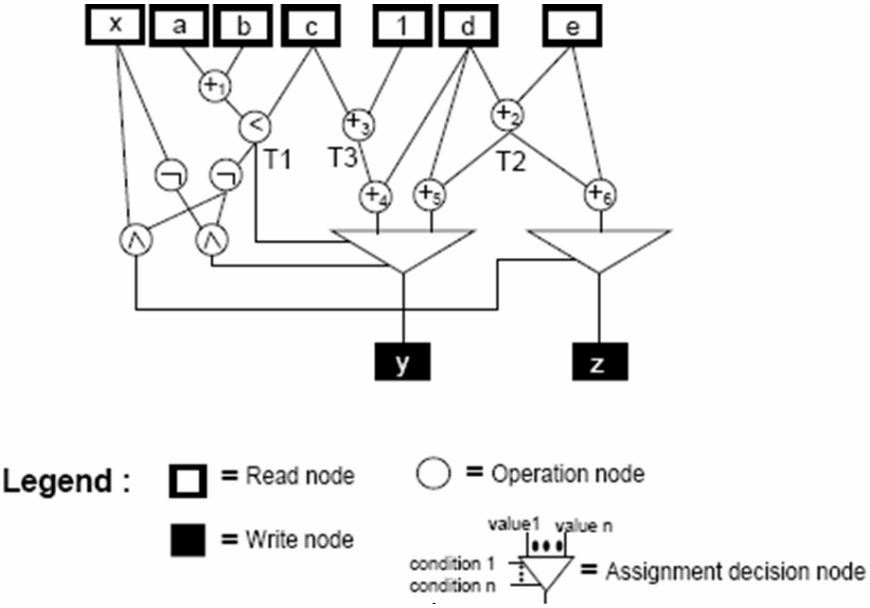


Figure 0.1 . *Exemple de modélisation à l'aide d'un ADD*

L'idée développée dans ce modèle vient du constat que pour paralléliser l'ensemble des opérations contenues dans les structures conditionnelles, il faut uniquement conditionner l'affectation des mémorisations des données et leur écriture sur les ports de sortie. Pour cela, Gajski développe un nouveau type de noeud nommé "noeud de décision d'assignation" (ADN), qui en fonction de ses paramètres, va décider quelle est la valeur à assigner au résultat. L'utilisation des noeuds ADN permet d'exécuter de manière spéculative toutes les opérations conditionnelles. Seules les affectations sont soumises aux résultats des conditions.

La représentation sous forme d'ADD est composée de 4 parties bien distinctes : l'assignation des résultats, la condition d'assignation des résultats, la décision d'assignation et les calculs des valeurs à assigner. Ces quatre parties sont composées à l'aide de 4 types de noeuds : les opérations, les lectures de données, les écritures de données et les conditions d'assignation.

2. Le modèle "Condition Graph" (CG)

Le graphe de condition nommé GC et défini par Juan dans [JUA94] est un graphe dont l'objectif premier est de représenter les conditions d'exécution des opérations. Le graphe prend comme point de départ une description comportementale représentée sous la forme d'un ADD. Dans la représentation ADD la condition d'exécution d'une opération est décrite comme une expression arithmétique qui se ramène à un résultat booléen *{true, false}*. Le graphe de condition est composé de 2 types de noeuds : les nœuds de données et les noeuds opérations. Les noeuds opérations représentent les opérations qui réalisent les calculs des conditions.

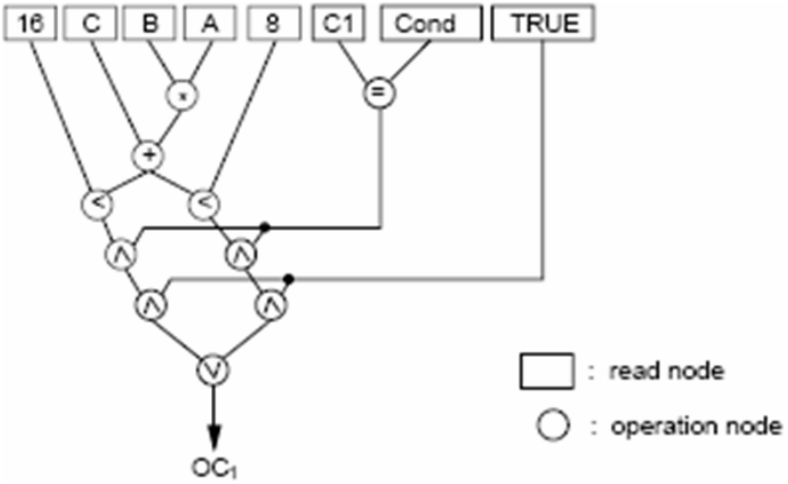


Figure 0.2 . *Exemple de modélisation à l'aide d'un graphe CG*

Une fois dérivée, le graphe de condition représente tous les calculs des conditions relatives aux opérations contenues dans le graphe ADD. Ce graphe est ensuite mis à profit dans une méthode d'identification des opérations mutuellement exclusives.

3. Les réseaux de Pétri

Un réseau de Petri (ou PN, pour Petri Net) [PET77] se représente par un graphe bipartite (composé de deux types de nœuds) orienté reliant des *places* P_i et des *transitions* T_i (les nœuds). Deux places ne peuvent pas être reliées entre elles, ni deux transitions. Les places peuvent contenir des *jetons*, représentant généralement des ressources disponibles. La distribution des jetons dans les places est appelée le *marquage* du réseau de Petri (cf. Figure 0.3).

