Synthèse d'Interface de Communication pour les Composants Virtuels

Philippe Coussy

N° d'ordre : 27

THESE

pour obtenir le grade de :

DOCTEUR DE l'UNIVERSITE DE BRETAGNE SUD

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

PHILIPPE COUSSY

Synthèse d'Interface de Communication pour les Composants Virtuels

Soutenue publiquement le 10 décembre 2003

COMPOSITION DU JURY

A.A. JERRAYA	Laboratoire TIMA / SLS	Président
F. PETROT	ASIM, Université Paris 6	Rapporteur
O. SENTIEYS	ENSSAT, Université de Rennes 1	Rapporteur
J.P. CALVEZ	IRESTE, Ecole Polytechnique de Nantes	Examinateur
E. MARTIN	LESTER, Université de Bretagne Sud	Directeur de thèse
A. BAGANNE	LESTER, Université de Bretagne Sud	Examinateur
H. MICHEL	STMicroelectronics	Invité

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

Remerciements

Je remercie tout particulièrement Monsieur Eric Martin, Professeur des Universités à l'Université de Bretagne Sud et Directeur du LESTER, pour m'avoir accueilli dans son équipe de recherche et pour avoir accepté d'être mon directeur de thèse. Je souhaite aussi lui exprimer toute ma reconnaissance pour sa gentillesse, son soutien, ses conseils, sa disponibilité et son encadrement qui a été sans faille.

Je remercie Monsieur Adel Baganne, Maître de Conférence à l'Université de Bretagne Sud, pour ses précieux conseils sur la rédaction scientifique sans lesquels les publications auraient été moins nombreuses.

Je remercie Monsieur Ahmed Amine Jerraya, Directeur de recherches CNRS au laboratoire TIMA de l'INPG de Grenoble, pour m'avoir fait l'honneur d'être président du jury.

Je remercie Monsieur Frédéric Petrot, Maître de Conférence au laboratoire LIP6 de l'Université Paris VI et Monsieur Olivier Sentieys, Professeur des Universités à l'ENSSAT, Université de Rennes I, pour avoir accepté d'être rapporteur de ce manuscrit.

Je remercie Monsieur Jean Paul Calvez, Professeur des Universités à l'Ecole Polytechnique de Nantes, et Monsieur Henri Michel, responsable de l'équipe SHIVA (Signal Processing Hardware Implementation and VAlidation) de la société STMicroelectronics pour avoir accepté d'examiner ces travaux de recherche.

Je tiens à exprimer ma gratitude à Emmanuel Casseau et Guillaume Savaton pour leur gentillesse, leur soutien, leur écoute et leurs conseils.

Je remercie également Pierre Bomel, Dominique Heller et Mikael Le Bohec pour le travail qu'ils ont effectué sur l'outil GAUT et sans lequel les expériences n'auraient pu être menées à bien.

Merci à tous les membres du laboratoire LESTER pour leur disponibilité et leur bonne humeur.

Enfin, je tiens à exprimer ma profonde reconnaissance à Léone pour avoir su me supporter durant ces longs mois.

A mes parents, ma sœur

Résumé

Actuellement, des systèmes complets, contenant une partie logicielle et une partie matérielle, sont intégrés sur une même puce nommée Système-sur-Silicium (SoC). Pour faire face à la complexité d'intégration et maîtriser les contraintes, les équipes de recherche proposent de nouvelles méthodologies de conception qui reposent sur (1) la réutilisation de blocs logiciels ou matériels préconçus *(Composant Virtuel VC, Intellectual Property IP)*, (2) sur l'élévation du niveau de description des applications (System-Level Design) et (3) sur l'orthogonalisation de différents aspects (fonctionnalité/architecture, calculs/communication, besoins/contraintes/choix d'implantation). Malheureusement la difficulté de réutilisation liée au niveau de description RTL (Register Transfer Level), auquel sont fournis les IPs, ne permet pas une intégration aisée et optimisée. Ainsi, les méthodologies d'intégration à base d'adaptateurs additionnels sont souvent inadaptées aux contraintes et à la spécificité des algorithmes utilisés dans le domaine du traitement du signal et de l'image (TDSI). Elles peuvent de ce fait aboutir à la violation des contraintes en terme de surface, consommation et performances du système.

Nous proposons dans ce mémoire une approche de réutilisation des IPs dans les applications orientées traitement du signal, de l'image et des télécommunications. Pour cela, nous basons notre approche sur la notion de composants virtuels de niveau algorithmique, définie dans le cadre des projets *RNRT MILPAT (Méthodologie et Développement pour les Intellectual Properties pour Applications Telecom)*. Le flot de conception proposé s'inscrit dans la démarche Adéquation Algorithme Architecture du projet *RNRT ALITPA (Définition et Application d'une méthodologie de développement pour les (IP) intellectual property de niveau comportemental dans les applications de télécommunication)* et est basé sur l'utilisation de techniques de synthèse haut niveau sous contraintes d'intégration. Les unités fonctionnelles constituant l'architecture cible du composant sont (re)conçues en fonction des caractéristiques de l'architecture de communication du système et de la spécificité de l'application.

Dans ce contexte, la spécification de l'IP est modélisée par un Graphe Flot de Signaux *(SFG)* qui, couplé aux temps de propagations des opérateurs et à la cadence d'itération, permet la génération d'un graphe de contrainte algorithmique *ACG*. Nous avons développé une d'analyse formelle des contraintes, qui repose sur les calculs de cycles, et permet de vérifier la cohérence entre la cadence, les dépendances de données de l'algorithme et les contraintes technologiques.

Les contraintes d'intégration, spécifiées pour chacun des bus (ports) connectants l'IP aux autres composants du système, sont modélisées par un graphe de contraintes d'Entrée/Sortie *IOCG (IO Constraint Graph)* dont la sémantique est issue des travaux de Ku et De Micheli. Ce modèle supporte, entre autre, la modélisation (1) du type de transferts, (2) des varations temporelles des dates d'arrivées des données, (3) du séquencement des données échangées (4) des mécanismes liés aux protocoles. Les contraintes d'intégration et les contraintes algorithmiques de l'IP sont fusionnées pour fournir un graphe détaillé des contraintes *GCG (Global Constraint Graph)*

exhibant les points de synchronisation entre l'environnement et le composant. Des optimisations pour l'implémentations sont proposées à partir de transformations formelles du graphe.

La synthèse de l'unité de traitement *(UT)* est réalisée à l'aide de l'outil *GAUT* (Générateur Automatique d'Unité de Traitement) dont l'ordonnancement est contraint par les paramètres temporel de l'intégrateur du composant virtuel. La synthèse de l'*UT* génère un ensemble de contraintes au E/S, modélisé sous la forme d'un *IPERM (IP Execution Requirement Model)*. Ce dernier modélise (1) les dates de production et de consommation des données dans l'UT et (2) les bus sur lesquels elles transitent entre l'unité de communication et l'unité de traitement. Les modèles *IPERM* et *IOCG* sont finalement utilisés pour synthétiser l'unité de communication.

Nous avons appliqué notre méthode à des algorithmes des domaines du TDSI et des Télécommunications. La première expérience est réalisée sur un exemple de Transformée de Fourrier Rapide (*FFT)*. Pour les conditions d'expérimentation retenues, l'optimisation du nombre d'opérateurs est en moyenne de 20% et celle des registres de 7%, par rapport à une approche classique. La deuxième expérience utilise une Transformée en Cosinus Discrète *(DCT)* pour comparer les résultats, obtenus en appliquant l'approche d'intégration proposée dans ce manuscrit, avec les résultats des méthodes utilisant des adaptateurs. Pour l'exemple considéré, le gain sur les registres de communications varie de -2% à 88% à débit d'E/S constant. La dernière expérience, réalisée en partenariat industriel, démontre l'applicabilité de notre méthodologie sur un composant virtuel comportemental complexe (Maximum A Posteriori MAP) dans le contexte d'une application de Turbo décodage temps réel.

Abstract

Performance requirements and constraints on design costs and power consumption still require that significant parts of Digital Signal Processing (DSP) systems are implemented using dedicated hardware blocks. To deal with the SoC challenges, designers use system-level description, co-design techniques and re-use IP cores. Unfortunately, the main problem when re-using pre-designed hardware accelerator arises from their integration and more particularly from the communication features. System integrator can use standard interface such as VCI proposed by VSIA. However in DSP applications, in addition to the protocol aspects, SoC designer has also to synchronize the components and to buffer data to ensure the system behavior and to meet timing constraints. IPs are indeed delivered at the RTL level that is, following the VSIA taxonomy, the highest abstraction level for synthesizable IP models (soft cores). However, such a description may be parameterizable, it relies on a fixed architectural model with very restricted customization capabilities. This lack of flexibility of RTL IPs is especially true for the communication unit whose sequence orders and timing requirements are set. IP are hence connected to the SoC bus through specific interfaces (wrappers) that adapt the system communication features to the IP requirements. Unfortunately, this adaptation increases the final SoC area and also decreases system performance. In some cases, the I/O timing requirements cannot be respected due to the wrapper overhead and can cause the SoC design to fail.

We propose an approach based on high-level synthesis techniques under constraints to design the behavioral IP specification. Hence, we aim to optimally synthesize the IP by taking into account, in its specification, the system integration constraints: application rate, technology, bus format, I/O timing properties specified by timing frames of transfers. We consider variable but bounded timing constraints in real time DSP applications to handle non-determinism on transfer times that can originate from (1) computation performed by other system components and/or (2) transfer delay and protocol overhead.

Our methodology proposes to raise the abstraction level of IP synthetizable models by introducing the concept of behavioral IP, described as an algorithm and specified using HDL language. Starting from the system description and its architecture model, the integrator, for each bus or port that connects the IP to SoC components, refines and specifies I/O protocols, data sequence orders and timing information of transfers. The virtual component specification is modeled by a Signal Flow Graph SFG. We first generate an intermediate Algorithmic Constraint Graph ACG from the operator latencies and the data dependencies expressed in the SFG. Having described the IP behavior and the IP design constraints in a formal model, we then analyze the feasibility between the rate, data dependencies of the algorithm and technological constraints. This analysis checks the ACG for positive cycles to ensure that the constraint graph is feasible without considering input arrival dates. In order to support the features of communication architectures specific to DSP application, we define a formal model named IOCG (IO Constraint Graph) that supports expressing of integration constraints for each bus (id. port) that connects the IP to the SoC components. It allows (1) to specify transfer related timing

constraints such as ordered transactions, relative timing specification, min-max delay, (2) to include architecture features and (3) to express non determinism in the data transfer time. Finally we generate a Global Constraint Graph (GCG) by merging the ACG with the IOCG graph. Merging is done by mapping the vertices and associated constraints of IOCG onto input and output vertices set of ACG. A minimum timing constraint on output vertices (earliest date for data transfer) of the IOCG are transformed into the GCG in maximum timing constraints (latest date for data computation/production). With the formal description of the set of constraints, we analyze the consistency of the IP design constraints according to the algorithm ones. Consistency analysis refers to the dynamic behavior of the GCG graph. The entry point of the IP design task is the GCG that is used to synthesize the processing, memory, control and communication units that compose the IP architecture.

We applied our method to DSP and telecommunication applications. A first experiment was carried out on a FFT example. With the experimental conditions, the optimization of operators is among 20% and that of the registers of 7%, compared to a HLS approach ignoring I/O constraints. A second experiment uses a Discrete Cosine Transform DCT to compare the results, obtained by applying the approach of integration we proposed, with the results of the wrapper-based methods. For the considered example, the communication register gain varies from -2% to 88% for a constant I/O rate. The last experiment carried out in industrial partnership, has shown the applicability of our methodology on a complex behavioral IP (Maximum A posteriori MAP) in an application of real time Turbo decoding.

Tables des matières

Tables des figures

Tables des tableaux

Introduction

Le marché des composants électroniques est fortement orienté vers les produits grand public qui ciblent de plus en plus des applications couplant les télécommunications et le multimédia : téléphones mobiles, PDA (Personal Digital Assistants), systèmes GPS (Global Positioning System), jeux et lecteurs vidéo... L'*"International Technology Roadmap for Semiconductors"* [ITR01] prévoit que les processeurs contiendront 97 millions de transistors en 2007 et 1,5 milliards en 2013. L'évolution de la technologie sub-micronique permettra à terme de combiner les fonctions d'un téléphone, d'un navigateur Internet, d'un appareil photo numérique, d'un écran couleur, de lecteurs multimédias et d'un PDA au sein d'un unique objet portable. Les systèmes électroniques mobiles de demain, dit de troisième ou quatrième génération, ont un marché potentiel dont le revenu mondial est estimé à 320 milliards de dollars US à l'horizon 2010.

Dans ce contexte, les systèmes complets seront intégrés sur une même puce nommée Système-sur-Silicium (SoC) contenant une partie logicielle et une partie matérielle : la réalisation d'une application entièrement logicielle exécutée sur une architecture multiprocesseurs dédiant au matériel des opérations périphériques ne sera plus envisageable. En effet, bien qu'actuellement cette solution facilite la conception de circuits numériques complexes elle ne permettra pas, pour les futurs systèmes embarqués, le respect de contraintes drastiques en terme de consommation électrique et de performances temporelles. Ainsi, une partie non négligeable des traitements intensifs sera effectuée par des accélérateurs matériels (coprocesseurs). A cette difficulté de conception s'ajoutera la contrainte de développement rapide des produits grand public et l'hétérogénéité des circuits (analogique/numérique, multiples domaines d'applications, contrôle/données, matériel/logiciel...) qui demande un ensemble de compétences difficiles à réunir au sein d'une même équipe de concepteurs.

Pour faire face à cette future conjoncture, les équipes de recherche proposent des nouvelles méthodologies de conception qui reposent sur (1) la réutilisation de blocs logiciels ou matériels préconçus *(Composant Virtuel VC, Intellectual Property IP)*, (2) sur l'élévation du niveau de description des applications (System-Level Design) et (3) sur l'orthogonalisation de différents aspects (fonctionnalité/architecture, calculs/communication, besoins/contraintes/choix d'implantation) (voir Figure 1-1). Ces méthodologies permettent, dans une certaine mesure, de réduire l'écart (*design gap*) qui existe entre l'évolution linéaire la productivité des concepteurs et la croissance exponentielle de la complexité des applications et des capacités d'intégration offertes par la technologie. Malheureusement de part l'orientation plutôt logicielle des solutions architecturales actuelles, les composants virtuels orientés Traitement du Signal et de l'Image (TDSI) et Télécommunications, contrairement aux processeurs (CPU, DSP), sont faiblement

représentés dans les bibliothèques d'échange de composants. A cela s'ajoute la difficulté de réutilisation liée au niveau de description RTL, auquel sont fournis ces IPs, qui ne permet pas une intégration aisée et optimisée. En effet, le faible niveau d'abstraction des composants virtuels demande à l'intégrateur de gérer la synchronisation, la temporisation des données et la conversion de protocole en utilisant un adaptateur (wrapper, module d'interface). Les méthodologies d'intégration à base d'adaptateurs additionnels sont souvent inadaptées aux contraintes et à la spécificité des algorithmes utilisés dans le TDSI. Elles peuvent de ce fait aboutir à la violation des contraintes en terme de surface, consommation et performances du système : le fort volume de données traitées dans les applications orientées TDSI et Télécommunications est un point critique des futurs systèmes embarqués qui nécessite une approche dédiée de conception et d'intégration des composants virtuels. A cela s'ajoute pour la réutilisation d'IP flexibles et génériques, un besoin d'adaptations des Entrées/Sorties, tant sur le parallélisme que sur le format des données.

Nous proposons dans ce mémoire une approche de réutilisation des IPs dans les applications orientées traitement du signal, de l'image et des télécommunications. Pour cela nous basons notre approche sur la notion de composants virtuels de niveau algorithmique définis dans le cadre des projets *RNRT MILPAT (Méthodologie et Développement pour les Intellectual Properties pour Applications Telecom)* [RNR02A]. Le flot de conception proposé s'inscrit dans une démarche Adéquation Algorithme Architecture du projet *RNRT ALITPA (Définition et Application d'une méthodologie de développement pour les (IP) intellectual property de niveau comportemental dans les applications de télécommunication)* [RNR03B] et est basé sur l'utilisation de techniques de synthèse haut niveau sous contraintes d'intégration. Les unités fonctionnelles constituant l'architecture cible du composant sont (re)conçues en fonction des caractéristiques de l'architecture de communication du système et de la spécificité de l'application.

Plan du mémoire

Le premier chapitre est consacré à la problématique générale de conception de SoC. Nous détaillerons les différentes étapes du flot de conception et présenterons les travaux relatifs. Nous passerons ensuite en revue les techniques de synthèse et de raffinement des communications allant de l'exploration architecturale à l'implémentation. Nous mettrons en avant les problèmes spécifiques de la réutilisation de composants virtuels dans des applications TDSI temps réel.

Dans le deuxième chapitre, nous dresserons dans une première partie un état de l'art sur les techniques d'intégration des composants virtuels de niveau RTL utilisant des adaptateurs. La deuxième partie sera quant à elle consacrée à la synthèse comportementale sous contraintes.

Le troisième chapitre détaillera le flot de conception que nous proposons. La modélisation des contraintes d'intégration, des contraintes algorithmiques aisni que les phases d'analyse seront définies.

Nous proposons dans une première partie du quatrième chapitre, une technique d'optimisation de synthèse de l'unité de traitement ainsi qu'une modélisation des contraintes de transferts

générées par la synthèse de l'unité de traitement utilisée pour la synthèse de l'unité de communication. Dans une deuxième partie, nous présentons l'environnement logiciel GAUT dans lequel ont été intégrés ces travaux. Nous rappellerons la philosophie initiale de l'outil puis détaillerons les modifications apportées permettant la prise en compte du nouvel ensemble de contraintes.

Dans le cinquième chapitre nous présenterons les résultats de synthèse pour des applications TDSI et télécommunications et mettrons en avant l'intérêt de l'approche proposée pour l'intégration des composants virtuels. Nous illustrons la capacité que présente notre méthode et outil à traiter des problèmes industriels de grande complexité

Enfin, dans le dernier chapitre nous conclurons ce manuscrit en présentant les apports de la méthode et en dégageant les perspectives.

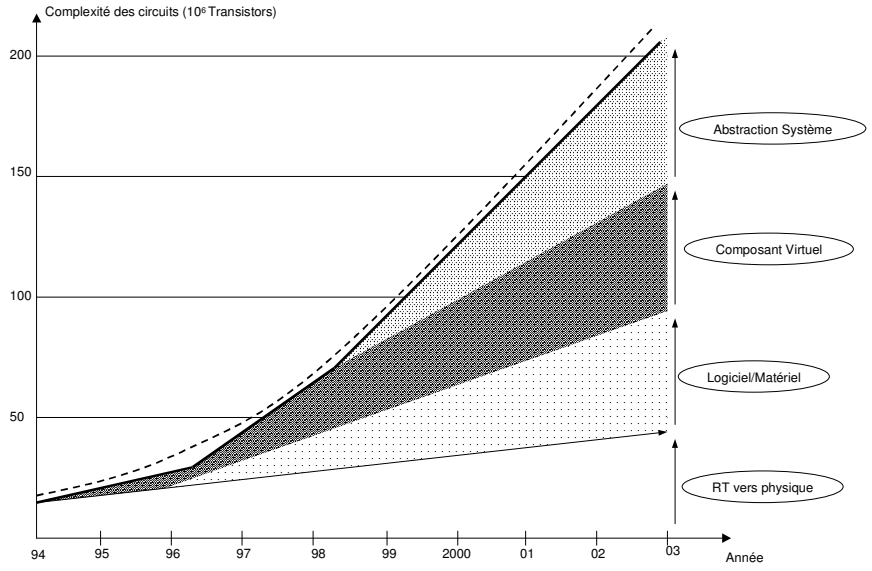


Figure 1-1 : Réduction du *"Design Gap"* par les nouvelles méthodologies

Chapitre 1

Problématique

Dans la première partie de ce chapitre introductif, les travaux relatifs aux approches de conception des SoC et aux niveaux d'abstraction seront passés en revue. La deuxième partie présentera le concept de composant virtuel sur lequel sont basées les méthodologies actuelles. La dernière partie relate les problèmes liés à l'intégration de ce type de composant dans des architectures orientés TDSI et Télécommunications.

1. Flot de conception d'un SoC

Depuis quelques années, nous assistons à la définition de méthodologies de conception permettant de réduire le fossé qui existe entre la productivité des concepteurs et la capacité d'intégration offerte par les technologies sub-microniques. Ainsi, les flots de conception autorisant la co-conception, la réutilisation de composant virtuel et la conception au niveau système réduisent de façon considérable les coûts de conception des systèmes complexes (voir Figure 1-1, source Garter Dataquest [GAR03]).

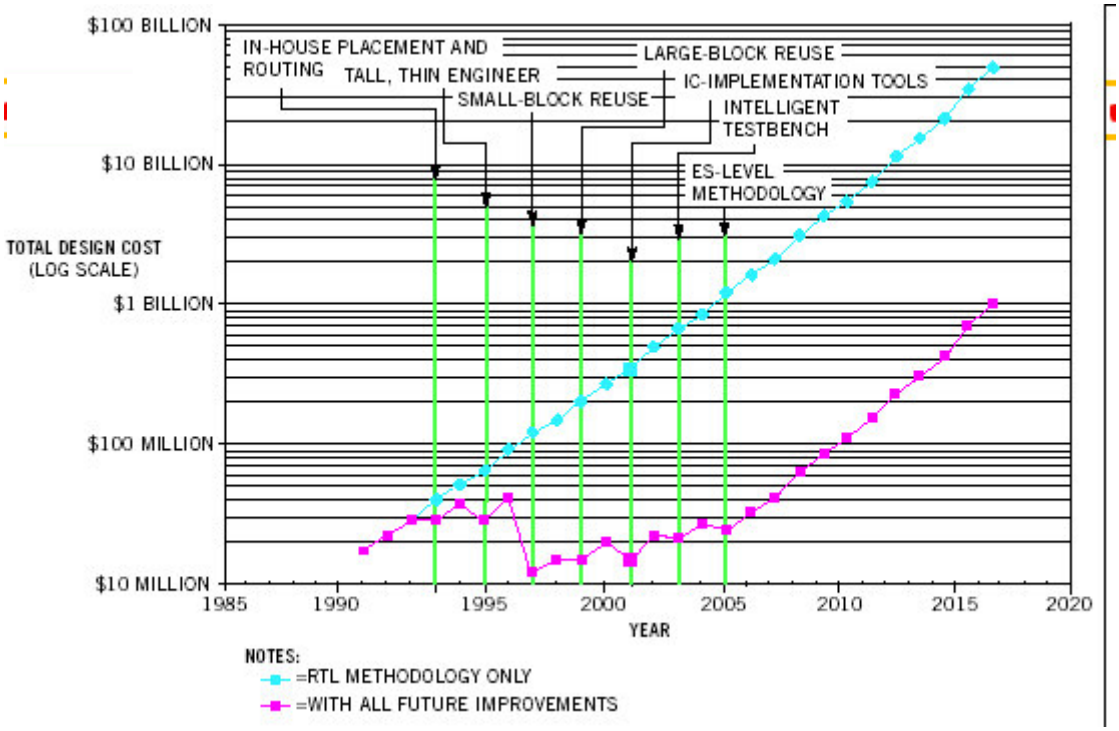


Figure 1-1 : Réduction du coût de production d'un SoC par les nouvelles méthodologies

Ces méthodes reposent sur l'orthogonalisation de différents aspects qui permet une plus large exploration de l'espace de solutions d'implémentation [KEU00], [BAL98], [ROW97]. L'orthogonalisation concerne : (1) la fonctionnalité et l'architecture, (2) les calculs et la communication et (3) les besoins/contraintes et les choix d'implémentation. Cette dissociation requiert la définition de modèles, de niveaux d'abstraction, de méthodes permettant le passage d'un niveau de raffinement à un autre et de flots de conception adaptés. Un flot de conception repose donc sur un ensemble d'étapes permettant de fournir, à partir d'une spécification initiale, l'implémentation d'une application. La complexité de ce problème de transformation nécéssite un processus de raffinement incrémental et la définition de modèles intermédiaires d'abstraction adéquats. Un flot de conception est constitué d'une suite d'alternances de phases d'exploration à un niveau d'abstraction et de raffinements entre niveaux d'abstraction. Cette gestion par palier permet de réduire la quantité d'information que le concepteur doit gérer à un niveau donné évitant ainsi les interactions et effets de bords avec les niveaux inférieurs.

Les nombreuses solutions apportées par la recherche académique ou industrielle peuvent être regroupées en trois familles [CYR01] [CAI03] : La synthèse à partir d'une spécification système fonctionnelle (*Synthesis from specification* ou *system synthesis*), la conception s'appuyant sur l'utilisation de plate-forme (*Platform-Based Design*) et la conception basée sur la réutilisation de composant (*Component-Based Design* ou *IP assembly*).

La première famille méthodologique, qui synthétise un système à partir de sa spécification haut niveau, requiert l'utilisation d'outils de co-design, de compilation et de synthèse architecturale ainsi que des langages de description adaptés. L'architecture du système est générée à partir de la description fonctionnelle initiale de l'application. Elle est ensuite raffinée, par ajout de détails d'implémentation, jusqu'à l'obtention de la description de son modèle physique. Dans ce type d'approche, l'accent est mis sur l'un des points suivants : interfaçage, partitionnement, raffinement, synthèse et simulation. COSMOS [BEN94], CHINOOK [CHO95] et SPECSYN [GAJ00] utilisent ce type de méthodologie dont le principal attrait, mais aussi le premier défaut, réside dans le haut degré d'automatisation qui requiert l'utilisation d'outils très flexibles difficiles à concevoir. Ce type de méthodologie a été étendu pour supporter la réutilisation de composants virtuels ([CHO99][GAJ98]).

La deuxième famille d'approches [CHA99], plutôt que de générer l'architecture du système à partir de sa description initiale, relie son comportement à un modèle d'architecture plus ou moins paramétrable (*Parameterised System Design*). Les outils tels que VCC de Cadence [CAD03], N2C de Coware [COW03], YAPI [DeK00] ou MCSE/Cofluent [MSC03][COF03] fournissent des solutions de ce type. Cette méthodologie est plus directe et actuellement moins automatisée que la précédente. La flexibilité qu'elle offre est importante mais oblige les concepteurs à développer des plates-formes spécifiques à des domaines d'applications, ce qui est coûteux.

La dernière classe de solutions repose sur la réutilisation et l'assemblage de composants préconçus. Elle est souvent utilisée afin d'accélérer la conception des plates-formes et est vue comme une solution intermédiaire entre les deux familles méthodologiques présentées précédemment. Le processus consiste en l'utilisation de composants virtuels de niveau RTL intégrés dans l'architecture au travers de *wrapper* [CES02] dont l'intégration est validée par un ensemble de simulations *(mix-and-match, capture-and-simulate)*.

Ce chapitre décrit de flot de conception *générique* d'un SoC réutilisant des composants virtuels. La Figure 1-2 présente les différentes étapes, allant de la spécification jusqu'à la synthèse, pour lesquelles nous avons volontairement omis les phases de vérification et les rebouclages afin d'en faciliter la lecture. Ces principales étapes seront présentées de façon détaillée dans les sections suivantes. Il est cependant important de noter qu'en fonction de la méthodologie les phases de conception peuvent ne pas être dissociées. Par exemple, dans la Figure 1-2 l'étape de synthèse et de raffinement des communications succède la phase d'exploration architecturale. Or dans certaines méthodologies ces deux étapes sont réalisées conjointement.

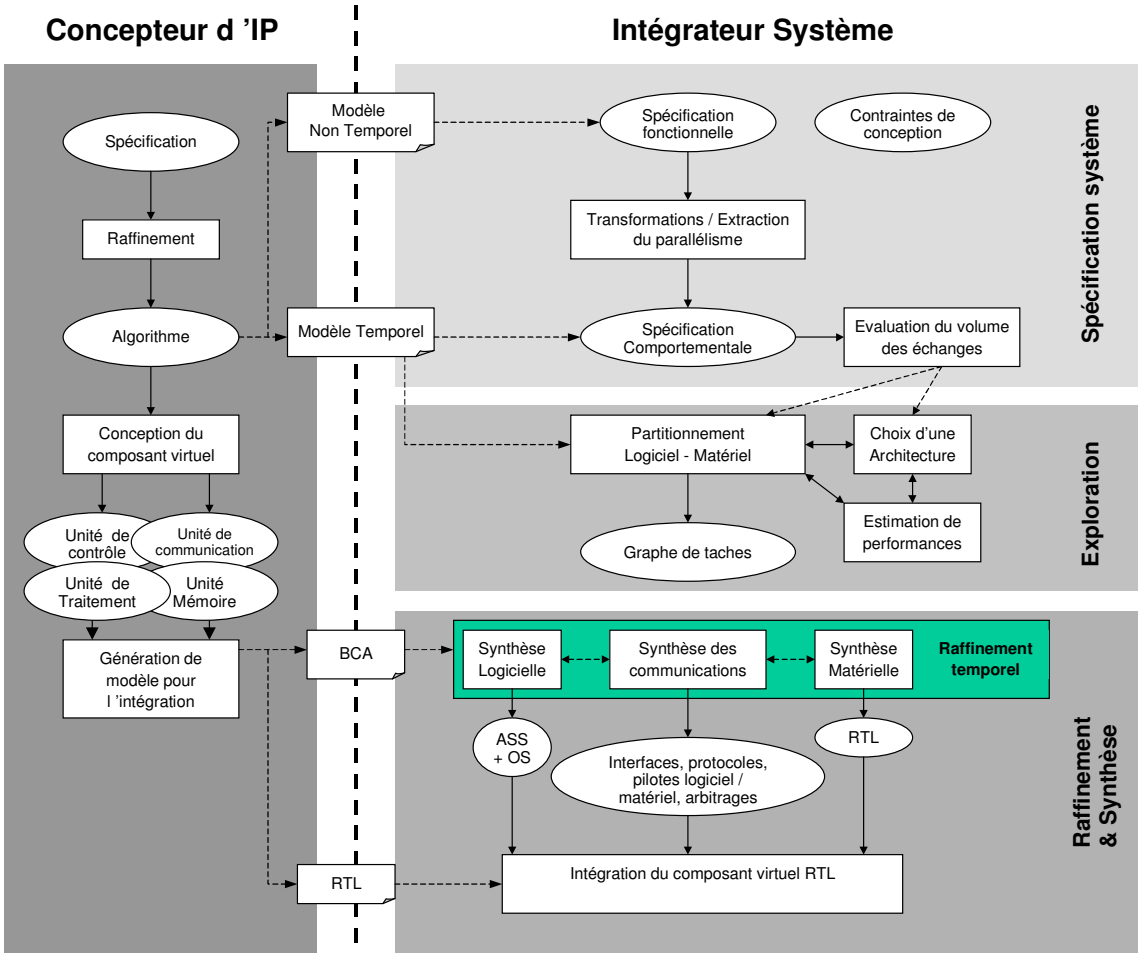


Figure 1-2 : Flot de conception d'un SoC pour l'intégration des Ips

1.1. Les spécifications du système

1.1.1. Niveaux de description

Le flot de conception débute par une spécification fonctionnelle du système, devant être exécutable. Cette spécification, qui décrit le comportement de l'application, est accompagnée d'un ensemble de contraintes de conception telles que le coût en surface, la consommation électrique, le débit, les interfaces externes etc. Ces exigences non fonctionnelles sont ensuite utilisées dans les phases avales de la conception (choix des IP, synthèse, définition de l'architecture...). L'écriture de la spécification fonctionnelle du système est une tâche fondamentale et non triviale puisqu'elle doit être non ambiguë, consistante et complète. En effet, c'est à partir de cette description initiale que sont générés les bancs de tests et les stimuli utilisés dans toutes les étapes ultérieures de la conception. La *spécification fonctionnelle* est souvent réalisée à l'aide d'un langage de programmation séquentiel (langage C par exemple) et ne fait aucune référence à la structure du matériel ou du logiciel qui permettra l'implémentation finale du système. Elle modélise les traitements à l'aide d'équations mathématiques sous la forme d'expressions purement algébriques. L'échange d'information entre les fonctions repose sur

l'utilisation de variables partagées (Figure 1-3). Un premier échange entre le concepteur de composants virtuels et l'intégrateur système est alors réalisable au travers d'une librairie de modèles en précision infinie.

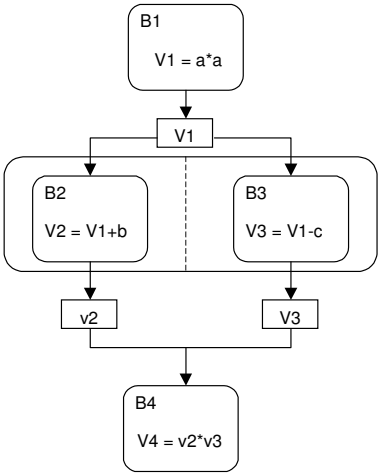


Figure 1-3 : Le modèle fonctionnel

Cette première spécification est ensuite transformée en un ensemble de blocs fonctionnels communiquants qui exhibent le parallélisme intrinsèque de l'application : la *spécification comportementale* aussi nommée *fonctionnelle parallèle* ou encore *prototype virtuel*. C'est lors de cette phase que les modèles mathématiques sont raffinés en modèles algorithmiques. Dans les systèmes orientés Traitement du Signal et de l'Image (TDSI), le choix d'un modèle d'algorithme est crucial. En effet, c'est à partir de ce dernier que vont être déterminés par exemple la précision des calculs (raffinement virgule fixe, virgule flottante) et le format des données à l'aide d'outils tel que MATLAB [MAT03] ou de méthodes telle que [MAR01].

L'utilisation de la spécification comportementale permet la décomposition d'un système complexe en sous-systèmes qui autorisent une plus grande maîtrise de la conception. Aucune référence à l'architecture ou au mode d'implantation n'est faite à ce stade : les fonctions réalisées par les sous-systèmes sont instantanées (*Un-Timed Functional Model UTF*). Le système est décrit par un ensemble de processus (ou blocs fonctionnels) concurrents, qui échangent des données au travers de canaux virtuels (voir Figure 1-4). Ces canaux réalisent la communication par passages de messages et permettent la synchronisation inter processus sans aucune référence au protocole ou au bus utilisé. Les accès aux canaux sont réalisés par l'intermédiaire d'interfaces rattachées aux ports. Des outils tels que COSSAP [SYN97], SPW [CAD03A] ou YAPI sont couramment utilisés pour réaliser ce type de description dans les domaines du TDSI ou des Télécommunications.

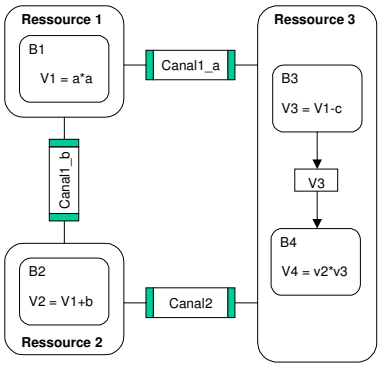


Figure 1-4 : Le modèle comportemental

1.1.2. Langages de spécification

Beaucoup de langages et méthodes associées, ont été développés pour répondre aux besoins de la spécification système. Ainsi, les spécifications peuvent être réalisées à l'aide d'un langage orienté objet pour l'analyse et la conception de système (OOA/OOD) tel que UML (Unified Modeling Language) [BOO99]. Toutefois, ce langage ne permet pas, à la différence de StateCharts [HAR87] ou SDL-2000 [ITU99], la génération d'un modèle exécutable. Des langages spécialisés dans la spécification des applications réactives ont été dérivés des StateCharts comme les langages synchrones Signal [LEG91], Lustre [HAL91] ou Esterel [BER00]. Ce dernier, qui ne modélise que la partie contrôle de l'application, a été étendu afin pour modéliser les traitements en utilisant le langage C : Esterel-C language ECL [LAV99] qui est utilisé dans l'environnement logiciel VCC. Les langages de programmation traditionnelle C et C++ ont aussi été complétés pour décrire des systèmes numériques et ont donné naissance à des langages comme SystemC [OSC01], SpecC [GAJ00] ou HardwareC [KU90]. Superlog [SUT99], VHDL+ [ICL99] ont été dérivés des langages de description de matériel VHDL et Verilog pour supporter la description d'architecture au niveau système. ROSETTA [ALE00], qui est en cours de définition, travaille à un niveau d'abstraction plus élevé et permet de spécifier l'ensemble des besoins et des contraintes d'une application.

1.1.3. Modèles sémantiques

La spécification fonctionnelle parallèle repose sur un modèle d'exécution (Model of Computation ou MOC) qui décrit le comportement des tâches. Les modèles de comportement se différencient par leurs granularités (processus, objet), leurs mécanismes de communication inter tâches (passage de message, appel de procédure distante RPC, variables partagées), leurs mécanismes de synchronisation (FIFO, poignée de main), leurs orientations (vérification formelle, synthèse, simulation), leurs domaines d'application, mais aussi leurs supports de la hiérarchie et leurs notions de temps. Toutes ces propriétés sont exprimées de façon implicite ou explicite et reflètent les caractéristiques du système modélisé. Dans cette section nous présentons brièvement et de façon non exhaustive les modèles concernant la spécification fonctionnelle parallèle.

Le modèle *CSP (Communicating Sequential Process)* [HOA85] modélise un réseau de processus séquentiels qui communiquent par passages de messages synchrones (rendez-vous).

Les *réseaux de processus (Kahn Process Network [KAH74])* décrivent le système comme un réseau de processus traitant des flux infinis de données. L'information transite par des canaux unidirectionnels à lecture bloquante et écriture non bloquante. Ce mécanisme de synchronisation est représenté par des FIFO de taille infinie qui permettent une communication asynchrone et imposent une chronologie identique aux données produites et consommées par les composants du système. Ce modèle est particulièrement adapté aux applications orientées Traitement du Signal et de l'Image (TDSI).

Les modèles *flots de données* peuvent être vus comme un cas particulier des *KPN* ayant des FIFOs dont les tailles sont bornées. Ainsi, dans le modèle *SDF (Synchronous Data Flow)* [LEE87] les processus nommés *acteurs* consomment et produisent un nombre fini de données. Les données sont divisées en *jetons*, qui sont traités comme des unités indivisibles. Un canal représente une séquence de jetons conceptuellement stockés dans une *FIFO unidirectionnelle*. L'exécution (*tirage* ou *firing)* d'un bloc est un calcul indivisible qui consomme et produit un nombre fixé de jetons. Les FIFOs sont remises dans leurs états initiaux après un certain nombre de tirages (une *itération*) calculé par un ensemble d'*équations de balance*. Une équation de balance repose sur la condition suivante : lors d'une itération le nombre total de jetons produits par le bloc source est égal au nombre total de jetons consommés par le bloc de destination. Trouver une solution à ce système d'équations revient à assurer que le graphe *SDF* peut être exécuté avec une quantité bornée de mémoire. Les *inter-blocages* sont détectés lors de l'ordonnancement des processus du système. Ce modèle est particulièrement adapté aux applications de traitements du signal ayant un comportement statique. Les *DFG, CFG* et *CDGF* appartiennent à la famille des SDF. Les *CSDF (Cyclo-Statique Dataflow)* [BIL96] sont des SDF dont les règles de tirages varient de façon cyclique mais dont l'évolution est définie de façon statique. Les *DDF (Dynamic Data Flow)* sont un super ensemble des SDF utilisés pour la modélisation de logiciels: le nombre de jetons consommés et produits peut varier d'un tirage à un autre. Evidement, les inter-blocages ainsi que les quantités mémoire nécessaires deviennent non prédictibles dans ce modèle. L'ordonnancement réalisé à la compilation pour les *SDF* devient impossible: il est donc réalisé dynamiquement lors de l'exécution.

Les *machines à états finis (FSM)* font partie des modèles de calcul les plus utilisés pour la description de composants numériques. Elles sont composées de deux éléments : des nœuds et des arcs. Les nœuds représentent les états du système et les arcs les transitions. Ces derniers sont étiquetés par des conditions de transition appelées *gardes* et par des *actions* qui représentent la génération des sorties. Ce modèle est assez proche de l'architecture matérielle qu'il modélise, ce qui le rend assez peu utilisable pour des modélisations faisant abstraction de l'implémentation. De plus, le nombre d'états croit de façon exponentielle proportionnellement à la taille du système à modéliser : leur nature synchrone n'autorise pas la description aisée d'un système dans lequel les composants évoluent en parallèle *(GALS Globalement Asynchrone Localement Synchrone)* puisque à tout instant le système est représenté par un unique état. Afin de palier à ce problème, D. Harel a introduit dans [HAR87] le concept de *HCFSM (Hierarchical and concurrent finite-state machine).* Sa représentation graphique est un diagramme d'états (FSM) agrémenté de trois principes fondamentaux qui permettent une réduction exponentielle

de la complexité: la hiérarchie (profondeur, *depth*), la concurrence (orthogonalité) et le non-déterminisme. Ce dernier permet de décrire des systèmes dont le comportement est partiellement connu au moment de la modélisation et/ou des systèmes dont la spécification des réactions à certains événements n'est pas nécessaire. Le langage graphique *Statecharts* repose sur ce modèle *HCFSM* et utilise un mécanisme de communication par diffusion d'événements.