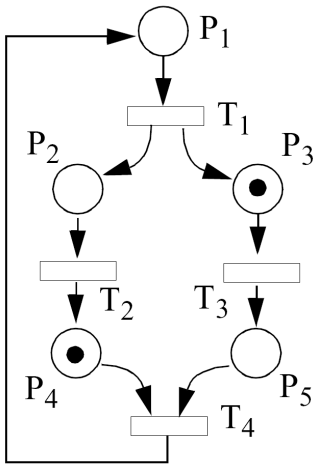


Figure 0.3. *Exemple de réseau de Petri*

Les entrées d'une transition sont les places desquelles part une flèche pointant vers cette transition, et les sorties d'une transition sont les places pointées par une flèche ayant pour origine cette transition.

L'état d'un tel réseau est défini par un marquage, constitué par un ensemble de jetons répartis sur les différentes places.

L'évolution d'un tel système se fait par le franchissement des transitions : on retire un jeton aux places situées en amont de la transition et on les ajoute aux places en aval de la transition. Un tel système doit cependant respecter un certain nombre de contraintes :

- Pour qu'une transition soit franchie, il faut que toutes les places en amont possèdent au moins un jeton.
- Le réseau ne peut évoluer que par le franchissement d'une seule et unique transition à un instant donné.
- Le franchissement d'une transition est d'une durée nulle et indivisible.

Deux propriétés intrinsèques importantes des réseaux de Pétri sont leur nature asynchrone et la concurrence. Les événements peuvent arriver de manière complètement indépendante ; il n'y a donc pas d'horloge pour piloter les transitions. Souffrant comme les FSM de l'absence de hiérarchie, des extensions aux réseaux de Pétri hiérarchisés (HRDP) ont été proposées. La nature totalement asynchrone des réseaux peut être modifiée par l'utilisation des réseaux de pétri temporisés. De nombreuses variantes des réseaux de Pétri ont été proposées dans la littérature, permettant d'intégrer des propriétés particulières.

En l'état, ce modèle se révèle trop contraint pour pouvoir prétendre répondre à nos attentes. En effet, au cours d'une séquence de communication, plusieurs données peuvent être transmises en même temps, ce que l'on ne peut modéliser avec un PN du fait qu'à un cycle donné, une seule transition peut être franchie.
De plus, nous devons pouvoir modéliser plusieurs comportements différents dans le cas d'architectures multi-modes. Ceci n'est pas non plus possible avec un réseau de Pétri.

4. Les autres modèles de représentation

D'autres modèles de représentation existent et sont dédiés à la modélisation de propriétés précises. On peut citer :

– Les réseaux de processus de Khan (KPN, pour *Kahn Process Networks*) [KAH74] sont utilisés dans la modélisation de programmes contenant des parties calculatoires parallèles. Il est en partie basé sur les sémantiques des CDFG. Les réseaux de processus de Kahn sont un cas particulier de réseaux de processus dans lesquels un processus ne peut pas savoir si une file de message est vide ou non, et est bloqué en attente lorsqu'il cherche à obtenir un message à partir d'une file vide. Ces restrictions garantissent que le réseau est déterministe, c'est-à-dire ses calculs ne dépendent que de sa topologie, indépendamment de l'implémentation qui en est faite.
– Les machines d'états finis (ou FSM pour *Finite State Machine*), sont connues et très utilisées en informatique (FSMC, FSMD, ACFSM, ECFSM…). Ces modèles se basent sur un ensemble de

d'états, de transition et d'actions associées aux états et/ou aux transitions. Ces modèles, s'ils sont relativement bien adaptés pour la description d'architectures simples, peuvent devenir extrêmement lourds pour des architectures plus complexes**.**

ANNEXE C

DIFFERENTES SOLUTIONS POUR LA GENERATION D'UN CONTROLEUR

Au cours de nos expérimentations, nous avons constaté que la modélisation en VHDL des machines de contrôle avait un impact important non seulement sur la surface du contrôleur dans le circuit après synthèse par Design Compiler Ultra (Synopsys), mais également sur les temps de synthèse par ce même outil. Nous illustrons cette analyse par l'exemple d'un contrôleur pilotant les accès en lecture/écriture d'un ensemble d'éléments mémorisants.

```
entity STARexempleCTRL_FSM_CP is
generic (
    Insts : integer;          -- Taille en nombre de bits du mot de contrôle
    Cycles : integer          -- Taille en nombre de bits du signal comptant les cycles  ) ;
port (
    EN : in  std_logic;       -- Signal d'activation du controleur
    Nrst : in  std_logic;     -- Signal reset (actif bas)
    Cycle : in  std_logic_vector (Cycles-1 downto 0); -- Signal en provenance du compteur de cycles
    MINST : out std_logic_vector (Insts-1 downto 0)  -- Message de contrôle  ) ;
end STARexempleCTRL_FSM_CP;

architecture fsm_v1 of STARexempleCTRL_FSM_CP is

signal Message : std_logic_vector(Insts-1 downto 0) := (others => 'Z');

begin  -- fsm_v1
     Message <=  (others => '0') when (Nrst = '0') else
                 "1000000" when (conv_integer(Cycle) = 5) else
                 "1100010" when (conv_integer(Cycle) = 7) else
                 "0110000" when (conv_integer(Cycle) = 8) else
                 "0110111" when (conv_integer(Cycle) = 9) else
                 "1000101" when (conv_integer(Cycle) = 10) else
                 "0011001" when (conv_integer(Cycle) = 13) else
                 "0001011" when (conv_integer(Cycle) = 15) else
                 "0001001" when (conv_integer(Cycle) = 16) else
                 (others => '0') ;
    MINST <= Message;
end fsm_v1;
```

Figure 0.1. *Syntaxe d'un contrôleur construit autour du mot de commande*

Une première solution consiste à utiliser la syntaxe proposée dans la Figure 0.1. Dans cette approche, les différentes valeurs des signaux de commande sont directement affectées au mot de commande en fonction de signaux d'entrée. Ici, le signal pilotant l'affectation des valeurs du signal de commande est un signal provenant d'un compteur de cycles. Ainsi, selon le cycle auquel on se trouve, le mot de commande pilotera différemment les éléments qui dépendent de lui.

```
entity STARexempleCTRL_FSM_CP is
generic (
    Insts : integer;            -- Taille en nombre de bits du mot de contrôle
    Cycles : integer            -- Taille en nombre de bits du signal comptant les cycles  ) ;
port (
    Clk : in std_logic;         -- Signal d'horloge
    EN : in  std_logic;         -- Signal d'activation du controleur
    Nrst : in  std_logic;       -- Signal reset (actif bas)
    Cycle : in  std_logic_vector (Cycles-1 downto 0);       -- Signal en provenance du compteur de cycles
    MINST : out std_logic_vector (Insts-1 downto 0)         -- Message de contrôle  ) ;
end STARexempleCTRL_FSM_CP;

architecture fsm_v2 of STARexempleCTRL_FSM_CP is
    signal Message : std_logic_vector(Insts-1 downto 0) := (others => 'Z');
    type type_etat is ( EtWait, Et0, Et1, Et2, Et3, Et4, Et5, Et6, Et7);
    signal Etat : type_etat;

begin
    --fsm V2 : pas de ROM
    FSM : process(Clk, Nrst)
    begin
        if (Nrst = '0') then
            Message <= (others => '0');
            Etat <= EtWait;
        else
            if (Clk'event and Clk = '1') then
                if (EN = '1') then
                  case Etat is
                        when    Et0 => Message <= ”1000000”;        Etat <= EtWait;
                        When    Et1 => Message <= ”1100010”;        Etat <= Et2;
                        when    Et2 => Message <= ”0110000”;        Etat <= Et3;
                        when    Et3 => Message <= ”0110111”;        Etat <= Et4;
                        when    Et4 => Message <= ”1000101”;        Etat <= EtWait;
                        when    Et5 => Message <= ”0011001”;        Etat <= EtWait;
                        when    Et6 => Message <= ”0001011”;        Etat <= Et7;
                        when    Et7 => Message <= ”0001001”;        Etat <= Et0;
                        when    EtWait => Message <= (others => '0');
                                if (conv_integer(Cycle) = 3) then
                                    Etat <= Et0;
                                elsif (conv_integer(Cycle) = 5) then
                                    Etat <= Et1;
                                elsif (conv_integer(Cycle) = 11) then
                                    Etat <= Et5;
                                elsif (conv_integer(Cycle) = 13) then
                                    Etat <= Et6;
                                else
                                    Etat <= EtWait;
                                end if;
                  end case;
                else
                    Message <= '0';
                end if;
            end if;
        end if;
    end process FSM;

    MINST <= Message;
end fsm_v2;
```

Figure 0.2. *Syntaxe d'un contrôleur construit avec un automate d'état finis*

La première solution génère des contrôleurs très optimisés, mais les temps de synthèse avec l'outil DC Ultra augmentent de façon exponentielle avec la complexité du contrôleur. La synthèse de cet exemple pédagogique prend à elle seule (hors synthèse du reste du design) 1 à 2 minutes. Des exemples plus complexes (mot de commande de plus de 100 bits et un nombre de branchements conditionnels de l'ordre de 50 à 200), DC peut mettre plusieurs jours avant de converger vers une solution. Ce type d'architectures complexes s'est présenté au cours des expérimentations sur l'entrelaceur UWB (cf. Chapitre V).

Pour accélérer les temps de synthèse avec l'outil DC Ultra, nous avons utilisé une architecture plus classique pour implémenter le contrôleur (cf. Figure 0.2). Dans cette approche, nous utilisons un automate d'états finis et selon l'état courant, le signal de commande est affecté avec le mot de commande adéquat.
Grâce à cette solution, les temps de synthèse des architectures les plus complexes sont réduits à quelques minutes. Toutefois, la surface totale du composant obtenu et plus importante (de 10% à 15% selon les expériences) que celle du composant utilisant la solution de la Figure 0.1.

Il serait intéressant de poursuivre cette étude et de l'étendre à d'autres outils de synthèse matérielle (ASIC ou FPGA) afin de déterminer si ces résultats sont dus à une particularité de Design Compiler Ultra, ou bien s'ils sont caractéristiques d'une complexité combinatoire ou architecturale plus forte selon l'approche retenue.