Les *FSMD (Finite-State Machine with Datapath)* introduites par Gajski [GAJ98] sont des machines d'état fini étendues avec un chemin de données. Pour rendre les FSMs utilisables pour la description d'architectures plus complexes, un ensemble de variables (entières ou flottantes) stockées dans des registres, mémoires ou files de registres a été introduit. Ce modèle permet la modélisation de systèmes dominés par le contrôle et/ou les calculs, mais comme les FSMs, elles ne supportent ni la hiérarchie ni la concurrence. De ce fait, elles ne sont utilisables que pour des systèmes de faible importance. Pour palier ce problème, les auteurs définissent un modèle de plus haut niveau nommé *SFSMD (pour Super Finite State Machine with Datapath)* [LEH99] dont un état peut représenter un calcul complexe.

Les *FSMC (Finite State Machine with Coprocessor)* [JER97] sont dérivées des FSMD. Une FMSC modélise une architecture hiérarchique faite d'un contrôleur principal et d'un ensemble de chemins de données qui peut lui-même inclure des FSMC.

Les *CFSM (Co-design Finite-State Machine)* [CHI93] utilisent des *machines à états finis* auxquelles ont été ajoutés la manipulation de données et un mode de communication asynchrone réalisé par des *"buffers"* une place. Les événements émis par une CFSM (ou par l'environnement où le système fonctionne) peuvent être détectés par une ou plusieurs CFSMs *(Broadcast communication model)*. Les événements implantent directement un protocole synchrone entre les partenaires qui communiquent. La lecture est bloquante et le récepteur attend que l'émetteur ait émis l'événement évitant ainsi l'acquisition d'informations corrompues. L'émetteur n'est pas bloqué après l'émission d'un événement. Le tampon implicite d'une place, qui est placé entre l'émetteur et tous les récepteurs, mémorise l'événement jusqu'à ce qu'il soit détecté ou écrasé. Les CFSMs utilisent un model de temps discret où chaque élément de calcul (CFSM) réalise une tâche en un temps non nul et non borné. Le trop bas niveau de description des CFSMs et leur modèle de communication implicite a motivé la création des *ACFSM (Abstract Co-design Finite State Machine)* et *ECFSM (Extended Co-design Finite State Machine)* [SGR00].

Un réseau de ACFSMs est un réseau de FSMs communiquant par événements (ordre partiel) à travers des signaux sans perte modélisés par des FIFO infinies. Ces FIFO supportent les temps d'exécution inconnus et découplent la communication de la spécification fonctionnelle du système rendant ainsi leur raffinement indépendant. Contrairement aux KPN ce modèle autorise les lectures non bloquantes : une transition peut être conditionnée par l'absence d'un événement sur un signal. Les ACFSM sont ensuite raffinées en ECFSMs qui sont des réseaux de FSM communiquant à travers des FIFOs bornées. Les CFSMs sont donc un cas particulier des ECFSMs communiquant par des FIFOs à une place. Les ECFSMs permettent la spécification de communication à perte (*FIFO bornée*) et sans perte (*FIFO bornée + protocole + fonction de contrôle d'erreurs CCR, ...*) [SAN00].

1.2. Exploration architecturale

Nous présentons dans ce paragraphe les principales étapes de l'exploration architecturale : le choix de l'architecture, le partitionnement matériel/logiciel et l'estimation de performances. Ces opérations sont interdépendantes et l'ordre dans lequel elles sont réalisées ne dépend que de la méthodologie et du flot de conception utilisé. L'exploration architecturale assigne les blocs fonctionnels de la spécification comportementale, aux ressources matérielles sélectionnées et ordonnance l'exécution des tâches. L'assignation de tâches à des ressources matérielle permet d'associer des temps d'exécution aux blocs composant la description comportementale. On parle dans ce cas de *description comportementale temporelle (Timed Functional TF)*. En fonction de la méthodologie de conception utilisée, l'information temporelle provient d'une estimation ou d'un résultat de synthèse.

Durant cette étape, les communications entre blocs fonctionnels peuvent être réalisées au travers de canaux plus ou moins abstraits (niveau *Bus Cycle Accurate BCA)*. La communication peut ainsi être instantanée et dans ce cas aucun délai n'est associé aux canaux. Elle peut aussi, en fonction de son niveau d'abstraction, contenir des informations telles que la latence, le débit, la largeur des bus, les modes de transferts, les priorités, le type maître/esclave, le mode d'arbitrage, et/ou les protocoles.

1.2.1. Partitionnement Logiciel / Matériel et choix d'une architecture

Deux types d'approches de partitionnement se proposent au concepteur de système. La première s'appuie sur architecture cible déjà établie, ce qui contraint fortement le partitionnement. La deuxième déduit l'architecture de l'analyse des besoins. Cette approche nécessite d'une part, de choisir le type des composants (CPU, DSP, ASIC), leur modèle (par exemple DSP C6X ou C52), leur nombre et d'autre part la topologie de l'architecture des communications (bus, mémoires, modes de transferts...). Un compromis entre ces deux approches réside dans l'utilisation d'un modèle d'architecture générique.

La littérature fait référence à de nombreux travaux concernant les approches de partitionnement de système qui ne seront pas détaillées ici. Ces approches peuvent être entièrement automatisées, semi-automatiques ou entièrement manuelles ce qui, en fonction de la complexité de l'application, peut être fastidieux. L'étape de partitionnement s'appuie sur les résultats d'estimations qui quantifient la qualité de la solution trouvée en fonction de certains critères comme la vitesse, la surface, le nombre de composants ou la communication.

1.2.2. Estimation de performances prenant en compte la communication

Cette section ne s'intéresse qu'aux méthodes prenant en compte la communication dans l'estimation de performances d'une architecture. Constatant la prépondérance des temps de communication dans les architectures de type SoC, certains travaux proposent des **techniques**

d'analyse de performances, au niveau système, qui intègrent, durant l'étape de partitionnement, certains aspects de la communication.

Ainsi dans LYCOS [KNU99], la méthode utilisée repose sur un modèle de temps de communication qui prend en compte : le nombre de données transférées, la largeur des données en fonctions de la taille de bus, le temps d'accès à une donnée, le type de transfert (mot, octet…), la taille des "burst" et la direction des données (écriture ou lecture). Ce modèle ne considère cependant pas les attentes de bus et les comportements non déterministes. Dans [DEY97], la technique PERC analyse la communication inter processus en intégrant les délais ajoutés par la synchronisation et estime les pires cas de performances des systèmes décris par un ensemble de processus communiquants. Les échanges de données sont modélisés par un graphe qui capture la synchronisation et la communication entre les processus. Dans ces travaux, l'ordonnancement des tâches est supposé statique, l'architecture est monoprocesseur et les délais introduits par les conflits d'accès aux bus ne sont pas supportés. A la différence des deux approches précédentes la technique proposée dans COSYMA [YEN95] supporte, en considérant le pire cas, les attentes liées aux contentions d'accès à un bus. L'architecture visée contient plusieurs processeurs qui échangent des données au travers de mémoires partagées.

Dans [LAH01] les auteurs considèrent les architectures de communication personnalisées dans la phase d'analyse. La technique proposée est un compromis entre les méthodes dynamiques de simulation d'un système complet qui sont lentes et les méthodes d'analyse statique qui sont rapides mais peu précises. La première étape de cette méthodologie consiste en une cosimulation du système dans lequel les communications sont décrites de façon abstraite (événement, transfert de données). Le temps nécessaire pour consommer et produire un événement de communication dépend uniquement de la taille du transfert associé à cet événement. Les résultats de simulation permettent la construction d'un ensemble de traces qui contiennent les informations sur les calculs et les communications des composants. Ces traces d'exécution du système sont imprécises temporellement. La deuxième étape (en 3 phases) analyse les effets de l'architecture sur les performances système. (1) A partir de la trace produite par la simulation, un Graphe de communication Acyclique (CAG) est construit : Ce CAG reflète les calculs, les communications ainsi que les synchronisations. (2) L'architecte système spécifie ensuite l'architecture de communication : (topologie, liaisons communications abstraites / composants de communication, des protocoles utilisés). (3) (*analyse*) Ces CAG sont ensuite manipulés pour intégrer les contraintes et les paramètres de l'architecture. Ceci permet la génération de traces (précises dans le temps) des performances du système. L'outil fournit en sortie :

- Un CAG incorporant les latences introduites par le raffinement des communications
- Une estimation des performances du système
- Le chemin critique du système sous la forme d'une suite de calculs et de communications
- Des statistiques sur l'usage du bus, les conflits, l'implication des composants dans le chemin critique

Cette technique permet de modéliser les effets dynamiques des architectures de communication (attente due aux contentions), de prendre en compte les interdépendances entre les calculs, les synchronisations et les échanges de données réalisés par les différents composants, tout en estimant les performances du système.

1.3. Synthèse du système

1.3.1. Synthèse Logiciel / Matériel

La synthèse des parties logicielles du système consiste à raffiner la description haut niveau d'une tâche en une description bas niveau, exécutable par le processeur cible. La spécification est initialement décrite par un ensemble de fonctions et de procédures dans un langage tel que le C ou le C++. Le code binaire exécutable peut être généré automatiquement à l'aide du compilateur associé à la cible. Malheureusement, les résultats rapidement fournis par cette technique sont souvent moins optimaux en terme de vitesse, de consommation ou d'optimisation mémoire, que le code assembleur obtenu manuellement. L'utilisation d'un système d'exploitation est nécessaire lorsque plusieurs tâches, ne pouvant être fusionnées, sont assignées à une même ressource matérielle [COR00].

Tout comme la synthèse logicielle, la synthèse matérielle transforme un modèle décrit à un haut niveau d'abstraction en une description bas niveau. Une première étape de raffinement peut être réalisée à l'aide d'outils de synthèse d'architecture (également appelée synthèse comportementale ou synthèse haut niveau) qui à partir d'une description proche de l'algorithme produisent une description au niveau transfert de registres (Register Transfer Level RTL) [GAJ91]. Les outils universitaires tels que GAUT [MAR93], AMICAL [PAR93], HERCULES [DEM88], ou commerciaux comme Monet [ELL00] et Behavioral Compiler [KNA96] réalisent cette étape. Cette spécification RTL, est ensuite raffinée à l'aide d'outils de synthèse logique (ou synthèse RTL) pour obtenir un ensemble d'interconnexions de transistors (Netlist).

1.3.2. Synthèse des communications

Cette section relate brièvement les différentes approches et points traités par la *synthèse des communications*. Ce terme générique fait globalement référence à la conception des moyens d'interconnexion entre les composants d'un système et couvre un ensemble de tâches très différentes les unes des autres. En effet, la synthèse des communications dépend fortement du flot de conception adopté mais aussi du niveau d'abstraction utilisé et peut ainsi faire référence au choix de la topologie de communication, à la définition de modes d'échanges ou à la minimisation des coûts de transferts. Les travaux traitant cette étape seront regroupés, dans ce mémoire, en trois familles : la synthèse des interfaces, l'optimisation des communications et l'intégration de composants virtuels et ce en vue de la synthèse ou de la simulation. Ce dernier aspect sera développé plus en détails dans le chapitre suivant.

Les premiers travaux qui concernent la **synthèse des interfaces** entre deux composants ont un niveau d'abstraction très faible et sont basés sur des chronogrammes. Ces approches se concentrent sur la synthèse des interfaces en utilisant une description détaillée des protocoles. Ainsi, à partir d'une description bas niveau des contraintes sur les signaux, elle génère un

automate de contrôle. La description initiale peut être textuelle [NES86] ou graphique [BOR87], et peut faire référence à des communications de type *broadcast* comme dans Polis [CHI93]. [NAR95], [PAS98], et [LIN94] (méthode intégrée dans CoWare), réalisent l'interfaçage de composants utilisant des protocoles incompatibles et ont comme description initiale une spécification événementielle. [CHO95] et [MAD95] proposent une solution au problème spécifique de l'interconnexion de composants dans des architectures hétérogènes.

Dans une optique d'optimisation plus globale, d'autres approches utilisent un niveau de description plus abstrait afin d'**optimiser les communications**. Cette étape affecte à chaque canal abstrait un protocole de communication de type synchrone ou asynchrone. Les approches existantes se distinguent par leur placement dans le flot de conception, et plus particulièrement par rapport à la phase de partitionnement. Ainsi lors du découpage matériel/logiciel, [HEN94] utilise des critères tels que le coût logiciel (taille du code), coût matériel (surface) et la durée des transferts pour choisir le type et les modes de transfert dans une architecture composée d'un processeur, un coprocesseur et une mémoire partagée. [BEN96] ajoute à ces métriques l'occupation des bus dans des architectures multiprocesseurs. Contrairement à [YEN96], il ne considère qu'un unique mode de communication DMA qui n'est pas adapté aux transferts de faible volume de données. La deuxième famille d'approche se positionne après l'étape de partitionnement : le système est décrit à l'aide d'un graphe composé d'un ensemble de tâches (les nœuds) reliées par des canaux abstraits (les arcs). Dans ce contexte, la plupart des travaux tentent de minimiser les délais de synchronisation en réduisant les communications bloquantes. Dans [NAR94] les auteurs distribuent temporellement les communications liées à l'ordonnancement et fixées par le partitionnement. Ils maximisent ainsi l'utilisation du bus partagé sur lequel ils basent leur architecture système et ce pour une largeur minimale de bus. [GON96] étend les travaux précédants en ciblant quatre architectures basées sur l'utilisation de connexions point à point / bus partagé avec des mémoires de type locales / globales. La méthodologie proposée permet d'obtenir une partie contrôle gérant le protocole, de raffiner les primitives d'échanges de données (send, receive) et de définir un arbitre lors de l'utilisation d'un bus partagé. [FIL93] propose de modifier l'ordonnancement effectué durant le partitionnement afin de minimiser les communications. D'autres techniques de sélection de protocole de communication, similaires à celles précédemment décrites sont développées dans [FRE96] (construction exhaustive de l'ensemble des solutions), [GOG97], [CUE01], ou [BEN96], [DAV01] (utilisation des taux de communication crêtes pour une synthèse d'interfaces par allocation).

1.4. Conclusion

Comme nous l'avons montré, la conception d'un système sur silicium a été largement abordée dans le domaine académique et dans le domaine industriel. Ainsi, de nombreuses propositions, de méthodologies et d'outils, ont été formulées. Cependant, les projets couvrent rarement la totalité des phases impliquées dans le spectre de la conception et s'orientent souvent vers un type particulier d'application comme le montrent le Tableau 1-1 et le Tableau 1-2.

Projet	Origine	Problème visé	Application
Amical	TIMA	Synthèse	Contrôle
(IP)Chinook	U. Washington	Simulation, Synthèse	Contrôle
Cosmos	TIMA	Simulation, Synthèse	Contrôle
Cosyma	TU. Braunschweig	Exploration, Synthèse	Mixte
Cathedral	IMEC	Synthèse	Données
Disydent/Alliance	ASIM-Lip6	Exploration, Simulation, Synthèse	Mixte
Gaut	UBS	Synthèse	TDSI
IP Centric	UC Irvine	Méthodologie	Mixte
Lycos	TU. Denmark	Synthèse	Contrôle
MCSE/Cofluent	IRESTE/Cofluent	Méthodologie, Exploration	Mixte
Olympus	UC Berkeley	Simulation, Synthèse	Contrôle
Polis	UC Berkeley	Modélisation, Synthèse	Contrôle
Ptolemy	UC Berkeley	Simulation	TDSI
Scenic	UC Irvine/Synopsys	Simulation	Mixte
Specsyn	UC Irvine	Synthèse	Mixte

Tableau 1-1 : Travaux Universitaires

Projet	Origine	Problème visé	Application
Behavioral Compiler	Synopsys	Synthèse	Données
COSSAP	Synopsys	Spécification, Simulation	TDSI
Co-Centric	Synopsys	Synthèse	Données
CVE	Mentor Graphics	Co-Simulation	Mixte
Eaglei	ViewLogic	Co-Simulation	Mixte
Monet	Mentor Graphics	Synthèse	Données
Napkin to Chip N2C	Coware	Co-Simulation, Synthèse	TDSI
Protocol Compiler	Synopsys	Synthèse de protocole	Contrôle
SPW	Cadence	Spécification, Simulation	TDSI
VCC	Cadence	Exploration	Mixte

Tableau 1-2 : Travaux Industriels

2. Composant Virtuel

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

2.1. Historique et évolutions

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 dont la complexité et l'hétérogénéité sont grandissantes. La complexité des cœurs réutilisés n'a de ce fait n'a jamais cessé de croître. 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 cœurs utilisés sont majoritairement les processeurs généraux (CPU) et spécialisés (DSP) (voir Figure 1-5, source Gartner Dataquest) et les domaines d'applications ciblés s'orientent vers les communications et le traitement intensif de données *(EDP : Electronic Data Processing)* qui à eux seuls représentent la moitié du marché mondial des composants virtuels. (Figure 1-6, source Gartner Dataquest). Cependant, l'évolution des produits électroniques vers des applications portatives offre un avenir prometteur à la réutilisation d'accélérateurs matériels. En effet, la solution, consistant à concevoir des systèmes "tout logiciel" sur des architectures multiprocesseurs, est plus coûteuse en surface, en consommation électrique, en dissipation d'énergie et en performances temporelles qu'une solution utilisant des coprocesseurs matériels pour réaliser les traitements intensifs [EDW00].

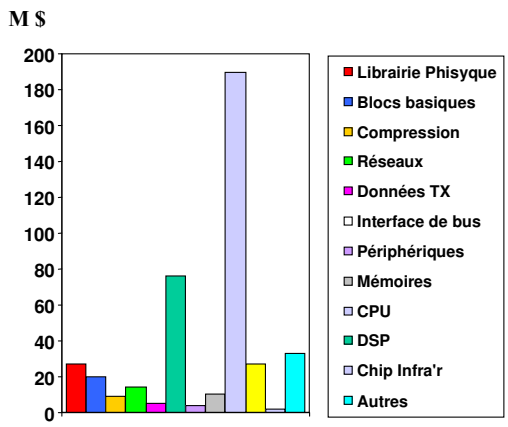


Figure 1-5 : Marché des Ips : revenus par type de composants

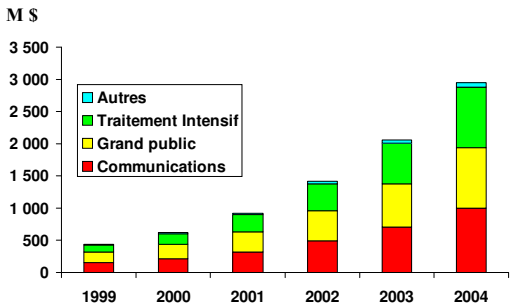


Figure 1-6 : Marché des Ips : revenus mondiaux par applications

2.2. Vers une standardisation

La conception et la réutilisation de composants virtuels posent un certain nombre de difficultés d'ordre méthodologique, le premier problème étant de définir clairement ce que l'on entend par "réutilisable". L'aptitude à intégrer un composant virtuel dans un système repose sur trois facteurs : (1) l'interopérabilité des formats de représentation des informations qui le constituent ; (2) la flexibilité, autrement dit l'aptitude de l'utilisateur à adapter la fonctionnalité et les performances d'un composant à ses besoins propres ; (3) la prédictibilité, c'est-à-dire la précision avec laquelle l'utilisateur peut estimer les performances d'un composant avant qu'il n'ait été intégré au système cible.

L'interopérabilité est étroitement liée aux standards régissant les niveaux d'abstraction des descriptions synthétisables et simulables d'un composant, les formats de données et langages de description, la liste minimale des délivrables pour garantir l'aptitude à réutiliser un composant en l'absence du concepteur. Le principal acteur en matière de recommandation et de standardisation concernant les composants virtuels est l'organisation VSIA (Virtual Socket

Interface Alliance) [VSI03], formée en 1996 dans le but de promouvoir la conception de systèmes intégrés par la réutilisation de composants provenant de sources variées. VSIA réunit essentiellement des entreprises industrielles, concepteurs de circuits et vendeurs d'outils de CAO, et se décompose en groupes de travail.