ANNEXE D
EXEMPLE DE FICHIER DE CONTRAINTES DE COMMUNICATION

Nous présentons ici un exemple de fichier de communication (cf. Figure 0.1). On trouve dans ce fichier : les informations relatives à la bibliothèque technologique sélectionnée (Nombre de cycles de lecture/écriture des FIFOs/LIFOs ; le nombre de modes de fonctionnement de l'application ; les paramètres architecturaux (taille, nombre et liste des ports de l'IP ciblée, et des ports du système connectés à cette IP) ; et enfin, le nombre de données de chaque mode et la liste des données et de leurs différentes dates d'accès (lecture/écriture) depuis ou vers différents ports.

```
#File generated by  StarTor V2.2
#Build date : Jun 2007
cadence = 250
clock = 10
FIFO_lat_W = 0
FIFO_lat_R = 1
LIFO_lat_W = 0
LIFO_lat_R = 2
```

Paramètres technologiques

```
NbreCONFIG : 1
```

Nombre de modes

```
LIST_PORT_IP : 3
#Num    Way     Width   Group
 1      OUT     8       0
 2      OUT     8       1
 3      OUT     8       2

LIST_PORT_SYS : 2
#Num    Way     Width   Group
 1      IN      8       0
 2      OUT     8       1
```

Paramètres architecturaux

```
CONFIG : 0
LISTDATA : 9
#Data  Width  Port  Order_PortSys  Bus  Order_PortIP  Input  Output
a1     8      1     0              1    0             10     50
b1     8      2     0              3    0             20     90
c1     8      1     0              2    0             30     80
d1     8      1     0              2    0             40     90
e1     8      1     0              1    0             50     70
f1     8      2     0              3    0             60     70
g1     8      1     0              1    0             90     100
h1     8      1     0              3    0             100    150
i1     8      1     0              2    0             110    130
FIN
```

Contraintes de communication

Figure 0.1. *Exemple de fichier de contraintes de communication*

Sur cet exemple, l'application n'a qu'un seul mode de fonctionnement (Config 0) qui manipule 14 données.

On trouve également un champ concernant l'ordre (Order_PortSys, Order_PortIP) de la donnée sur le port. Ceci permet, dans le cas où sur un port plusieurs données peuvent transiter en parallèle, de déterminer avec certitude où se trouve la donnée concernée sur le port (Si on trouve *n* données en parallèle sur un port, la donnée d'ordre *i* se trouve sur le $i^{ième}$ mot du port en partant des poids faibles). Sur notre exemple, les données font 8 bits de large, tout comme les ports de données, il n'y a donc qu'une seule donnée qui transite sur ce port.

ANNEXE E
EXEMPLE DE CODE VHDL GENERE

Nous présentons ici un exemple de code VHDL généré par notre flot de synthèse de haut niveau. On trouve successivement : l'ensemble des constantes utilisées dans le code VHDL ; les descriptions des architectures et des entités des éléments mémorisants et des éléments de multiplexage ; les descriptions des différentes machines de contrôle ; et la description de l'interconnexion de ces différents éléments pour former l'architecture finale.

ANNEXE F
INTERFACE UTILISATEUR

La Figure 0.1 représente les différents éléments constituant cette interface. On observe ainsi trois menus déroulants (*File*, *View*, *Metrics*) et de deux zones distinctes, l'une indiquant l'étape courante du flot de conception, et l'autre offrant les interfaces utilisateur de chacun de nos outils de conception (tabulations).