Les missions de ces groupes couvrent les objectifs suivants :

- définir un vocabulaire commun pour les niveaux d'abstractions, les modèles de représentation et les délivrables d'un composant virtuel;
- standardiser des méthodologies de conception et d'intégration de composants de différentes natures (numériques, analogiques, logiciels);
- standardiser des techniques de vérification des composants virtuels, seuls et à l'intérieur d'un système, à différents niveaux d'abstraction et après fabrication;
- définir des standards pour la protection de la propriété intellectuelle;

Loin de s'orienter vers la définition systématique de nouvelles techniques et standards, l'effort de standardisation de ces groupes porte de préférence sur la sélection de techniques existantes et la recommandation de pratiques éprouvées industriellement [VSI97].

2.2.1. Les niveaux d'abstraction

La spécification synthétisable d'un composant virtuel peut être délivrée sous différentes formes, en fonction de l'étape du flot de conception à laquelle elle a été extraite (e.g. avant ou après synthèse RTL, après placement/routage), du format ou langage utilisé par la description, et de la cible technologique considérée (bibliothèques standard-cell ou full-custom, familles de circuit programmables). VSIA définit trois classes de composants virtuels matériels en fonction du niveau d'abstraction de leur description synthétisable [VSI97] : (1) un composant virtuel *hard* se présente sous la forme de masques prêts à fondre ; (2) un composant virtuel *firm* se présente comme une netlist optimisée au niveau logique, et munie éventuellement de directives de placement ; (3) un composant virtuel *soft* se présente comme une description synthétisable en VHDL ou Verilog au niveau transfert de registres. Ces trois classes de composants virtuels se distinguent par des degrés différents de prédictibilité et de flexibilité.

- Les composants *hard* sont très prédictibles, mais très peu flexibles, car leur placement/routage interne est déjà optimisé pour des contraintes et une bibliothèque technologique fixées.
- Les composants *soft*, au contraire, sont peu prédictibles, car l'optimisation au niveau logique n'a pas encore eu lieu, et supportent un degré de flexibilité qui autorise la personnalisation non seulement de la technologie cible et des contraintes d'optimisation logique, mais aussi de paramètres architecturaux.

- Les composants *firm* sont fournis sous la forme d'un réseau de portes logiques pouvant être optimisé pour une bibliothèque technologie générique.

Les principales caractéristiques des niveaux d'abstraction sont résumées dans le Tableau 1-3.

	Flot de conception	**Représentation**	**Librairies**	**Technologie**	**Portabilité**
Soft Non prédictible Très flexible	Conception système	Comportementale		Indépendante	Illimitée
	Conception RTL	RTL			
Firm Flexible Prédictible	Synthèse du floor plan	RTL + Placement de sous-blocs	Librairie de référence	Générique	Library mapping
	Placement	Netlist	Modèle de temps et d'interconnexions		
Hard Pas flexible Très Prédictible	Vérification du routage	Polygones	Spécifique au process Règles de dessin	Fixe	Process mapping

Tableau 1-3 : Caractéristiques des niveaux de description

Dans la pratique, les intégrateurs de composants virtuels privilégient les IP soft pour leur flexibilité. Les créateurs, en revanche, apparaissent réticents à délivrer du code RTL synthétisable, plus difficile à protéger contre les modifications et réutilisations abusives. Dans un grand nombre de cas, un composant virtuel soft pour son créateur est en réalité délivré sous la forme d'un composant firm, la synthèse logique étant réalisée chez le créateur en fonction des directives de l'utilisateur. Les IP hard sont utilisés de préférences pour délivrer des modèles de processeurs et de mémoires, comme l'indique la Figure 1-7.

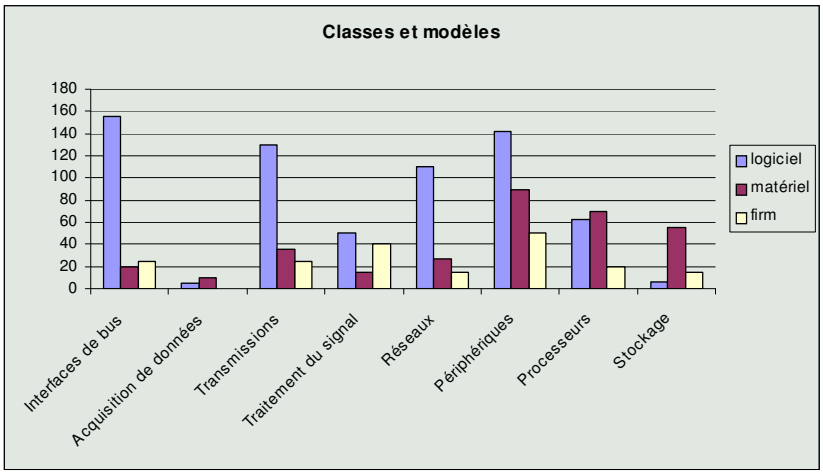


Figure 1-7 : Classes et modèles [ITR01]

2.2.2. Les délivrables

VSIA a établi une liste conséquente des délivrables associés à chaque classe de composants virtuels [VSI00] dont une partie est présentée dans le Tableau 1-4. Ces délivrables comprennent notamment un manuel utilisateur, un ou plusieurs modèles simulables du composant et un modèle synthétisable. Le manuel utilisateur décrit tout d'abord la fonctionnalité, l'interface, les éventuels paramètres et les performances du composant. Il détaille les procédures à suivre pour simuler le composant seul et une fois intégré au système, le personnaliser et le synthétiser.

Objet	Format	Soft	Firm	Hard
Guide utilisateur				
Spécifications	Document	M	M	M
Caractéristiques	Document	M	M	M
Vérifications	Document	M	M	M
...				
Architecture système				
Modèle d'évaluation système	C, C++, VHDL, Verilog	CR	CR	CR
Conception système				
Test Bench	VHDL, Verilog, +PLI	R	R	R
Modèle comportemental	C, C++, VHDL, Verilog	R	CM	M
Modèle Processeur	C, C++, VHDL, Verilog	R	CM	M
Modèle fonctionnel des bus	VHDL, Verilog	R	R	R
Conception logique				
Source RTL synthétisable	Sous-ensemble synthétisable (VHDL, Verilog)	M	CM	CR
...				

M : Obligatoire	R : Recommandé	CM : Obligatoire Conditionnellement CR : Recommandé Conditionnellement

Tableau 1-4 : Délivrables VSIA

Les modèles simulables fournis avec un composant virtuel répondent à trois objectifs : (1) évaluer avant achat la conformité du produit, en termes de fonctionnalité et de performances temporelles, avec les besoins de l'utilisateur ; (2) vérifier après achat que le modèle synthétisable est correct ; (3) accélérer la simulation fonctionnelle et temporelle du système en cours de conception en y insérant un modèle comportemental exact au cycle près du composant.

Le modèle synthétisable, enfin, est fortement dépendant de la classe à laquelle appartient le composant virtuel. Les formats de description standard de facto sont les langages VHDL et Verilog pour les descriptions RTL, EDIF pour les netlists et GDSII pour les masques. Un modèle synthétisable s'accompagne de scripts de synthèse, modifiables par l'utilisateur, pour les principaux outils du commerce.

2.2.3. Les protocoles

Durant la phase d'exploration architecturale ou de synthèse des communications le concepteur système doit sélectionner un support physique permettant aux composants d'échanger des

données. Deux familles d'interconnexion s'offrent alors : le modèle réseau et le modèle bus partagé.

Les réseaux, tel RSPIN [GUE00] ou OCTAGON [KAR02], utilisent un modèle de communication distribuée et tentent de répondre aux problèmes de débit, latence, flexibilité et réactivité que requièrent de plus en plus les applications TDSI de future génération telles que MPEG-4, radio logicielle et téléphonie mobile. Cette solution est à l'heure actuelle surtout utilisée dans le domaine de la recherche.

La deuxième famille, largement répandue, est bien adaptée aux échanges de données réguliers point à point. Cette solution repose sur un mécanisme d'arbitrage centralisé qui malheureusement peut devenir le goulet d'étranglement lorsque le nombre de composants accédant au média augmente. Un grand nombre de bus, dont les principales propriétés sont présentées dans [COU02G] (fourni en annexe), est actuellement disponible sur le marché : AMBA [ARM99], Core Connect [IBM03], Core Frame [COR99] [PAL01], PI Bus [OMI96], Silicone Backplane [SON00A], et Wishbone [SIL01]. Afin de limiter les problèmes d'interconnexion VSIA a dans un premier temps proposé de définir un bus 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. Ce type de connexion, point à point, permet au concepteur d'IP de délivrer des composants tout en s'affranchissant du moyen de communication physique auquel il sera relié (voir Figure 1-9, Figure 1-10). Les composants virtuels ainsi fournis s'adaptent à tous types de média à l'aide d'un wrapper dans le cas des bus partagés. VSIA définit trois niveaux de complexité de protocole pour s'adapter aux niveaux hiérarchiques des bus système actuellement proposés : AVCI (Advenced VCI), BVCI (Basic VCI) et PVCI (Peripheral VCI). Ce dernier, décrit dans la Figure 1-8, est un sous-ensemble du BVCI et est dédié aux composants simples. Le niveau avancé ajoute des identifiants de requêtes permettant la gestion désordonnée des échanges.

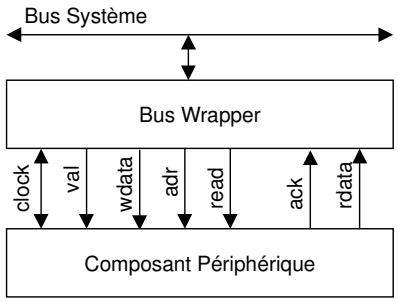


Figure 1-8 : Signaux de l'interface PVCI

Des propositions similaires de protocoles configurables, point à point, non spécifiques à un bus sont proposées : Open Core Protocole (OCP) [SON00B] et IP Interface (IPI) [MOT99].

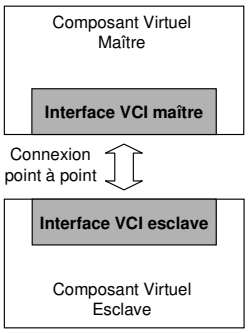


Figure 1-9 : Connexion directe

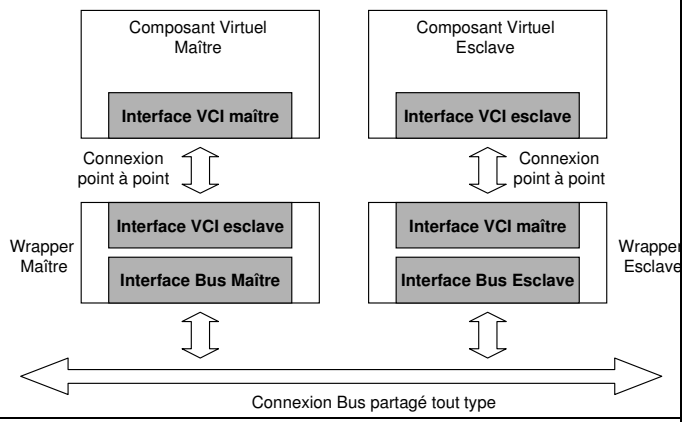


Figure 1-10 : Connexion par bus partagé

L'utilisation de ce type d'interface standard augmente cependant le temps de communication [CYR01] lorsque les composants échangent des données au travers de bus partagés. Le délai introduit par la conversion de protocole est dans certaines applications non négligeable et peut être réduit en insérant des caches dans les wrappers [LYS00].

3. Intégration des composants virtuels

3.1. Domaine d'application

Les parties calculatoires des applications dans les domaines du traitement du signal, de l'image (TDSI) et les télécommunications ont un nombre important de caractéristiques communes que nous exploitons dans notre approche d'intégration de composants virtuels. Les caractéristiques sont les suivantes :

- **Fort volume de données :** la fonctionnalité des systèmes est basée sur le traitement intensif de flux de données. Le concepteur du système devra donc se concentrer sur l'utilisation d'architecture de communication évitant les congestions des transferts sur les média de communication et dans les tampons tout en réduisant le coût du stockage des données.

- **Flux de données multidimensionnelles :** le traitement de l'information est typiquement réalisé sur des flux de données structurées représentés sous la forme de matrices mono ou multidimensionnelles.

- **Motifs de calcul répétitifs :** les traitements réalisés dans les applications TDSI ont une structure régulière. La description des opérations réalisées sur des tableaux multidimensionnels est spécifiée à l'aide de boucles imbriquées permettant la répétition d'un motif de calcul.

- **Temps réel :** la nature temps réel des applications du domaine ajoute des contraintes fortes sur les phases de spécification, d'exploration architecturale et d'intégration visant les systèmes embarqués.

- **Structure modulaire :** les flux de données dans des systèmes orientés vers le multimédia ou les télécommunications, sont soumis à un grand nombre de transformations consécutives (Codage source, cryptage, codage canal, modulation) qui peuvent elles même être constituées de sous étapes (codage canal : codeur Viterbi, entrelacement, codeur Reed-Salomon).

- **Fonction standard :** la structure modulaire est la plupart du temps basée sur des fonctions algorithmiques standards : Les filtres (FIR, IIR...), les transformations (DCT, FFT) et les contrôles d'erreurs (Viterbi, Turbo Codes, MAP, Reed-Salomon).

- **Contraintes de conception :** ces applications sont fortement utilisées dans des produits grand public qui requièrent un délai de mise sur le marché extrêmement court. A cela s'ajoutent des contraintes drastiques de consommation et de surface lorsque les systèmes sont embarqués dans des appareils électroniques portatifs.

3.2. Problématique

Suivant la taxonomie VSIA, le plus haut niveau d'abstraction pour les modèles d'IP synthétisables correspond aux IP "soft". Ces blocs fonctionnels sont fournis au niveau transfert de registres (RTL) dans un langage de description de matériel (VHDL, Verilog) et sont synthétisés à l'aide d'outils de synthèse logique. Bien que de telles descriptions puissent être paramétrables (par utilisation de mécanismes "généric"/"generate" fournis par le langage) elles reposent sur un modèle d'architecture établie avec des degrés de personnalisation très réduits. De ce fait, l'unique objectif des spécifications de composant virtuel au niveau comportemental est l'accélération de la simulation du système dans lequel il doit être intégré. Le manque de flexibilité des cœurs d'IP de niveau RTL est particulièrement vrai pour l'unité de communication dont les ordres d'acquisition et de production de données, le comportement temporel et les protocoles d'échanges sont figés. Les composants virtuels sont ainsi connectés au bus système par des d'interfaces spécifiques ou "wrapper". Une intégration réussie d'un composant virtuel requière de la part du concepteur de SoC de concevoir un wrapper qui (1) synchronise les composants (échanges de données, protocole) et convertit si besoin est les protocoles entre blocs "incompatibles" et (2) temporise les données pour garantir les contraintes temporelles et l'ordre des données [KEA99]. L'utilisation de protocoles standards comme VCI [OCB00] ou OCP [SON00] n'exempte cependant pas le wrapper de la tâche de temporisation des données.

Ainsi, les méthodologies partant du principe qu'un composant virtuel doit être réutilisé sans modification ne sont plus valides dans les applications dominées par le traitement des données. En effet, les systèmes travaillant sur de forts volumes d'information requièrent, afin d'obtenir une solution viable d'implémentation, une évaluation et une prise en compte de l'organisation des échanges et du stockage des données. La réutilisation de composants inadéquats en terme d'entrée/sortie aboutit à une adaptation coûteuse en terme de latence, débit et surface des

transferts de données. Nous illustrons ce problème par un exemple décrit dans la Figure 1-11. Supposons que nous voulions réutiliser un composant DCT RTL conçu pour acquérir ses entrées ligne par ligne et produire ses résultats suivant le même schéma. 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 (1) le producteur fournit les données au bloc DCT colonne par colonne et (2) le consommateur acquiert les données du bloc DCT colonne par colonne. Ce cas de figure impose l'utilisation d'un ensemble de tampons adaptant les séquences d'entrée et d'un ensemble de tampons adaptant les séquences de sortie. L'utilisation de tels adaptateurs fonctionnels, introduit un surcoût en surface, consommation et performance qui peut aboutir à la violation des contraintes imposées pour la conception de systèmes embarqués (voir chapitre 5).

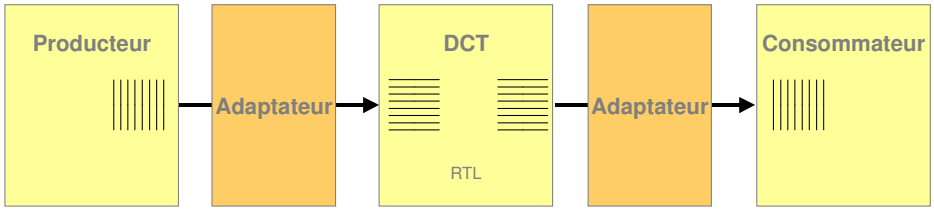


Figure 1-11 : Problématique de réutilisation

4. Bilan

Comme nous l'avons vu au cours de ce chapitre, l'intégration des composants virtuels ne concerne plus uniquement le niveau implémentation mais doit couvrir l'ensemble du flot de conception d'un SoC allant ainsi de la description système jusqu'à la description bas niveau des composants de l'architecture [GAJ98],[FER99],[LEN00]. Cependant le plus haut degré d'abstraction des descriptions actuelles est le niveau transfert de registre (IPs *soft*) qui repose sur des modèles architecturaux figés. De ce fait, l'intégration de ces composants virtuels requiert des techniques qui, dans les applications TDSI, peuvent faire échouer la réutilisation d'un composant. En effet, l'utilisation d'adaptateurs peut aboutir à la violation des contraintes (1) de performance telles que la latence ou le débit, (2) de coût dû à la surface introduite par le module adaptant les séquences d'E/S, (3) de consommation liée à une utilisation trop importante d'éléments mémorisants.

Les fonctions standards utilisées dans les applications orientées TDSI et Télécommunication ont un faible degré de variation au niveau purement fonctionnel mais offrent un large choix d'algorithmes : une DCT-2D peut être par exemple décrite fonctionnellement par deux produits de matrices mais être réalisée par un algorithme utilisant un, deux ou trois niveaux de papillons chacun impliquant un ensemble particulier de contraintes d'E/S. Ces algorithmes sont dans un premier temps, spécifiés et vérifiés à un niveau élevé d'abstraction à l'aide d'un outil comme Matlab. Une fois leur validité prouvée, ces représentations en virgule flottante sont alors traduites manuellement dans leurs équivalents VHDL ou Verilog en virgule fixe au niveau d'abstraction RTL. Cependant, les méthodologies conventionnelles permettant ce raffinement sont source d'un grand nombre d'erreurs. En effet, les représentations Matlab et RTL sont conceptuellement différentes, ce qui rend la traduction manuelle longue et difficile. Une fois

que le RTL initial a été produit, l'évaluation des architectures alternatives est également longue et difficile et demande des temps de simulation fastidieux. De plus, l'exploration du vaste espace de solutions architecturales offert ne peut être couvert par une génération manuelle.

Cette conjoncture offre de fortes perspectives en terme de réutilisation des composants virtuels de niveau algorithmique dans le domaine du traitement du signal et de l'image et des télécommunications. Ainsi, nous pensons que la synthèse comportementale, qui permet d'obtenir à partir d'une description source un ensemble d'architectures, peut être pilotée par les contraintes d'intégrations afin de générer un composant RTL adapté aux besoins du système dans lequel il doit être intégré (voir Figure 1-12).

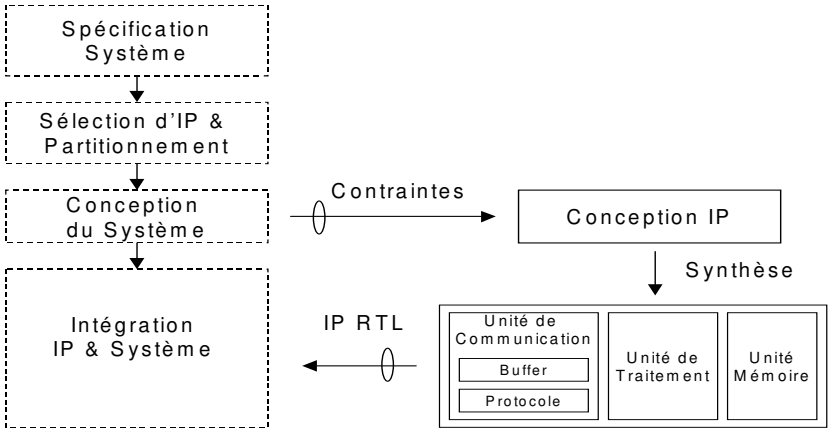


Figure 1-12 : Flot de conception d'un SoC

5. Objectifs et Contributions

Dans ce contexte, nous proposons de rehausser le niveau d'abstraction des descriptions en introduisant la notion de *composant virtuel comportemental* pour les applications orientées vers le traitement du signal, de l'image (TDSI) et les télécommunications. Ces composants sont décrits sous une forme algorithmique dans un langage de haut niveau comme le *VHDL ou SystemC,* et autorisant un haut degré de flexibilité par le biais d'un jeu de paramètres génériques [SAV02] [SAV01], [PIL01].

L'architecture ciblée par la synthèse est composée d'une unité de traitement, d'une unité de mémorisation, d'une unité de contrôle et d'une unité de communication. L'ensemble du flot de conception est réalisé *sous contrainte d'intégration* ce qui garantit une conception optimisée en terme de consommation, surface et performances de toutes les unités fonctionnelles. La synthèse de l'unité de communication est réalisée a posteriori de celle de l'unité de traitement, ce qui permet une prise en compte des contraintes algorithmiques et des contraintes d'intégration. Ainsi dans cette approche, l'unité de communication, qui permet l'adaptation des séquences d'E/S entre l'IP et le système, n'est plus un module additionnel mais une partie intégrante du composant.

La méthode présentée dans ce manuscrit vise donc à assister le travail de l'intégrateur système en proposant (1) un ensemble de modèles pour représenter les contraintes système et les contraintes algorithmiques (2) des méthodes et techniques pour synthétiser les différentes parties composant l'architecture de IP à intégrer.

Chapitre 2

Etat de l'art

Dans le chapitre précédent nous avons situé et présenté la problématique de réutilisation de blocs préconçus dans un flot de conception de systèmes embarqués. Nous décrivons maintenant plus en détails les méthodes proposées dans la littérature pour résoudre le problème de l'intégration des composants virtuels de niveau RTL. Dans une deuxième partie nous présenterons les travaux relatifs à la synthèse haut nivaux sous contraintes temporelles et mettrons en avant les besoins spécifiques que requiert notre approche.

1. Intégration des Composants virtuels

1.1. Introduction

Les différentes approches de conception autorisent de plus en plus la réutilisation de composants virtuels dans leurs flots. Les méthodologies de type *System Synthesis* définissent une méthode d'encapsulation permettant la réutilisation des Ips à tous les niveaux de description d'une application. Les flots proposés sont fortement liés à des langages tels que SpecC ou SystemC et ne proposent pas, à proprement parler, de technique de synthèse de wrapper. Les méthodologies orientées *Component-Based Design* utilisent quant à elles une architecture générique d'adaptateur pour générer une plate-forme contrairement aux méthodologies *Platform-Based Design* qui se basent sur des architectures ciblées de modules d'interfaces induites par les schémas de communication de la plate-forme.

Dans tous les cas, l'intégration des composants virtuels de niveau RTL est réalisée en utilisant un wrapper. Il existe trois types d'outils pour leur génération : les générateurs de wrapper matériel, les générateurs de wrapper pour la cosimulation et les générateurs de wrapper logiciel. La première catégorie, s'appuie sur des librairies de descriptions paramétrables et synthétisables pour générer un wrapper matériel. La deuxième catégorie produit un modèle exécutable qui est utilisé pour valider le comportement du composant et du système dans lequel il est intégré. La dernière catégorie, qui ne sera pas détaillée ici, produit des systèmes d'exploitation utilisés pour exécuter plusieurs taches logicielles sur un même processeur.

Bien que les flots de conception aboutissent dans tous les cas à une synthèse d'interface, les solutions apportées au problème d'intégration des composants virtuels se distinguent par leur orientation tournée vers la synthèse ou la simulation (voir Figure 2-1). C'est donc sur un découpage par famille méthodologique que nous avons choisi de présenter les travaux relatifs à l'intégration de composants virtuels de niveau RTL dans ce chapitre.

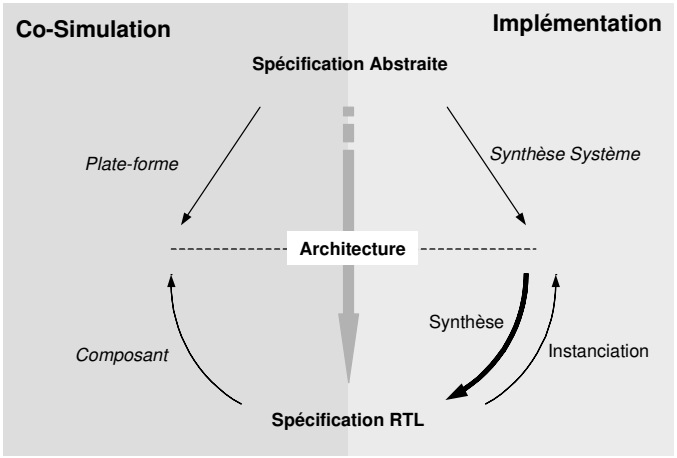


Figure 2-1 : Différents types de solution

1.2. Les approches orientées *" Synthèse "*

1.2.1. L'approche *"Three-Layer-Formalism"*

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 *firm* et *soft*. 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 spécifie le contrôle du processus (voir Figure 2-2).

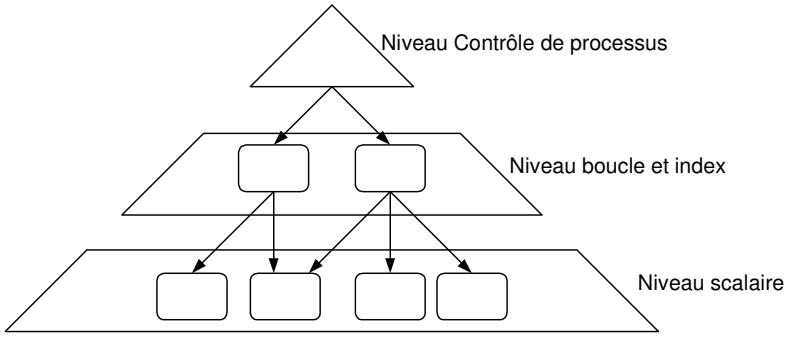


Figure 2-2 : Décomposition en trois niveaux hiérarchiques

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 (voir Figure 2-3 (a)) et/ou ré-ordonnent les accès aux structures multidimensionnelles en réalisant des calculs sur les indices de boucles. (voir Figure 2-3 (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 2-3 : Exemple de modification de calcul d'indice de boucles

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.

1.2.2. L'approche *"System-Synthesis"*

Le flot de conception proposé dans [GAJ98] utilise comme point d'entée une spécification fonctionnelle exécutable décrite en SpecC. Pour cela, le concepteur peut réutiliser des portions de code, des fonctions ou des procédures en les instanciant depuis une librairie d'algorithmes. La phase d'*exploration architecturale* qui est ensuite réalisée est un processus itératif incluant *l'allocation*, le *partitionnement* (des fonctions, canaux et variables) et *l'ordonnancement*. L'allocation détermine le nombre et le type des composants du système (CPU, DSP, mémoires, bus, IP). Le *partitionnement des fonctions* distribue les traitements sur différentes unités de calculs (*PE Processisng Element*), le *partitionnement des variables* assigne les variables aux mémoires et le *partitionnement des canaux* assigne les canaux de communication aux bus. *L'ordonnancement* détermine l'ordre d'exécution des calculs sur les processeurs standards ou spécialisés après partitionnement. Cette phase d'exploration architecturale permet d'aboutir à un modèle d'architecture à partir duquel la synthèse des communications va être réalisée.

Durant l'étape d'exploration, les communications sont réalisées par des variables partagées. La phase de raffinement des communications regroupe puis encapsule ces variables dans des canaux virtuels (Figure 2-4). Ces canaux abstraits réalisent les communications par de passage de messages et supportent un grand nombre de types complexes de données ainsi que des mécanismes de synchronisation (bloquant, non bloquant, fifo…).

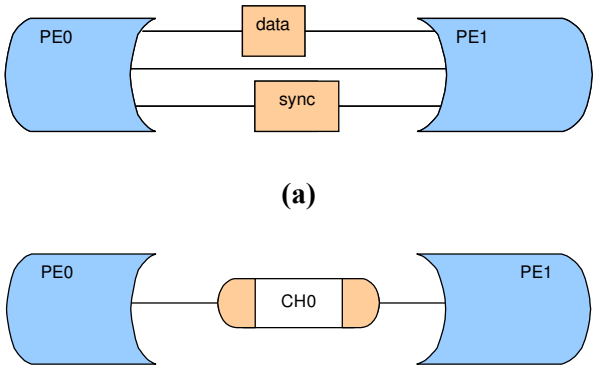


Figure 2-4 : Modèle (a) avant et (b) après partitionnement des canaux

Les canaux abstraits sont ensuite remplacés par (1) des bus personnalisés ou par des bus sous forme d'IPs, et (2) des appels de pilotes logiciels et/ou (3) des adaptateurs matériels. Cette phase inclut l'insertion des protocoles, la synthèse des interfaces et des transducteurs et la mise en ligne des protocoles dans les composants.

- *Insertion de protocole* : les spécifications du protocole sélectionné par le concepteur sont décrites avec les primitives des canaux en SpecC. Ces appels de fonctions sont remplacés par des instructions locales d'E/S pour la partie logicielle et par de nouveaux composants pour la partie matérielle.

- *Insertion des transducteurs :* chaque fois que sont trouvés des protocoles de bus incompatibles dans un même canal, des transducteurs sont insérés (séparation du bus virtuel en deux bus virtuels connectés par un transducteur). Le transducteur est inséré comme un *comportement* qui transfère un message d'un bus virtuel à un autre (Figure 2-5).

```
behavior Transducer ( IBus0Send i0 , IBus1Receive i1 ) {
int [ 10 ] [ 10 ] temp ;
void main ( void ) {
  i0 . receive ( temp ) ;
  i1 . send ( temp ) ;
 }
}
```

Figure 2-5 : code d'un transducteur

- *Mise en ligne de protocole* : pour les fonctions réalisées par du matériel les méthodes du canal sont insérées dans le composant qui lui est connecté. Les ports de son interface sont composés de signaux au niveau bit. Cette opération est automatique car toutes les informations nécessaires se trouvent dans le modèle architectural.

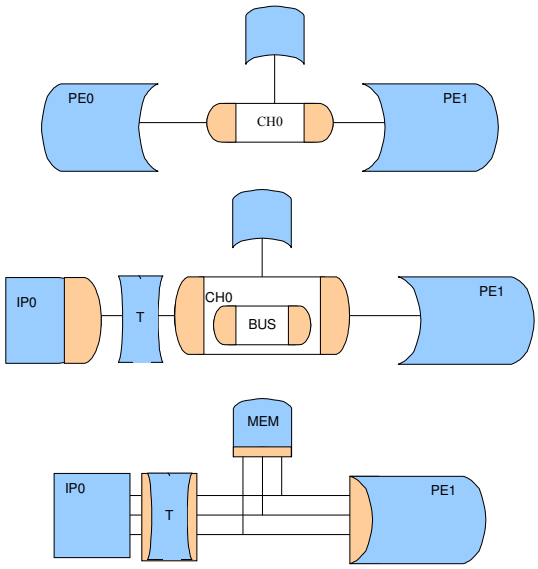


Figure 2-6 : Raffinement des communications

La *taille des canaux* est fixée pour correspondre à la largeur des bus utilisés. Après comparaison, une procédure de conversion transforme le type de donnée complexe utilisé par le canal virtuel en un flux de donnée de type vecteur de bit. Tout ceci aboutit à l'insertion d'une nouvelle fonctionnalité (boucle *for* par exemple) qui permet la réalisation de transferts valides sur le bus. De plus, lorsque plusieurs canaux virtuels sont groupés sur un seul canal, un arbitre

est inséré pour résoudre les problèmes de conflits d'accès au bus. Dans le modèle de communication résultant la communication est décrite en terme de connexions et les relations temporelles par des protocoles de bus.

Bien que les auteurs ne définissent pas d'architecture précise des transducteurs, ils incluent dans ces derniers un contrôleur réalisant les protocoles d'échange [SHI02A] et des files d'attentes dont les caractéristiques sont les suivantes [SHI02B] :

- la largeur en nombre de bits $b_{wq} = max[bw_s, bw_r]$ où bw_s, bw_r représente respectivement la largeur des mots du producteur et du consommateur;

- la profondeur Q_n à un instant n, en nombre de mot de taille b_{wq} est $Q_n = max(0, Q_{n-1} + (P_n - C_n))$ avec P_n la quantité de données produite à l'instant n et C_n la quantité de données consommée. La profondeur Q est définit comme le maximum de tous les Q_n.

1.3. Les approches orientées *"Simulation"*

1.3.1. L'approche *"Component-Based Design"*

[NIC02], [CES02], [YOO01] et [LYO01] utilisent un flot de conception basé sur la séparation du traitement et de la communication qui autorise un raffinement indépendamment des deux aspects (voir Figure 2-7). Le système est décrit comme un réseau hiérarchique de *modules* constitués de ports et de comportements *(behavior)*. Les modules communiquent au travers de leurs ports par des canaux *(communication channel)*. La communication est réalisée par appel de fonctions *(port functions)*. Un wrapper est composé d'un ensemble de ports internes et externes et d'un comportement. La Figure 2-8 décrit deux modules, leurs wrappers respectifs ainsi que les canaux qui les relient.

C'est dans ce contexte que les auteurs proposent une approche permettant d'interconnecter des composants de natures hétérogènes en vue de leur simulation. Pour cela ils définissent une architecture générique de *wrapper* permettant la communication entre (1) des modules utilisant des protocoles incompatibles (2) des modules décrits à niveaux hétérogènes d'abstraction ou (3) des modules de niveaux hétérogènes utilisant des protocoles incompatibles. Les niveaux d'abstraction considérés sont : *System Level (SL), Architecture Level (AL)*, *Register Transfer Level (RTL)*.

Au *niveau système*, la communication se fait au travers d'interconnexions abstraites (canaux SL) par *passages de messages* : il n'y a pas de protocole spécifique à ce niveau de description. Le canal fournit aux ports deux primitives pour accéder au canal SL : *send, receive*. Les données échangées n'ont pas de type particulier mais un type de donnée générique**.**

Au *niveau architectural***,** les canaux fournissent leur propre protocole (FIFO, file de messages, handshake…) ainsi que les paramètres (taille des FIFO…). Les données qui transitent par le canal ont un type fixé (int, float…). Le canal fournit aux ports les fonctions adaptées à son protocole (fifo_available, fifo_write…). Chaque module est relié à une ressource matérielle.

Au *niveau RTL*, les coprocesseurs sont connectés au réseau physique d'interconnexion par un coprocesseur de communication *(CC)*. Les processeurs ont, en plus du *CC*, un système d'exploitation *(OS)* qui permet la communication entre les tâches logicielles qu'ils exécutent.

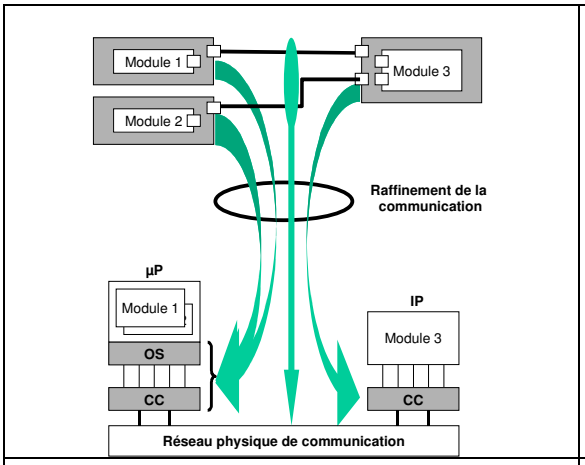


Figure 2-7 : Raffinement des communications

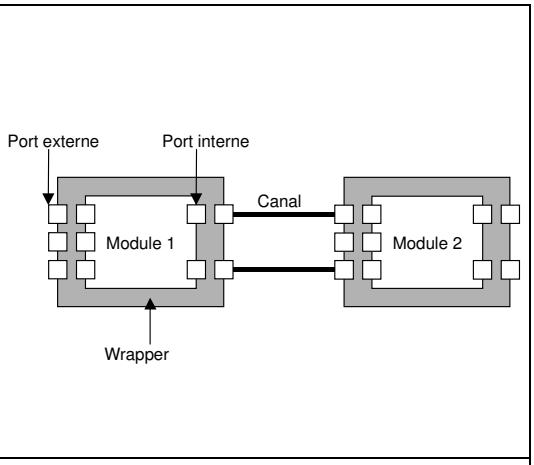


Figure 2-8 : Modules, wrappers et canaux

L'architecture du wrapper (voir Figure 2-9) est composée d'un ensemble de ports internes et externes, d'adaptateurs de module *(MA)*, d'adaptateurs de canaux *(CA)* et de média de communication interne *(ICM)*. Ainsi, les *CA* convertissent le protocole du canal externe vers le protocole de *l'ICM* et les *MA* convertissent le protocole de *l'ICM* vers celui du module. Pour chaque type d'adaptation de protocole ou de niveau d'abstraction, une instance de l'architecture du wrapper est construite. Le module *ICM* est utilisé pour transférer les données entre l'adaptateur de module et le ou les adaptateurs de canaux. En fonction des niveaux d'abstraction du module *l'ICM* peut être (1) un appel de fonction (module niveau système ou architectural) ou (2) un bus interne (module décrit au niveau RTL). Afin de permettre l'automatisation de la génération du wrapper, les auteurs supposent que le raffinement d'un module implique le raffinement de tous ses ports internes. Cette restriction réduit la généralité du wrapper puisqu'un grand nombre de méthodologies autorise le raffinement séparé des canaux.

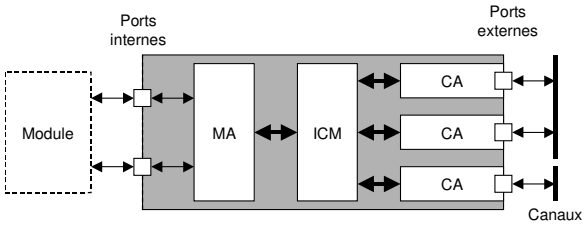


Figure 2-9 : Architecture générique du wrapper

Après avoir dimensionné, par un ensemble de simulations et de raffinements itératifs, les paramètres du système et les caractéristiques des wrappers, les modèles simulables sont alors remplacés par des modèles synthétisables provenant de librairies [DAV01].

Ainsi, les auteurs utilisent deux types de librairies : la *librairie de protocoles* et de *librairie de processeurs*. La *librairie de protocoles* est une liste d'adaptateurs de canaux. Chaque adaptateur de canal a un modèle de synthèse, de cosimulation et d'estimation paramétrable (type de

données, direction, taille mémoire). Le modèle d'estimation permet une estimation performance/coût du protocole en terme de surface, consommation et latence. La *librairie de processeurs* est constituée de bus locaux, de cœurs d'IP, de mémoires locales, de décodeurs d'adresse, et de coprocesseurs. Chaque *processeur* a un modèle pour la cosimulation (ISS) et un modèle pour la synthèse (layout macro). Les modèles ont des paramètres génériques ciblant des applications spécifiques. Il y a donc une *cosimulation library* et une *synthesizable code library***.**

La contrepartie de la généricité de cette approche est qu'elle repose sur d'une part sur une exploration itérative et d'autre part sur l'existence de bibliothèques contenant tous les modèles possibles de communication et ce, pour la simulation et la synthèse.

1.3.2. L'approche "Platform-Based Design"

Afin d'illustrer l'intégration des composants virtuels dans les approches *Platform-Based Design* nous présentons les travaux détaillés dans [HOM01A][HOM01B]. Le module d'interface proposé est utilisé dans le projet COSY [Bru00] qui définit un ensemble de niveau de description permettant le raffinement des communications dans des environnements de conception tel que VCC [CAD03B]. Cet outil, originellement nommé *FELIX initiative,* est issu de la recherche universitaire et industrielle et repose sur le flot de conception de POLIS ([KIE97], [BAL97]) décrit dans la Figure 2-10. L'utilisation de COSY et VCC a permis la modélisation, l'analyse et l'ordonnancement d'un système de diffusion vidéo numérique [BRU99][KEN99] dans lequel la spécification a été réalisée à l'aide de l'outil YAPI [DeK00]. Malheureusement, les auteurs ne présentent pas les résultats obtenus en terme d'implémentation matérielle et de synthèse des interfaces.