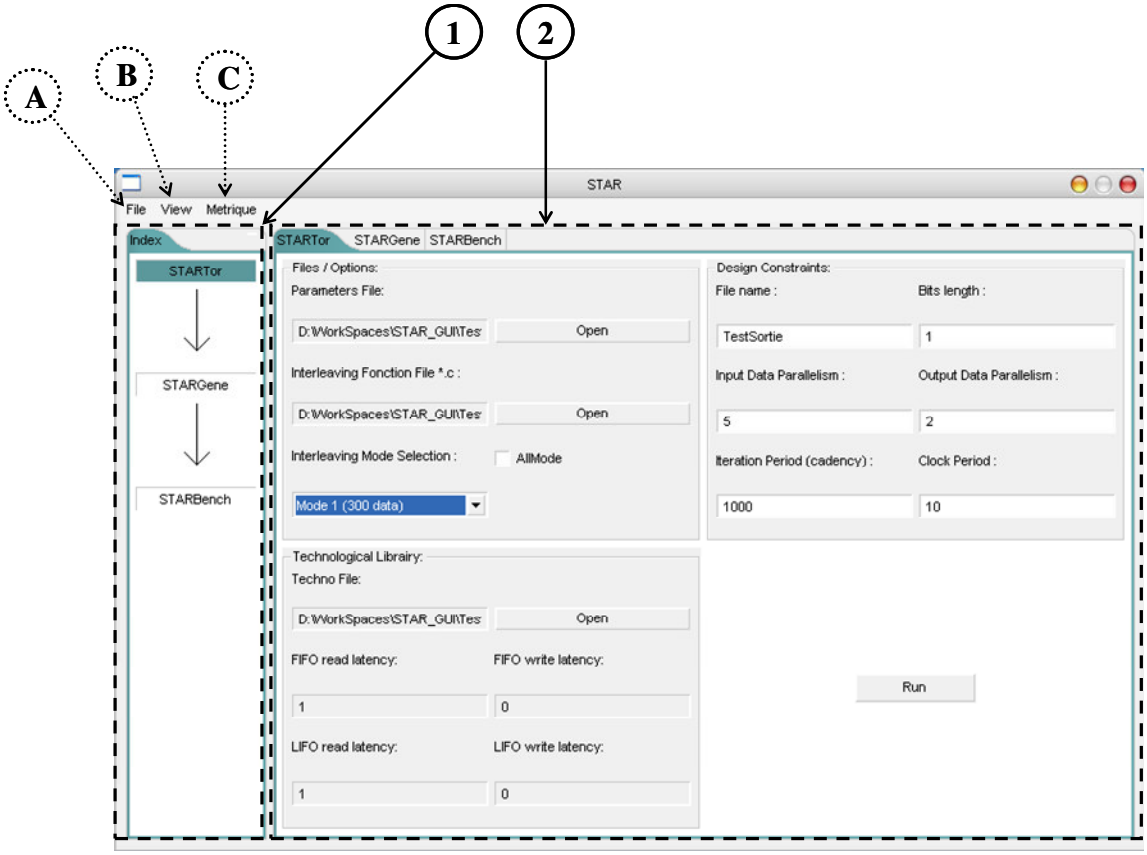


Figure 0.1. *Description générale de l'interface utilisateur*

A - Menu *File* : ce menu permet à l'utilisateur de paramétrer des répertoires de travail.

B - Menu *VIew* : ce menu permet à l'utilisateur de passer de l'interface dédiée aux outils de synthèse, à celle de l'outil d'exploration/analyse des résultats.

C - Menu *Metrics* : ce menu permet à l'utilisateur de sauvegarder ou de charger un ensemble de métriques d'exploration.

1 - Ce cadre permet de suivre le déroulement du flot de conception STAR sur une représentation schématique de ce flot. L'étape courante apparaît en surbrillance. La version actuelle de l'interface ne permet pas encore d'utiliser l'outil StarDFG qui sera intégré prochainement.

2 - Dans cette fenêtre les interfaces dédiées aux trois outils intégrés sont accessibles sous la forme de tabulations.

Sur cette figure, la représentation schématique du flot de conception nous indique que nous sommes face à l'interface dédiée à l'outil StarTor. En effet, la tabulation activée est bien celle correspondant à cet outil.

Extraction des contraintes de communication

L'interface utilisateur de l'outil StarTor (cf. Figure 0.2) se décompose en trois zones concernant : la sélection de la description en langage C de l'algorithme de brassage de données, la sélection d'une bibliothèque technologique et le paramétrage des contraintes de débit et de latence.

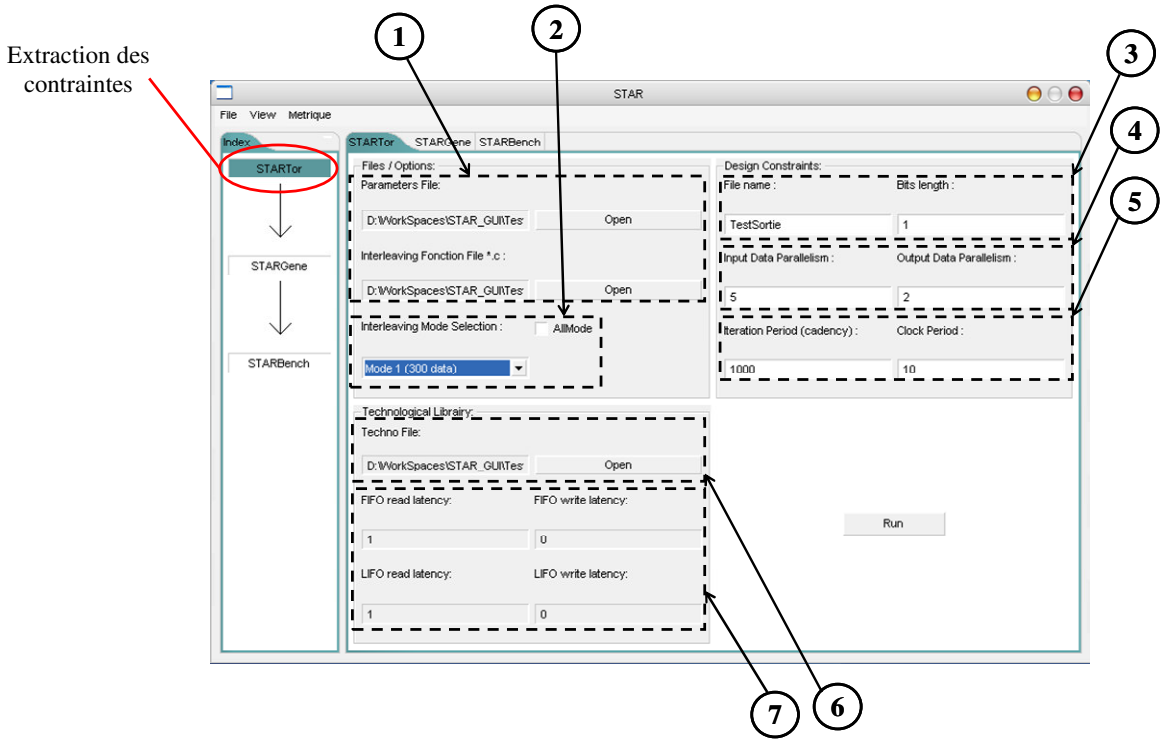


Figure 0.2. *Interface utilisateur de l'outil StatTor*

1 - Sélection du fichier C décrivant la fonction d'entrelacement et du fichier de paramètres associé (nombre de modes de fonctionnement, taille des trames…).