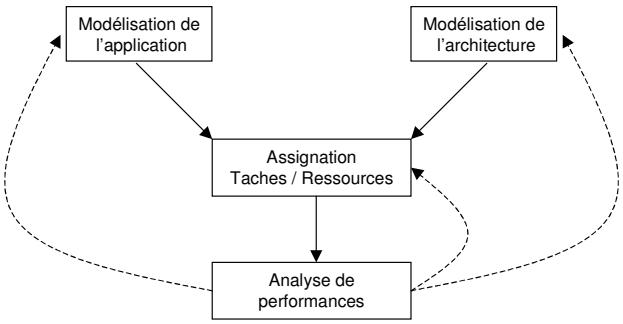


Figure 2-10 : Flot Y-chart

Dans [HOM01A] les auteurs réalisent la synthèse des communications à partir de trois descriptions : la spécification parallèle de l'application, l'architecture cible ainsi que le choix d'implémentation de chaque tâche en matériel ou en logiciel. Dans ce contexte, la spécification parallèle de l'application repose sur le formalisme *KPN (Kahn Process Network [KAH74])* qui décrit le système comme un ensemble de tâches exécutant chacune un programme séquentiel et communicant au travers de FIFO de taille infinie. Les tâches sont des *threads POSIX* dont le comportement est décrit à l'aide du langage C. Les primitives de communications bloquantes

read() et *write()* du modèle Kahn sont réalisées par les moyens de synchronisation POSIX [PET03] : *mutex* et *condition.* L'architecture cible peut contenir un ou plusieurs CPU, plusieurs coprocesseurs de type maître ou esclave et doit fournir, afin de connecter les composants au média de communication, les wrappers réalisant la conversion protocole VCI / protocole bus système. En fonction de la nature matérielle ou logicielle de la tâche productrice et de la tâche consommatrice, le concepteur système spécifie un schéma de communication décrivant les moyens utilisés pour les transmissions et la synchronisation. La méthode de synthèse des communications se résume pour la partie matérielle à instancier le ou les modules d'interface nécessaires, et pour la partie logicielle à remplacer les primitives utilisées dans la description parallèle par celles correspondant au schéma de communication. Pour cela les auteurs ont défini une architecture générique de l'interface matérielle présentée dans la Figure 2-11. Ce module d'interface est composé de sous modules qui ne sont instanciés que si les services utilisés par le coprocesseur ne l'exigent.

Ainsi, l'interface comprend deux sous-modules VCI. La cible VCI communique avec le wrapper esclave et est utilisée pour accéder à toutes les ressources adressables du module d'interface. L'initiateur VCI communique avec le wrapper maître et est utilisé par les FIFOs maîtres pour émettre des requêtes vers un esclave sur le bus. Les FIFOs permettent de réaliser les canaux en fonctions des types maître/esclave et producteur/consommateur du coprocesseur. Le concentrateur d'interruption permet de n'utiliser qu'une ligne d'interruption par interface. Les registres d'états sont lisibles par les processeurs du système et indiquent l'état du coprocesseur (code erreur...). Les registres de configuration sont quant à eux accessibles en lecture / écriture par les processeurs du système et sont utilisés pour initialiser ou programmer le coprocesseur et spécifier les variables globales du système (tailles d'une image...).

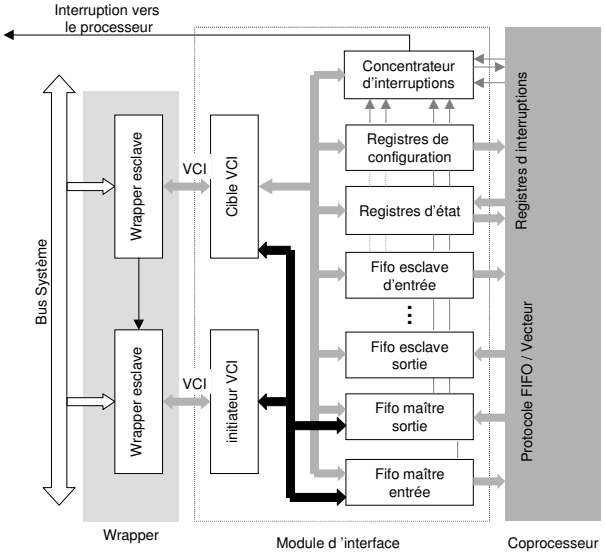


Figure 2-11 : Architecture générique du wrapper

L'utilisation de ce module d'interface générique permet ainsi le transfert des données et la synchronisation entre un coprocesseur et le système. L'utilisation de protocole FIFO permet au concepteur du coprocesseur de s'affranchir de la synthèse de protocole complexe et de la prise

en compte des contraintes temporelles des signaux de contrôle de bus. La synthèse est réalisée par instanciation de modèle générique VHDL.

La méthodologie et l'architecture générique de l'interface proposée repose sur l'utilisation de FIFO pour modéliser dans un premier temps puis synthétiser dans un deuxième temps la communication. Malheureusement, ce modèle ne prend qu'un unique schéma d'échange de données entre le composant virtuel et le système : les ordres d'acquisition et de production doivent être identiques.

1.4. Conclusion

Comme nous l'avons vu dans les sections précédentes, les approches présentées permettent l'intégration à gros grain de composants virtuels en utilisant des canaux de type FIFO qui autorisent, dans une certaine limite, l'adaptation des débits entre les composants et un lissage des transferts. En effet, elles reposent sur l'hypothèse forte suivante : l'IP/coprocesseur RTL a des séquences d'acquisition, et de production, exactement identiques à celles des composants avec lesquels il communique. De ce fait, aucune gestion des séquences d'entrée/sortie n'est proposée. Or, comme nous l'avons déjà mentionné, la temporisation des données permettant d'adapter les transferts, peut accroître considérablement la taille des FIFOs et des interfaces et augmenter ainsi le coût en surface et en consommation du système. De plus, l'utilisation de FIFO ne permet pas la conversion des ordres de transferts. Ce type de solution basé sur des cœurs de niveau RTL peut aussi aboutir dans certains cas, à la violation des contraintes temps réel caractéristiques des applications du Traitement du Signal et de l'Image (TDSI) et des Télécommunications et faire échouer la réutilisation de blocs préconçus. L'utilisation d'une description plus élevée du composant virtuel que nous proposons permet la gestion des contraintes de communication durant les phases de conception et d'intégration du composant virtuel : elle requiert pour cela un ensemble de techniques de raffinements accessible au travers d'un flot basé sur la synthèse haut niveau.

2. Synthèse architecturale

Cette section décrit brièvement, dans une première partie, les principales étapes constituant le flot typique de la synthèse architecturale aussi nommé synthèse haut niveau ou synthèse comportementale. Il existe un grand nombre d'outils concernant ce type de synthèse qui se différencient par leur domaine d'application et les contraintes supportées : GAUT [MAR93], AMICAL [PAR93], CATEDRAL, CADDY/DSL [CAM89], CALLAS [STO92] ou HERCULES/HEBE [DeM88] pour les travaux académiques et MONET [ELL00], A|RT Designer [ADE03], Get2Chip-AC [G2C03] ou Behavioral Compiler [KNA96] pour les produits industriels. Les outils AMICAL et GAUT seront dans un premier temps présentés. Nous détaillerons ensuite deux outils permettant la synthèse sous contraintes de temps : HERCULES/HEBE développé à l'université de Stanford dont nous avons étendu le formalisme *SIF (Sequencing Intermediate Format)* [KU92] et Behavioral Compiler de Synopsys le plus représentatif des outils commerciaux, récemment étendu par SystemC Compiler pour supporter le langage SystemC.

2.1. Flot de synthèse

Le flot typique de synthèse de haut niveau produit une description structurelle d'une architecture en transformant, par un ensemble d'étapes successives, la description initiale de l'algorithme.

Ainsi, l'étape de **compilation** réalise la vérification syntaxique et sémantique de la description source 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...

Deux types de modèles sont couramment employés pour la **représentation intermédiaire**. Les Graphes Flot de Données *(DataFlow Graph DFG)* et les Graphe flot de données et de contrôle (Control and Dataflow Graph CDFG). Le premier modèle permet d'exhiber les dépendances de données et d'exprimer le parallélisme maximum des calculs des applications ne comportant pas de non-déterminisme : le nombre d'itérations des boucles doit être connu statiquement. Cette restriction peut être est évitée en utilisant les CDFG qui sont plus adaptés aux applications orientées contrôle.

L'étape de **sélection** consiste à choisir la nature des ressources matérielles (opérateurs) qui réaliseront les opérations. Le choix des composants se fait sur des critères tels que la surface, la vitesse ou la consommation.

L'étape **d'allocation** détermine, pour chaque type d'opérateur sélectionné, le nombre de ressources utilisées dans l'architecture finale.

L'étape **d'ordonnancement** affecte une date d'exécution à chacune des opérations en tenant compte d'une part des dépendances de données et d'autre part des contraintes. Ainsi, l'ordonnancement peut chercher à :

- minimiser le nombre d'étape de contrôle en fonction d'une quantité de ressource *(List Scheduling...);*
- minimiser le nombre de ressources en fonctions d'un nombre d'étape de contrôle *(Force Directed Scheduling...);*
- minimiser le nombre de ressources et le nombre d'étape de contrôle *(Force Directed List Scheduling...).*

L'étape **d'assignation** associe à chaque opération un opérateur. Il est important de noter que l'ordre des étapes de synthèse peut varier selon les outils et les contraintes supportées. En effet, certaines techniques de synthèse effectuent l'ordonnancement puis l'assignation, d'autres l'assignation puis l'ordonnancement et dans certains cas l'ordonnancement et l'assignation de façon simultanée.

2.2. AMICAL

AMICAL [JER97] est un outil de synthèse orienté vers les applications dominées par le contrôle qui utilise comme informations initiales un algorithme et une bibliothèque de composants. La spécification de l'algorithme est décrite à l'aide d'un sous-ensemble comportemental du langage VHDL. La bibliothèque contient trois niveaux de description des composants : (1) une vue comportementale, décrite sous la forme d'un sous-programme, qui est utilisée pour la simulation, (2) une vue structurelle décrivant l'interface, le protocole de communication et un modèle d'estimation de performances et (3) une vue de la réalisation décrite au niveau RTL qui sera instanciée dans l'architecture finale. Les composants contenus dans cette bibliothèque peuvent être des opérateurs élémentaires (registres, additionneurs, multiplexeurs...), des unités fonctionnelles (ALU...) ou des composants complexes (co-processeur...) délivrés par un fournisseur d'IP ou résultant d'une synthèse antérieure. La Figure 2-12 décrit le flot de synthèse de l'outil AMICAL.

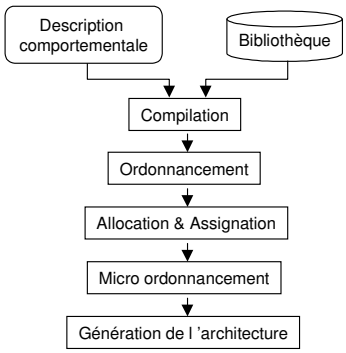


Figure 2-12 : Flot de synthèse de l'outil AMICAL

La phase initiale réalise la compilation de l'algorithme en une représentation interne sous la forme d'un graphe de flux de contrôle. L'étape d'ordonnancement produit une machine d'état comportementale qui représente l'ossature de la structure de contrôle de l'architecture. Les phases d'allocation et d'assignation sont ensuite exécutées et produisent une machine à états finis FSMC : FSM avec des coprocesseurs [JER97]. Une FMSC modélise une architecture hiérarchique faite d'un contrôleur principal et d'un ensemble de chemins de données qui peut lui-même inclure des FSMC. Le micro ordonnancement génère une machine d'état finis au niveau RTL. La génération de l'architecture alloue les connexions entre le contrôleur et les unités fonctionnelles et génère la description RTL finale. La Figure 2-13 illustre l'architecture modélisée par une FSMC.

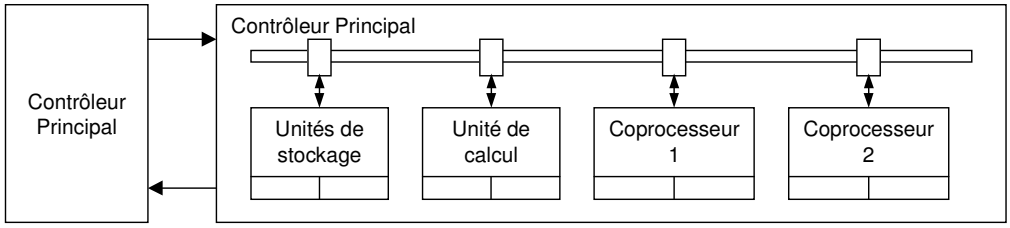


Figure 2-13 : Architecture modélisée par une FSMC

Bien que cet outil utilise les caractéristiques des entrées/sorties durant la synthèse, il ne supporte que l'expression des ordres d'acquisition et de production dans la spécification initiale de l'algorithme par une modélisation orientée machine d'états de type SFSMD.

2.3. GAUT

GAUT (acronyme de Générateur Automatique d'Unité de Traitement) est un environnement de synthèse d'architecture matérielle, dédié aux algorithmes de Traitement du Signal et de l'Image (TDSI) sous contrainte de cadence d'itération. A partir de la spécification d'un algorithme en VHDL il synthétise une description structurelle VHDL de niveau RTL optimisée en surface et destinée aux outils de synthèse logique du marché tels que ISE/Foundation de Xilinx ou Design Compiler de Synopsys. Cet outil universitaire résulte de travaux de recherche commencés au LASTI dans les années 1990 et poursuivis au LESTER depuis 1994 ([MAR93] et [PHI95]). Le développement informatique représente actuellement un volume de l'ordre de 200000 lignes de code C et JAVA.

2.3.1. Architecture des unités fonctionnelles

Le modèle cible des architectures synthétisées par *GAUT* est un cœur générique de processeur dédié au traitement du signal *DSP*. Ce modèle est composé de quatre unités fonctionnelles travaillant en parallèle (voir Figure 2-14) : unité de traitement *(UT),* unité de mémorisation *(UM),* unité de contrôle *(UC)* et unité de communication *(UCOM).*

Une horloge commune assure le fonctionnement synchrone entre les différentes unités qui fonctionnent en mode pipeline et s'échangent les données au travers d'un réseau multi-bus parallèles.

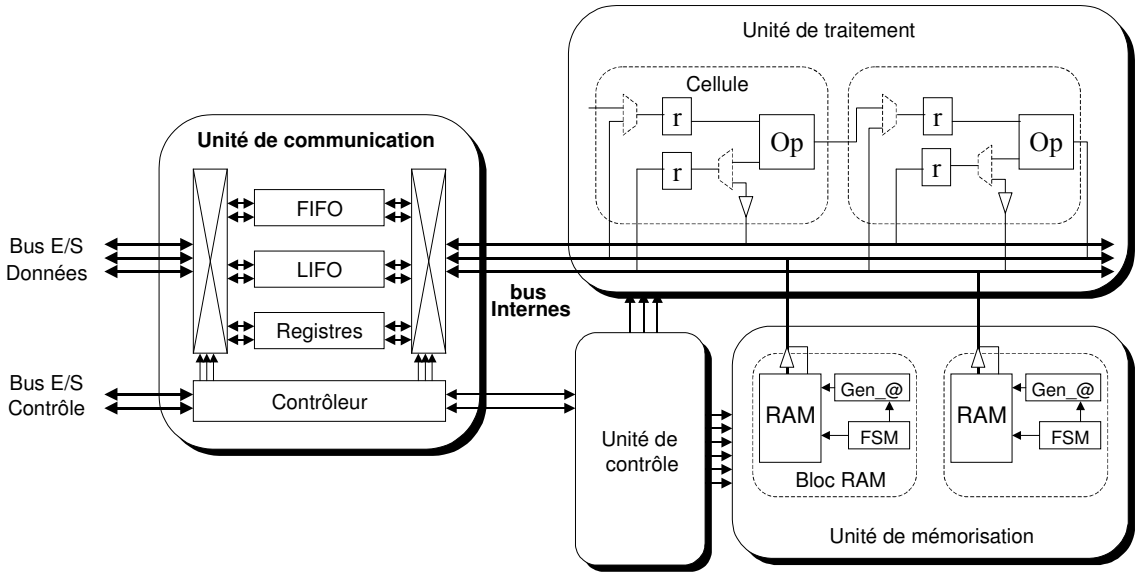


Figure 2-14 : Modèles architecturaux de l'outil *GAUT*

2.3.1.1 Unité de contrôle

L'unité de contrôle *(UC)* est constituée d'une machine d'états finis résultant de l'ordonnancement des opérations et d'un décodeur d'instructions qui se charge de générer les signaux de contrôle à destination des unités de mémorisation, de traitement ou de communication.

2.3.1.2 Unité de mémorisation

L'unité de mémorisation *(UM)* répond au problème de stockage des données [CHI97A], [CHI97B]. Elle est constituée de bancs mémoires *(RAM / ROM)*, de registres et de générateurs d'adresses. Un générateur est une machine d'états finis dont chaque état correspond à la lecture de la valeur d'une constante ou à la lecture/écriture de la valeur d'une variable en RAM. Une thèse est actuellement en cours sur la synthèse comportementale sous contraintes de ressources mémoires [COR03A], [COR03B]

2.3.1.3 Unité de Traitement

L'unité de traitement *(UT)* est en charge de la partie calcul de l'algorithme. Le modèle, sur lequel elle est basée, s'articule autour de cellules (arithmétiques ou logiques) élémentaires. Chaque cellule est composée d'un opérateur, d'un ensemble de registres, de multiplexeurs, de démultiplexeurs et de buffers trois états. Les registres permettent le stockage temporaire des données et la synchronisation des transferts de données au sein de *l'UT*. Les multiplexeurs, démultiplexeurs et buffers trois états ont en charge l'aiguillage des données. Leur présence optionnelle dans une cellule résulte du partage des registres et des opérateurs, réalisé durant la phase d'optimisation du matériel. Les cellules communiquent au travers d'un réseau de bus parallèles. Les bus dédiés assurent les échangent entre les cellules alors que les bus généraux permettent les accès aux mémoires contenues dans l'unité de mémorisation.

2.3.1.4 Unité de Communication

L'unité de communication *UCOM* est composée d'une partie opérative *(chemin de données)* et d'un contrôleur [BAG98]. Le chemin de données inclut des modules de stockage des données d'entrée/sortie *(FIFO, LIFO, registres)*, des bus (*internes et externes*) et des éléments d'interconnexion *(multiplexeurs, démultiplexeurs et barrières trois états)*. Les bus externes représentent les liens physiques entre le cœur d'IP et les composants du système. Les bus internes représentent les liens physiques avec de l'unité de traitement. Le nombre est définit par la synthèse et peut être supérieur ou égal au nombre maximum d'échanges simultanés de données entre l'*UCOM* et l'*UT*. Le contrôleur implémente d'une part le protocole de communication système et d'autre part synchronise l'unité de communication et l'unité de traitement.

Dans [BAG97] l'auteur présente le flot de conception de l'*UCOM* à partir des contraintes d'une unité de traitement, synthétisée sous contrainte de cadence, et des contraintes système dans un flot de conception matériel / logiciel. L'auteur démontre que dans certains cas, les contraintes imposées par l'unité de traitement et le système sont trop éloignées pour être adaptées par une interface rendant ainsi la synthèse de l'*UCOM* impossible. Bien qu'elle repose sur une modélisation des échanges entre le composant et un DSP, la méthodologie ne propose pas de

gestion explicite des points de synchronisation. Elle utilise de plus une unité de traitement synthétisée sous contrainte de cadence qui ne prend pas en compte les caractéristiques des séquences d'E/S.

2.3.2. Flot de conception

Le flot de conception mis en œuvre dans *GAUT* peut globalement être représenté par la Figure 2-15. La description algorithmique est décrite dans sous-ensemble du langage VHDL par un unique processus (couple entité / architecture). La description initiale est accompagnée d'une contrainte temporelle. Cette valeur, appelée *Tcadence*, correspond à la plage temporelle sur laquelle s'étale l'arrivée de l'ensemble des données nécessaire à une itération de l'algorithme. La phase de compilation effectue une analyse syntaxique, un contrôle sémantique et une parallélisation du code. Cette dernière étape réalise (1) la suppression des dépendances de contrôle (déroulage des boucles, mise en ligne des appels de procédures, parallélisation des branchements conditionnels) et (2) la suppression des fausses dépendances de données. En effet une distinction entre les vraies dépendances (producteur-consommateur) et les fausses dépendances que sont l'anti-dépendance (consommateur-producteur) et la dépendance de sortie (producteur-producteur) est faite dans l'outil *GAUT*. En raison de l'assignation unique la parallélisation du code se termine par un renommage des variables.

La compilation produit une représentation interne de l'algorithme sous la forme d'un graphe flot de signaux *(SFG)*. Cette modélisation permet l'expression du parallélisme maximal de l'algorithme. Après l'étape de synthèse de l'unité de traitement, qui sera décrite dans le quatrième chapitre, l'unité de contrôle et de mémorisation sont générées. Le flot de conception se termine par la synthèse de l'unité de communication qui est réalisée en tenant compte des contraintes d'intégration et des contraintes fournies par la synthèse de l'unité de traitement (voir chapitre 4). Afin de garantir la compatibilité avec le plus grand nombre d'outils de synthèse RTL du commerce, le VHDL RTL produit par *GAUT* est strictement conforme à la norme IEEE P1076 (Standard for VHDL Register Transfer Level Synthesis).

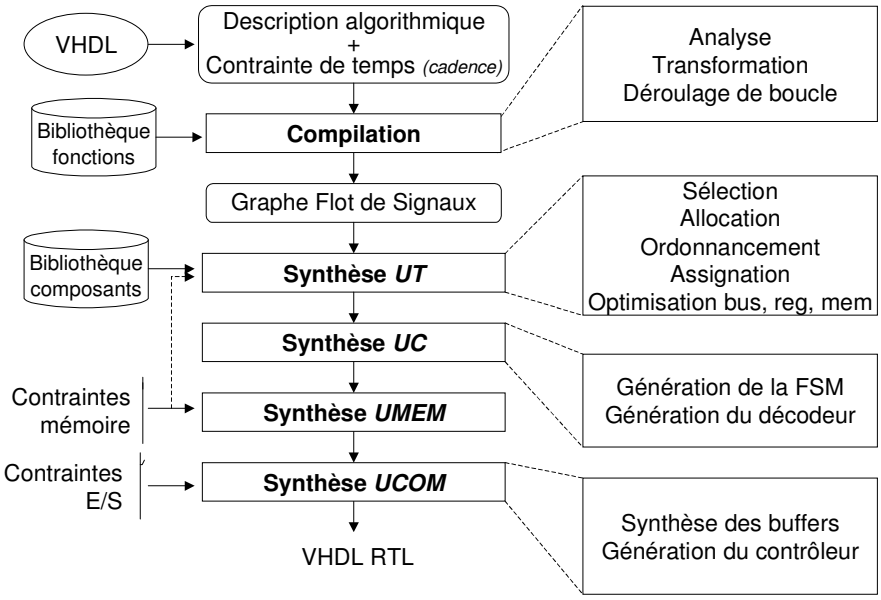


Figure 2-15: Flot de conception de l'environnement *GAUT*

2.4. Behavioral Compiler BC / Monet

2.4.1. Description succincte

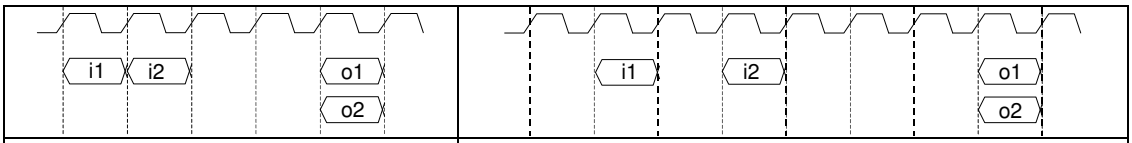
Behavioral Compiler de Synopsys et Monet de Mentor Graphics font partie des rares outils commerciaux de synthèse d'architecture ayant vu le jour. Bien que n'ayant pas de domaine d'application ciblé, ils reposent tous deux sur un flot de synthèse qui privilégie les algorithmes de calcul intensif.

La description comportementale initiale peut être indifféremment réalisée à l'aide des langages de description matériel VHDL ou Verilog. Elle est transformée par la phase de compilation, en une représentation interne de type CDFG. Le comportement de l'algorithme est décrit sous la forme d'un unique processus dans lequel les échanges de données avec l'extérieur sont réalisés au travers de ports. La synchronisation est spécifiée directement dans le code source au moyen d'instructions *wait* associées à la détection d'un front d'horloge.

BC comme MONET offrent trois modes d'ordonnancement qui se différentient par leur gestion du comportement aux entrées/sorties et/ou le nombre de cycle de contrôle qu'ils peuvent insérer entre deux états d'attente [KNA95].

1. Le mode *cycle-fixed* est le plus contraint et permet de maintenir un comportement pré et post synthèse identique en préservant l'ordre et les relations temporelles sur les opérations d'E/S. L'ordonnancement des calculs s'effectue donc dans les limites fixées par les contraintes sur les entrées/sorties qui, dans certains cas, peuvent rendre la synthèse impossible. Notons que ce mode, aussi supporté par l'outil CALLAS [STO92], est orienté vers la synthèse d'applications qui utilisent des protocoles de communications ayant des contraintes temporelles précises au cycle près.

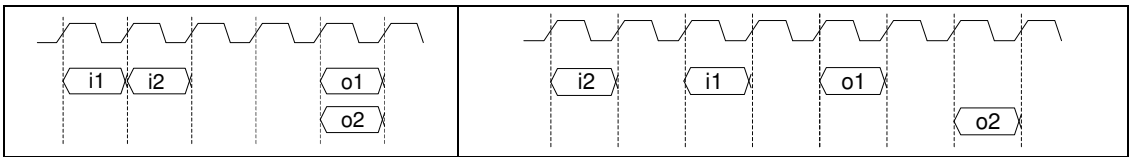
2. Le mode *Superstate-fixed*, à la différence de son prédécesseur, conserve uniquement une relation d'ordre d'acquisition et de production des E/S. Dans ce contexte, deux instructions d'attente *wait* adjacentes sont considérées comme les bornes d'un état spécifique nommé *super-état*. Les opérations contenues dans un super état ne sont plus ordonnancées dans un nombre de cycle fixé par la spécification puisque l'ordonnancement s'autorise un décalage temporel sur les entrées/sorties en étendant la durée d'un super-état par ajout de cycles d'horloge (voir Figure 2-16). Les écritures des sorties sont réalisées simultanément sur le dernier front d'horloge, ce qui est une restriction forte. Ainsi, le mode *Superstate-fixed* synthétise une architecture dont le comportement aux E/S est différent de celui de la description initiale. Afin de garantir par simulation un fonctionnement correct post synthèse, ce mode requiert de la part des concepteurs l'utilisation de protocole de type poignée de main (asynchrone). Notons que le concept de supers-états est analogue à celui des états des BFSM (Behavioral FSM) de l'outil PUBSS [WOL92], HLFSM (High Level FSM) [BER92] et SFSMD (Super State FSM with Datapath) [LEH99] proposé par D. Gajski.



(a) Comportement Pré synthèse | (b) Comportement Post synthèse

Figure 2-16 : Le mode Super-état

3. Le mode *free-floating* ignore les instructions de synchronisation *wait* ce qui résulte en un ordonnancement libre des entrées/sorties. Ainsi, tout en respectant les dépendances de données l'outil permute certaines opérations de lecture ou d'écriture sur les ports (voir Figure 2-17) afin d'optimiser l'architecture en fonction d'un critère de coût spécifié par le concepteur. Ce mode, qui synthétise une architecture dont le comportement est différent de celui de la description initiale, est orienté vers l'exploration de l'espace de conception. Notons que SystemC Compiler, l'extension de l'outil BC, ne propose plus ce mode d'ordonnancement.



(a) Comportement Pré synthèse | (b) Comportement Post synthèse

Figure 2-17 : Le mode Free-Floating

Le concepteur peut utiliser, en plus du mode d'ordonnancement, un ensemble de contraintes temporelles spécifiées à l'aide de directives de synthèse dans un script. Ces contraintes spécifient des délais relatifs minimum et maximum entre deux instructions marquées explicitement dans le code à l'aide de *tags* [LY95].

2.4.2. Restrictions

Une première limitation provient des règles d'écriture imposées, en fonction du mode d'ordonnancement, par cet outil. En effet, les trois modes d'ordonnancement associent des sémantiques différentes aux instructions de synchronisation en fonction des structures de contrôle dans lesquelles elles se trouvent (boucle, instruction conditionnelle). Ainsi, lorsque le concepteur veut synthétiser un composant dans un mode différent de celui pour lequel il a été conçu, il doit réécrire partiellement le code source.

La dissociation de la communication et du traitement n'est supportée que très partiellement puisque les instructions de synchronisation sont présentes dans le code source. Cet inconvénient peut être contourné en décrivant le protocole de communication sous forme de sous-programme. Il est ainsi possible de fournir différents modes d'échange de données en rédigeant différentes bibliothèques de sous programmes. Les styles d'écriture imposée par BC pour l'utilisation de sous programme de communication rendent cependant cette solution difficile à utiliser [MEL99].

La troisième restriction majeure concerne le non-déterminisme. En effet, bien qu'il fournisse un mécanisme de spécification de contraintes de type minimum et maximum, BC ne supporte pas l'expression de fenêtre temporelle sur les dates d'arrivées des entrées. De plus, BC bien que permettant l'expression de lecture non-déterministe sur un port d'entrée, à l'aide de boucle *while*, il ordonnance statiquement le CDFG relativement à cette opération. De ce fait, l'utilisation des contraintes relatives n'est possible que pour l'ordonnancement des opérations de traitements. La spécification de comportement parallèle, au sein d'un même processus est de ce fait, assez limitée.

Enfin, la création d'une architecture pipeline demande au concepteur de spécifier le nombre de d'étages qu'il désire à l'aide de directives de synthèse là où des outils comme *GAUT* sont capables de le déterminer seul en fonction de la contrainte de cadence et du chemin critique.

2.5. HERCULES/HEBE

Bien que la littérature référence de nombreux travaux dans le domaine de la synthèse comportementale, il n'en existe que très peu concernant la synthèse ou l'ordonnancement sous contraintes temporelles réparties sur les entrées/sorties [NES93][LY95][STO92]. Le nombre d'outils et de méthodologies prenant en considération les contraintes non déterministes et encore plus restreint. A notre connaissance, seuls les travaux de Ku et De Micheli, que nous présentons dans cette section, traitent de ce problème spécifique pour la synthèse (VULCAN [GUP91][GUP95] réalise en effet uniquement le partitionnement dans OLYMPUS [DeM90] avant la synthèse comportementale des parties matérielles). Ainsi, dans un premier temps nous décrivons succinctement l'environnement OLYMPUS dans lequel s'intègrent les outils HERCULE/HEBE. Nous détaillons ensuite le formalisme que les auteurs utilisent pour modéliser l'algorithme et les contraintes temporelles et présentons la méthode d'analyse permettant de garantir l'existence d'une solution au problème d'ordonnancement en présence d'opérations ayant des durées d'exécution non bornées.

2.5.1. Présentation générale

L'environnement de conception de circuits électroniques OLYMPUS (voir Figure 2-18) comprend trois principales tâches : la synthèse comportementale, la synthèse logique et l'implémentation physique. Cet environnement s'oriente vers la synthèse sous contraintes temporelles strictes d'ASIC ayant des protocoles de poignée de main complexes. Les architectures ciblées sont synchrones non pipelines et n'utilisent pas de logique synchrone multi-phase. Dans ce contexte, la synthèse comportementale est réalisée par les outils HERCULE et HEBE. CERES est quant à lui dédié à la synthèse logique. Les spécifications comportementale et structurelle peuvent être respectivement simulées à l'aide des outils ARIANE et MERCURY. La correspondance entre les descriptions pré et post synthèse de l'architecture est prouvée formellement à l'aide de THESEUS.

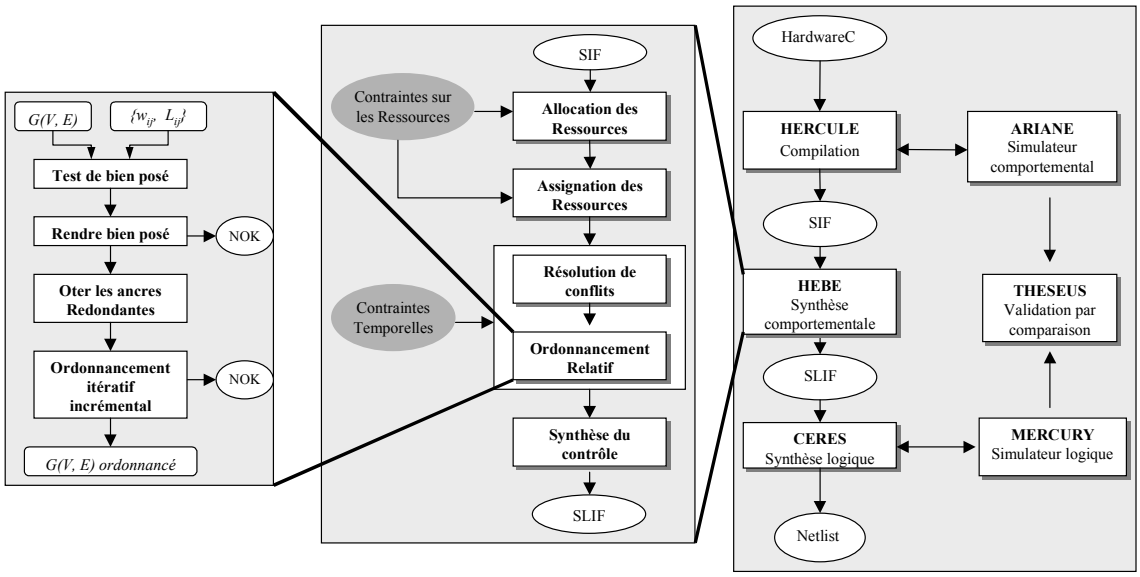


Figure 2-18 : L'environnement de conception OLYMPUS

La description initiale est réalisée à l'aide du langage HardwareC [KU90] dont les principales caractéristiques sont les suivantes :

- *Sémantique procédurale et déclarative* : une architecture peut être décrite comme une séquence d'opérations et/ou comme une interconnexion de composants.

- *Communication entre processus* : HardwareC modélise un composant comme un ensemble de processus concurrents qui communiquent par passages de messages synchrones (send(canal, mess) / receive(canal, mess)) ou par passages de ports (id. VHDL).

- *Appel lié et non lié* : les appels liés font explicitement référence à une ressource. Pour les appels non liés c'est durant la synthèse que la ressource est choisie afin d'exécuter cette fonctionnalité. Le partage de ressource peut donc être spécifié par le concepteur.

- *Modèle* (*Templates models*) : les modèles sont au nombre de quatre.

  - *Bloc* : l'abstraction de plus haut niveau qui décrit l'assemblage structurel et physique des composants. Il a une sémantique déclarative qui consiste en une interconnexion d'instances d'autres blocs et de processus. C'est l'équivalent d'une entité en VHDL.

  - *Processus* : c'est une hiérarchie de procédures et de fonctions qui se répètent de façon infinie.

  - *Procédure* : c'est une encapsulation d'opérations. Une procédure peut appeler d'autres procédures et fonctions (la récursivité n'est pas supportée). Elle ne s'exécute que sur appel.

  - *Fonction* : une fonction est équivalente à une procédure à la seule différence qu'elle retourne une valeur (id. Langage C).

- *Plusieurs degrés de parallélisme* : le degré de parallélisme peut être ajusté en utilisant des groupements d'opérations :

  - *Sequential* (*[]*) : permet de spécifier la sérialisation de deux instructions même s'il n'y a pas de dépendances de données

  - *Data-parallel* (*{}*) : deux instructions se trouvant entre {} peuvent être exécutées en parallèle si elles n'ont pas de dépendance sur les données (*data-independent*)

  - *Parallel* (<>) : à la différence du *data-parallel* les instructions sont réalisées en parallèle même en présence de dépendances de données.

- *Spécification de contraintes*

  - Les *contraintes de temps* sont spécifiées sous forme de durées minimum ou maximum entre deux "instructions", et d'une durée pour une instruction particulière (voir Figure 2-19). Pour identifier les instructions, auxquelles ces contraintes font référence, des *tags* (voir Figure 2-20) sont associés aux "instructions" comme dans Behavioral Compiler.

constraint mintime from *tag-src* **to** *tag-dst* = *num* **cyclesunits**;
constraint maxtime from *tag-src* **to** *tag-dst* = *num* **cyclesunits**;
constraint delay of *tag-src* = *num* **cyclesunits**;

Figure 2-19 : Définition des contraintes de temps

```
tag label1, label2, labelvec[3];
labelvec[0]: send( channelA, msg );
labelvec[1]: write port = value;
label1: label2: y = read(x);
```

Figure 2-20 : déclaration et utilisations de tags

  - Les *contraintes sur les ressources* permettent de limiter le nombre d'opérateurs et de contraindre leur utilisation par des opérations en vue de l'implémentation finale. Ceci est réalisé par la déclaration du nombre de ressources utilisables dans la future architecture (voir Figure 2-21).

constraint resource usage *modelname num*;
constraint resource usage *templatename* **with** (*int-arguments*) *num*;

Figure 2-21 : Déclaration des contraintes sur ressources

2.5.2. Synthèse haut niveau

HERCULE, bien que présenté par les auteurs comme un *synthétiseur comportemental,* réalise la phase de compilation. Il identifie le parallélisme maximal par un ensemble de transformations automatiques : déroulage de boucle, propagation des constantes et des variables, résolution des assignations multiples et conditionnelles de variables, élimination des opérations redondantes,

élimination du code mort, fusion des branches conditionnelles ne contenant que des opérations logiques et identification des dépendances de contrôle et de données. La description initiale HardwareC est ainsi transformée en un format intermédiaire *SIF (Sequencing Intermediate Format)* sous la forme d'un graphe de séquencement *G(V, E)* des opérations définit comme suit

Un graphe de séquencement *G(V,E)* est un graphe pondéré polaire orienté où l'ensemble de nœuds $\{V\}$ représente des opérations avec $|V| = n+1$ nœuds où v_0 et v_n dénotent respectivement la source et le puits, l'ensemble d'arcs $\{E\}$ représente les dépendances.

Cette représentation intermédiaire est utilisée par l'outil HEBE qui réalise la synthèse comportementale *(synthèse structurelle pour les auteurs)*. Son rôle est d'obtenir une structure satisfaisant les contraintes matérielles et temporelles. Pour cela, chaque *modèle* est traité comme une ressource qui peut être allouée et partagée parmi les appels de modèles (procédure ou fonction) et chaque appel est implanté comme l'activation particulière d'une ressource.

Le flot de synthèse débute par les étapes d'allocation et d'assignation des ressources. Le graphe de séquencement *SIF* est pondéré durant ces deux étapes. Pour cela des temps d'exécution sont ajoutés aux nœuds du graphe :

un poids W_{ij} associé à l'arc (v_i, v_j) est égal au temps d'exécution du nœud v_i noté $\boldsymbol{\delta(v_i)}$.

Afin de prendre en compte les contraintes temporelles les auteurs définissent un graphe de contraintes dans lequel :

une contrainte temporelle minimum $l_{ij} > 0$ nécessite $\sigma(v_j) \geq \sigma(v_i) + l_{ij}$ et une contrainte temporelle maximum $u_{ij} > 0$ nécessite $\sigma(v_j) \leq \sigma(v_i) + u_{ij}$ où $\sigma(v_i)$ représente la date d'exécution de l'opération v_i relativement à $\sigma(v_0)$.

La transformation (voir exemple Figure 2-22), du graphe de séquencement en un graphe de contraintes, est réalisée en :

1. insérant un *Arc avant* de poids w_{ij} positif de valeur l_{ij} pour une contrainte minimum;
2. insérant un *Arc arrière* de poids w_{ij} négatif de valeur $-u_{ij}$ pour une contrainte maximum.

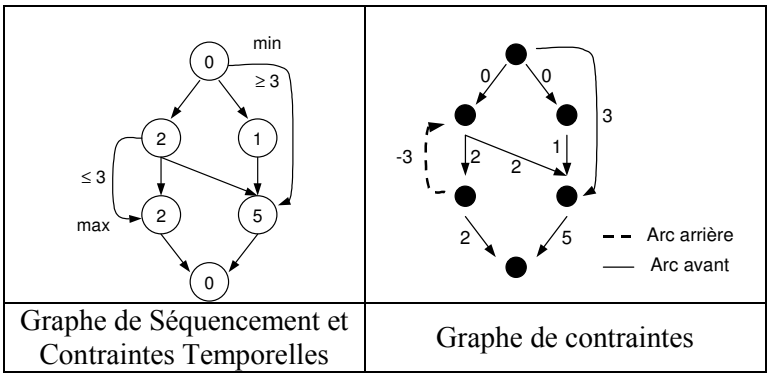


Figure 2-22 : Exemple de transformation de graphe

Le graphe de contraintes résultant *G(V, E)* est constitué d'un ensemble d'arcs avants et arrières : $E = E_f \cup E_b$. Le sous graphe $G_f(V, E_f)$, supposé acyclique, contient uniquement les arcs avants. Le sous graphe $G_b(V, E_b)$ ne contient quant à lui que les arcs arrières.