2 - Une fois le fichier de paramètres chargé, ce menu déroulant va permettre, dans le cas d'architecture multi-modes, de sélectionner l'un des modes de fonctionnement. Ceci permet à l'utilisateur, s'il le souhaite, d'explorer séparément les différents modes de son application. Il peut également demander la génération des contraintes pour tous les modes de fonctionnement en sélectionnant la boite de dialogue *All Mode*.

3 - Ces cases permettent de nommer le fichier de sortie et de définir la taille, en bits, des données scalaires qui seront manipulées.

4 - Ces paramètres permettent à l'utilisateur de définir des contraintes de débits en entrée et en sortie de l'entrelaceur. Pour l'instant, ces débits ne sont exprimés qu'en nombre de données entrant/sortant en parallèle dans/de l'application.

5 - Ces paramètres permettent de définir une contrainte de cadence et la période de l'horloge (en ns) qui servira à faire fonctionner l'application.

6 - Ce bouton permet de sélection une bibliothèque technologique de composants FIFO, LIFO et Registres.

7 - Affichage des caractéristiques temporelles des éléments mémorisants sélectionnés à l'aide de **6**.

Si au cours de son analyse (cf. chapitre IV) l'outil StarTor constate que la contrainte de cadence spécifiée par l'utilisateur est trop forte, ce dernier en est informé par un message spécifique.
A terme, cette interface permettra également d'automatiser l'exploration architecturale par différents débits applicatifs (différents parallélismes d'entrée/sortie).
Une fois les contraintes de communication générées, l'outil active directement l'interface utilisateur de l'outil StarGene.

Synthèse du composant STAR

L'interface de StarGene est la plus riche car elle doit permettre d'automatiser l'exploration de l'espace des solutions en jouant sur les paramètres de conception. Ce panel se décompose en quatre zones : la première, *File/Option*, est dédié à la sélection des fichiers de contraintes de communication et de la bibliothèque technologique (Si l'utilisateur vient de générer les contraintes avec l'interface StarTor, ces champs sont automatiquement pré-remplis) et à la sélection des algorithmes d'exploration multi-modes ; la seconde zone, *Storage Element*, permet de sélectionner le type d'élément mémorisant qui sera assigné ; la troisième, *Binding*, regroupe les métriques liées à l'étape d'assignation des éléments mémorisants ; enfin, la dernière, *Opt Fct*, permet de sélectionner et de paramétrer les algorithmes de l'étape d'optimisation de notre approche.

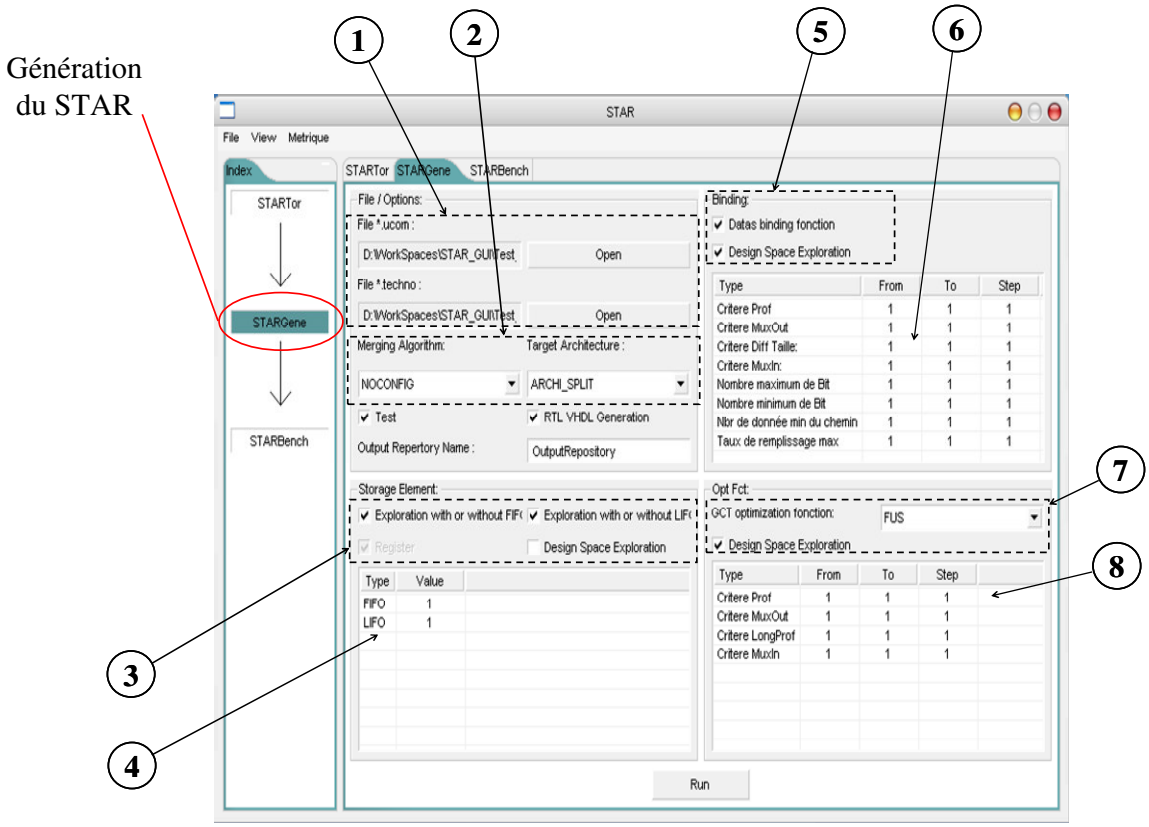


Figure 0.3. *Interface utilisateur de l'outil StarGene*

1 – Sélection du fichier de contraintes de communication à explorer, et sélection de la bibliothèque technologique.

2 - Sélection des algorithmes de fusion des modes (dans le cas d'architecture multi-modes nous proposons plusieurs algorithmes de fusion) et de l'architecture cible (avec ou sans mécanisme de gel).

3 - Sélection des éléments mémorisants autorisés pour l'assignation. L'utilisateur peut générer une architecture sans FIFO ni LIFO s'il le souhaite. Une boite de dialogue (*Design Space Exploration*) permet également de déterminer si l'utilisateur souhaite explorer les métriques mises en jeu ici.

4 - Métriques permettant à l'utilisateur de définir une priorité entre l'assignation d'éléments FIFO et l'assignation d'élément LIFO. Si l'utilisateur choisit d'explorer différents paramétrages de ces valeurs, il peut le faire directement ou en sélectionnant la boite de dialogue présenté en **2** (dans ce cas la table de **4** est modifiée comme dans **6**).

5 - Activation de l'algorithme d'assignation.

6 - Table d'exploration des métriques d'assignation. Pour chaque métrique, l'utilisateur peut définir une borne supérieure, une borne inférieure et un pas d'exploration. L'outil StarGene explorera automatiquement toutes les combinaisons de ces métriques.

7 - Sélection de l'algorithme d'optimisation. L'utilisateur peut choisir entre l'algorithme d'optimisation proposé dans le chapitre IV ou différentes versions du Left-Edge.

8 - Table d'exploration des métriques d'optimisation.

Le flot de synthèse de l'outil StarGene impose un séquencement strict des étapes de synthèse :

- *Fusion des différents modes de fonctionnement et sélection de l'architecture cible*. Plusieurs algorithmes de fusions sont proposés à l'utilisateur : fusion naïve (agrégat des différents modes) ou fusion des graphes selon l'approche MMRCG. Les architectures cibles peuvent ou non, intégrer un mécanisme de gel.
- *Assignation des éléments mémorisants* sélectionnés dans la bibliothèque technologique.
- *Optimisation.* Plusieurs algorithme d'optimisation sont ici proposés : optimisation des graphes MMRCG ou bien optimisation par un algorithme de type Left-Edge.

Validation du composant STAR

L'interface de StarBench permet de lancer automatiquement génération du banc de test pour la validation du composant généré à l'aide de StarGene.

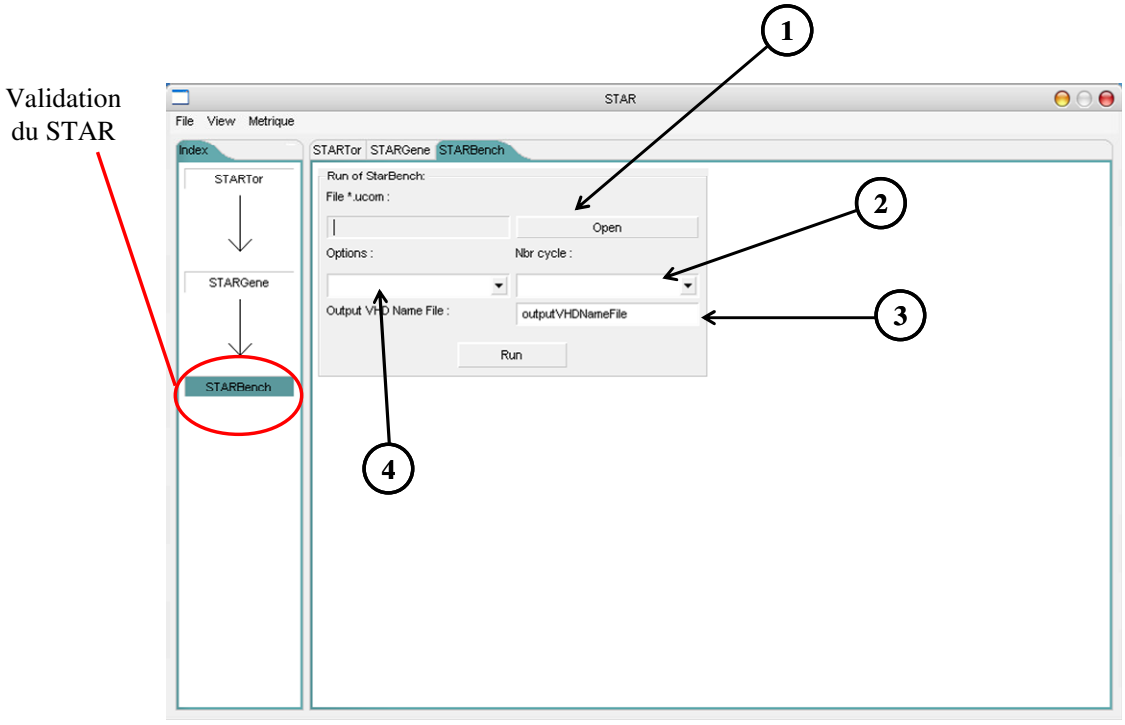


Figure 0.4. *Interface utilisateur de l'outil StarBench*

1 - Sélection du fichier de contraintes de communication.
2 - Nombre d'itération de simulation souhaité.
3 - Nom du fichier de sortie.
4 - Sélection du type de simulation souhaitée (Aléatoire ou prédéfinie).

L'utilisateur peut déterminer le nombre de cycles de simulation et le type de simulation (aléatoire ou selon un scénario prédéfinit) qu'il souhaite effectuer. Le banc de test généré est alors utilisable avec des outils de simulation VHDL du commerce (ModelSim, NCSim…).

Analyse des résultats

L'interface utilisateur présente un atout non négligeable : une interface dédiée à l'exploration des résultats de synthèse. Cette interface doit permettre à l'utilisateur de naviguer dans l'ensembl des résultats, de les comparer, de sélectionner les architectures qui feront l'objet d'analyses plus poussées…

Ce panel se décompose en deux zones : la première, permet l'affichage des résultats sous forme graphique (nuages de points, colonnes de niveau) ; la seconde permet à l'utilisateur de déterminer où se trouve les résultats et comment les afficher.

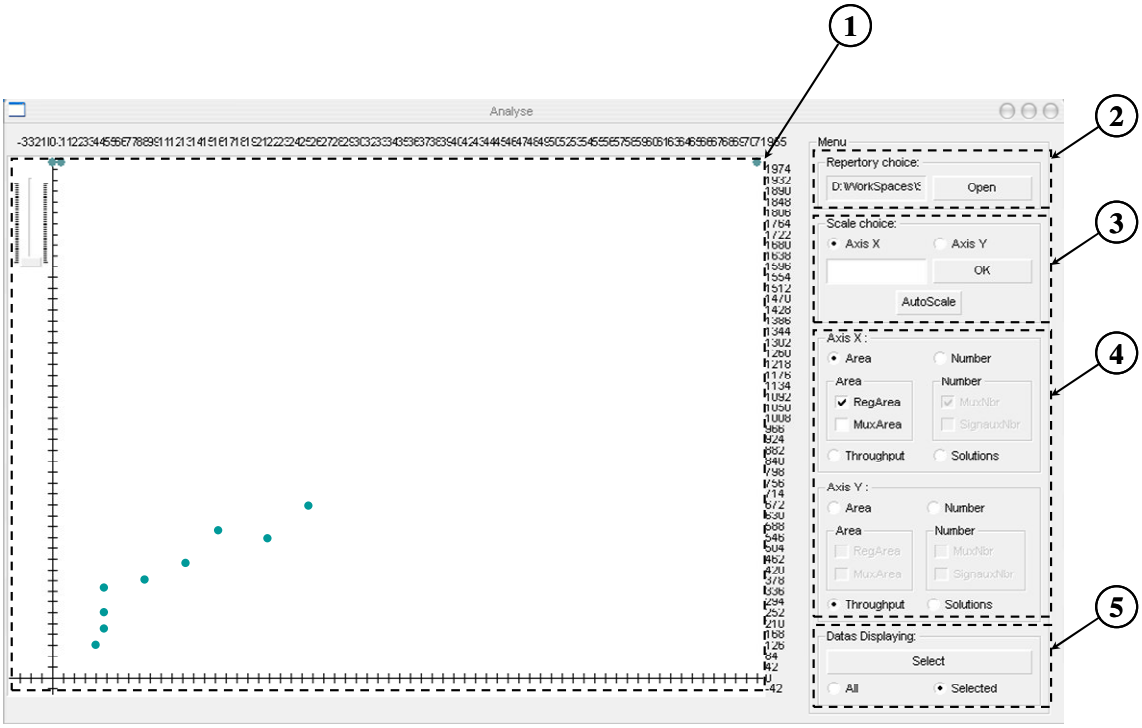


Figure 0.5. *Interface utilisateur de l'outil StarBench*

1 – Panneau d'affichage des graphiques.

2 – Sélection du répertoire contenant les résultats (Automatiquement pré-rempli si les architectures ont été générées avec l'interface).

3 – Réglage de l'échelle du graphique.

4 - Sélection des informations à afficher (en abscisse et en ordonnée).

5 - Sélection des d'architectures sur le graphique.

L'utilisateur peut déterminer à loisir les informations qu'il souhaite croiser sur le graphique. Il peut ainsi choisir d'afficher la surface, le débit, le nombre d'éléments à contrôler ou bien lister les différentes solutions architecturales.

Il est également possible d'accéder à une architecture, et aux informations qui la concernent, en cliquant directement sur le graphique.