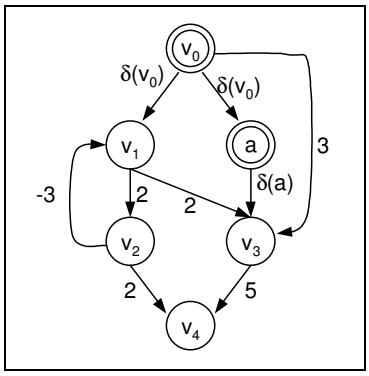
Afin de prendre en compte la nature non déterministe de certaines opérations, les auteurs introduisent la notion d'*ancre* et d'*ensemble d'ancre* (noté *A*) comme étants des nœuds dont les temps d'exécution sont non bornés. Ils définissent *l'ensemble des ancres d'un nœud* v_j et d'*offset* comme suit

L'*ensemble des ancres* d'un nœud v_i est le sous ensemble des ancres $A(v_i) \subseteq A$, tel que $a \in A(v_i)$ s'il existe un chemin dans $G_f(V, E_f)$ de *a* vers v_i contenant au moins un arc dont le poids non borné est égal à $\delta(a)$.

L'*offset d'un nœud* $v_j \in V_a$ par rapport à l'ancre *a* est une valeur entière $\sigma_a(v_j)$ telle que $\sigma_a(vj) \geq \sigma_a(vi) + wij$ s'il existe un arc de poids w_{ij} allant du nœud v_i au nœud v_j dans le sous graphe $G_a(V_a, E_a)$, et $\sigma_a(a)$ est normalisé à 0.

Si $\sigma_a(v_i)$ est la valeur minimum, alors c'est l'offset minimum de v_i relativement à *a* et est noté $\sigma_a^{min}(v_i)$

Ainsi, une ancre *a* est dans l'ensemble des ancres d'un nœud si ce nœud ne peut s'exécuter qu'après la complétion de *a*. Le nœud v_0 est toujours considéré comme une ancre. La Figure 2-23 décrit un graphe de contraintes contenant deux ancres v_0 et *a* et les *offsets* associés à chacun des nœuds en fonction de leur *ensemble d'ancres*.



Nœud v_i	Ensemble d'*Ancre* $A(v_i)$	*Offset*	
		$\sigma(v_i)$	$\sigma(v_0)$
v_0	∅	-	-
a	$\{v_0\}$	0	-
v_1	$\{v_0\}$	0	-
v_2	$\{v_0\}$	2	-
v_3	$\{v_0, a\}$	3	0
v_4	$\{v_0, a\}$	8	5

Figure 2-23 : Graphe de contrainte, Ancres et Offset associés

La phase suivante résout les conflits [KU91B] dus à l'assignation des ressources. Pour cela, elle insère des arcs de séquencement entre deux nœuds opérations assignés à un même opérateur pour qu'ils ne puissent s'exécuter en parallèle et ce de telle sorte que les contraintes temporelles fixées soient toujours respectées. Cette étape réalise donc un premier ordonnancement.

L'ordonnancement relatif [KU92] est composé de quatre étapes.

1. Le *test des contraintes* vérifie la cohérence *(bien posé)* selon les théorèmes suivants :

   Théorème 1 : Un graphe de contrainte *G(V, E)* est *faisable* si et seulement si aucun cycle positif n'existe dans G, en supposant les valeurs des délais non bornés fixées à 0.

Théorème 2 : Soit $G(V, E_f)$ acyclique. Un graphe de contrainte faisable G(V, E) est *bien posé si et seulement si* $A(v_j) \subseteq A(v_i)$ *pour tous les arcs* $e_{ij} \in E$.

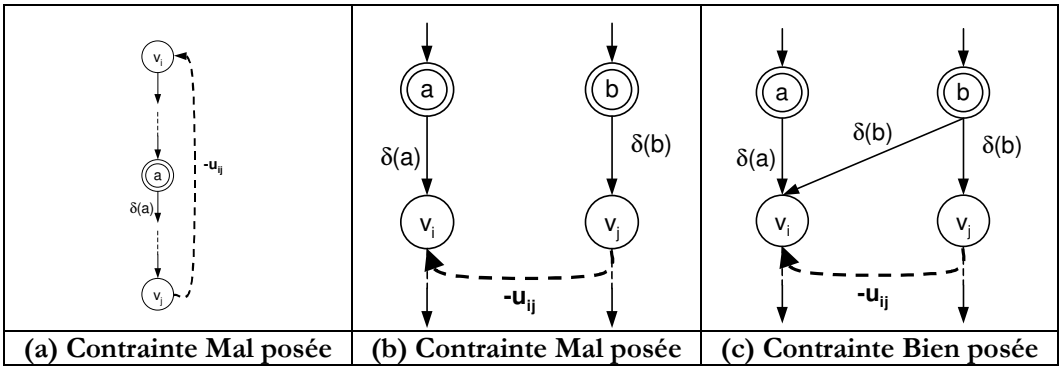


Figure 2-24 : Exemple de contraintes (a) et (b) mal posées et (c) bien posé

2. La *transformation* tente de rendre un graphe mal posé en graphe bien posé (voir Figure 2-24 (b) et (c)) en ajoutant des arcs de séquencement.

3. La suppression des ancres *redondantes* permet d'une part d'augmenter l'efficacité de l'algorithme d'ordonnancement en diminuant le nombre de nœuds à vérifier et d'autre part de réduire la complexité du contrôle afin d'améliorer les performances de l'architecture. La *redondance* est basée sur la propriété de *pertinence* qui permet de définir un lien direct entre une ancre *a* et un nœud v_i. Ces deux notions sont définies comme suit :

   Définition : Un chemin « *définissant* » d'une ancre $a \in A$ vers un nœud v_i, *dénoté* $\rho(a, v_i)$, est un chemin de *a* vers v_i dans $G(V, E)$ tel qu'il contient exactement un arc ayant un poids non borné (égal à $\delta(a)$). La longueur de $\rho(a, v_i)$ dénotée par $|\rho(a, v_i)|$, est la somme des poids des arcs du chemin excluant la valeur non bornée $\delta(a)$.

   Un chemin *maximal définissant* d'une ancre pertinente $r \in R(v_i)$ d'un nœud v_i est le plus long chemin définissant $\rho^*(r, v_i)$ parmi les chemins définissants entre *r* et v_i.

Remarque : Un chemin *définissant* ne contient qu'une seule et unique ancre : sa source.

   Définition : L'ensemble des ancres *pertinentes* d'un nœud v_i d'un graphe de contrainte $G(V, E)$ est l'ensemble des ancres $R(v_i) = \{r \mid r \in A$ *tel qu'il existe un chemin définissant* $\rho(r, v_i)\}$

Remarque : La date d'exécution relative à la date de complétion de chaque ancre (offset) est bornée. De plus la définition de l'ensemble des *ancres pertinentes* considère **tous** les chemins dans le graphe complet G contrairement à la définition de l'ensemble des ancres qui considère **uniquement** les chemins dans le graphe de contraintes avant G_f (voir Figure 2-25 (a)).

Ainsi, une ancre pertinente peut directement affecter l'activation d'un nœud. Cependant des redondances entre les ancres pertinentes peuvent exister comme le montre la Figure 2-25 (b) où le nœud v_i ne pourra s'exécuter qu'après complétion de *b*.

Définition : Une ancre $r \in A(v_i)$ d'un nœud v_i est redondante s'il existe une ancre q telle que :

1. $r \in A(q)$ et $q \in A(v_i)$ et
2. $\rho^*(r, v_i) < \rho^*(r, q) + \rho^*(q, v_i)$

Autrement, r est une ancre non redondante de v_i.
L'ensemble des ancres non redondantes de v_i est dénoté par $IR(v_i)$.

La Figure 2-25 (c) donne un exemple d'ancre redondante.

4. L'ordonnancement relatif du graphe de contraintes incrémente itérativement les offset jusqu'à ce que les contraintes minimum soit vérifiées. Les offsets sont ensuite ajustés pour respecter les contraintes de temps maximum. Ces deux opérations sont répétées jusqu'à l'obtention d'un ordonnancement respectant l'ensemble des contraintes. La méthode d'ordonnancement utilisée étend l'algorithme de compaction de *layout* présentée dans [LIA83] pour supporter les solutions vectorielles et trouve une solution minimum ou détecte la présence de contraintes inconsistantes en un temps polynomial ($|E_b + 1|$ itérations).

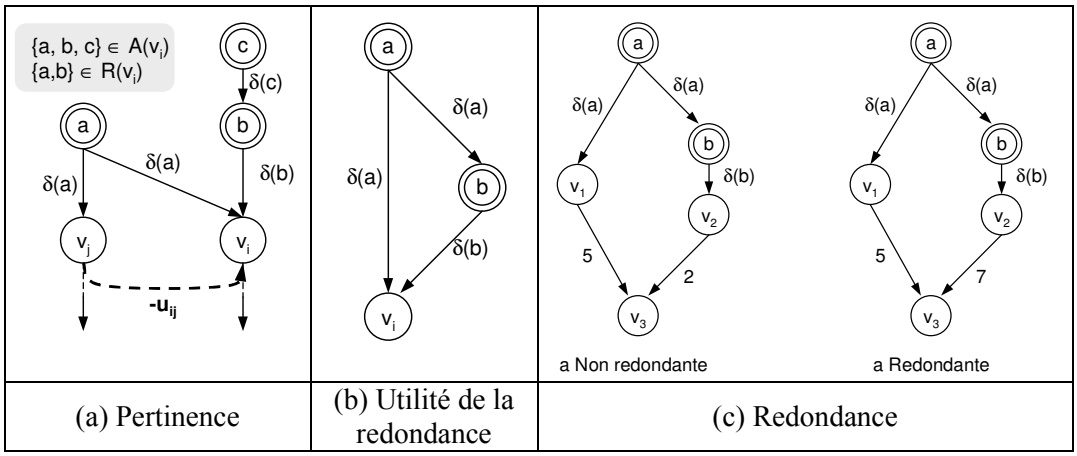


Figure 2-25 : Pertinence et redondance

Les phases d'allocation et d'assignation étants réalisées avant l'ordonnancement, la synthèse se résume donc à la génération de la partie contrôle. La technique utilisée est une extension de la de la synthèse de contrôle adaptatif présenté dans [KU91A]. Cette étape produit une interconnexion modulaire de FSMs qui communiquent et se synchronisent par un ensemble de signaux de poignée de main. La complétion d'une ancre est signalée par un signal «done» qui permet la génération d'un signal «enable» pour chaque nœud $v \in V$. Le signal «enable» initie l'exécution de chaque nœud v. Dans ce contexte, chaque ressource matérielle est pilotée par une FSM. Afin de réduire le coût de la logique de contrôle, une optimisation, basée sur la re-synchronisation des opérations, est proposée dans [KU91C]. Cependant cette dernière limite fortement le parallélisme de l'application et ne donne pas toujours une solution acceptable [FIL93].

L'architecture résultant de la synthèse architecturale est décrite comme une interconnexion structurelle d'éléments logiques à l'aide d'une représentation *SLIF (Structural / Logic Intermediate Format)*.

2.5.3. Bilan

Les auteurs considèrent les temps d'exécution non déterministes comme non bornés et réalisent l'ordonnancement a posteriori des phases d'allocation et d'assignation. La prise en compte de contraintes temporelles non bornées permet de vérifier la cohérence d'un graphe de contraintes en analysant les propriétés d'inclusion des ensembles d'ancres *(Théorème 2)*. Dans ces conditions, une solution au problème d'ordonnancement est obtenue en insérant des arcs de sérialisation qui permettent le séquencement des entrées/sorties et des opérations. Cependant, la technique utilisée n'est plus valide dans le cas des contraintes temporelles variables bornées que nous considérons dans notre approche.

3. Conclusion

Dans ce chapitre nous avons présenté, dans un premier temps, différents travaux portant sur l'intégration de composant virtuel de niveau RTL par l'utilisation de wrapper ou module d'interface contenant des FIFO. Les différentes méthodologies s'appuient souvent sur une modélisation des bibliothèques de composants permettant dans le cadre de la simulation puis de la synthèse une intégration à gros grains des Ips. En effet, les composants virtuels RTL utilisés sont supposés avoir des séquences d'acquisition et de production, identiques à celles des composants avec lesquels ils communiquent. Dans ce contexte, les contraintes d'adaptation considérées relèvent du protocole et de l'adaptation de débit.

Dans les applications de TDSI, bien que les données manipulées soient des structures régulières adaptées aux modules de type FIFO (ou LIFO), l'adaptation des séquences des E/S système aux séquences du composant requiert aussi l'utilisation d'autres structures mémorisantes telles que les registres. En effet, les données susceptibles d'être stockées dans une FIFO doivent

- avoir une même direction *entrée* ou *sortie*
- avoir des dates de transferts différentes
- avoir des intervalles de durée de vie disjoint dans l'*UCOM*
- respecter les cohérences d'écriture et de lecture : La séquence d'entrée dans une FIFO doit être la même que la séquence de sortie

Nous avons choisi dans nos travaux de nous concentrer sur la conception adéquate des composants en vue de leur intégration. Ainsi, la solution que nous proposons ne s'oppose pas à l'utilisation de modules d'interfaces incluant des FIFO et des protocoles génériques, mais s'inscrit plutôt dans une démarche complémentaire. En effet, la prise en compte des contraintes d'intégration lors de la synthèse d'un composant virtuel permet la conception de l'unité de traitement et de communication dédiée au système dans lequel l'IP doit être utilisé. La méthodologie proposée permettra à terme une intégration optimisée en terme de surface, latence et consommation.

Ainsi, la modélisation des contraintes, en plus du débit de l'application (période d'itération) et des caractéristiques technologiques doit aussi, pour piloter efficacement la synthèse du composant virtuel, être capable d'exprimer les contraintes temporelles d'intégration provenant des étapes de raffinement de la communication qui définissent

- **les protocoles** : Bloquant/Non bloquant, Maître/Esclave, Protocole spécifique
- **les contraintes temporelles** : temps de transfert, temps de propagation dans le média de communication, dates ou fenêtres temporelles
- **les modes de transferts** : mémoire partagée, DMA, transfert direct
- **la topologie de l'architecture de communication** : bus partagé, point-à-point
- **le mode d'arbitrage** : TDMA, priorités
- **les formats et types de bus** (standard, périphérique)
- **les tailles des données et les largeurs de bus**
- **les caractéristiques des transferts** : taille de *"burst"*, DMA
- **Les séquences de transferts**
- **Les contraintes temporelles relatives de type minimum et maximum**
- **Les dates et les échéances des échanges**

L'indéterminisme sur les dates d'arrivées des données que nous considérons sur les ports de l'IP peut provenir :

- des calculs réalisés par les autres composants du système *(boucle dynamique, branchement)*
- du temps de transfert, des protocoles, des attentes de bus

Remarque : Dans ce manuscrit nous utilisons la notion de non-déterminisme au sens d'une variation temporelle possible de l'exécution d'une tache. Cette notion diffère donc de celle utilisée dans le domaine des applications orientées contrôle dans lequel, le non-déterminisme d'un système est la possibilité, à partir d'un même état et d'un même événement d'aller dans plusieurs (au moins deux) états différents.

Le chapitre suivant présente :

- une modélisation formelle permettant la spécification des contraintes d'intégration
- un ensemble de techniques permettant d'étudier l'existence d'une solution au regard des contraintes d'intégration et des contraintes algorithmiques du composan

Chapitre 3

Synthèse Conjointe SoC / IP

Dans ce chapitre nous proposons une approche pour la synthèse comportementale sous contraintes d'Entrées/Sorties. Cette approche repose sur la modélisation formelle des contraintes d'intégration et des contraintes algorithmiques à base de graphes. Nous présentons ensuite deux phases d'analyse permettant de garantir si les contraintes de conception ont une solution d'ordonnancement.

1. Approche de conception

Dans notre approche, nous proposons l'utilisation de *composant virtuel comportemental*, décrit sous une forme algorithmique dans un langage de haut niveau (*VHDL*, *SystemC*, etc.) et autorisant un haut degré de flexibilité par le biais d'un jeu de paramètres génériques [RNR02]. L'architecture d'un *IP comportemental* repose sur les unités fonctionnelles suivantes : unité de traitement (*UT*), unité de mémorisation (*UM*), unité de contrôle (*UC*) et unité de communication (*UCOM*). Dans le cadre du projet *RNRT ALIPTA* [RNR03], nous proposons de générer un cœur d'IP RTL en synthétisant sous contraintes d'intégration sa description comportementale. La description des protocoles d'entrée/sortie ainsi que des caractéristiques des séquences de données, échangées entre l'IP et les composants du système, sont spécifiés par l'intégrateur. Une thèse est actuellement en cours sur l'extraction des contraintes à partir du raffinement des communications dans l'environnement Cofluent/MCSE. Le flot de conception de l'IP s'appuie sur un ensemble de transformations successives qui raffinent une description comportementale en une description structurelle. Dans ce contexte, l'IP est délivré par le fournisseur de composant virtuel sous la forme d'un algorithme synthétisable. L'intégrateur introduit les contraintes systèmes lors de la synthèse de ce composant.

L'approche de conception d'un composant virtuel que nous proposons repose sur trois étapes que sont la synthèse sous contraintes de l'unité de traitement, la synthèse sous contraintes de l'unité de communication et la synthèse sous contraintes de l'unité mémoire. Seuls dans ce document les deux premiers points sont abordés, la synthèse mémoire étant traitée dans la thèse, actuellement en cours, de Gwénolé Corre [COR03].

1.1. Flot de conception sous contrainte d'un IP

La spécification de l'IP est modélisée par un Graphe Flot de Signaux *(SFG)*. Un graphe de contrainte algorithmique *ACG* est ensuite généré en pondérant les arcs du *SFG* par les temps de propagation des opérateurs et la cadence d'itération. Une première phase d'analyse *(faisabilité)* permet de vérifier la cohérence entre la cadence, les dépendances de données de l'algorithme et les contraintes technologiques. Pour supporter les architectures spécifiques des applications Traitement du Signal et de l'Image (TDSI), nous avons étendu la modélisation de type *SIF (Sequential Intermediate format [KU92])*. Ce modèle formel nommé *IOCG (IO Constraint Graph)* permet l'expression des contraintes d'intégration pour chacun des bus (ports) connectants l'IP aux autres composants du système. Il supporte ainsi, la modélisation (1) des modes d'échanges de données, (2) du type de transferts, (3) de l'indéterminisme dans les dates d'arrivées des données dû aux phases d'arbitrage pour l'accès au bus partagé, (4) d'un séquencement des données échangées. Il permet en outre l'expression de mécanismes liés aux protocoles (synchrone / asynchrone) et les latences introduites par les conversions de format de données. Les contraintes d'intégration et les contraintes algorithmiques de l'IP sont ensuite fusionnées pour fournir un graphe détaillé des contraintes *GCG (Global Constraint Graph)* qui permet (1) d'analyser la pertinence des contraintes d'intégrations système vis à vis de

l'algorithme *(bien posé)* et (2) de détecter les nœuds de synchronisation entre l'environnement et le composant.

Basée sur les résultats de cette analyse, la synthèse de l'unité de traitement est réalisée à l'aide de l'outil GAUT (Générateur Automatique d'Unité de Traitement) dont le cœur d'ordonnancement a été modifié afin de prendre en compte les contraintes temporelles spécifiées par l'intégrateur du composant virtuel. Cette unité de traitement représente le chemin de données de l'IP et inclut des opérateurs de calcul (additionneurs, multiplexeurs,...). Le pilotage de chaque unité fonctionnelle est réalisé par une unité de contrôle, symboliquement représentée par une FSM. La synthèse de l'UT génère un ensemble de contraintes aux E/S modélisé sous la forme d'un *IPERM (IP Execution Requirement Model [COU02D])*. Ces contraintes représentent (1) les dates de production et de consommation des données dans l'UT et (2) les bus associés aux transferts de ces données entre l'UCOM et l'UT.

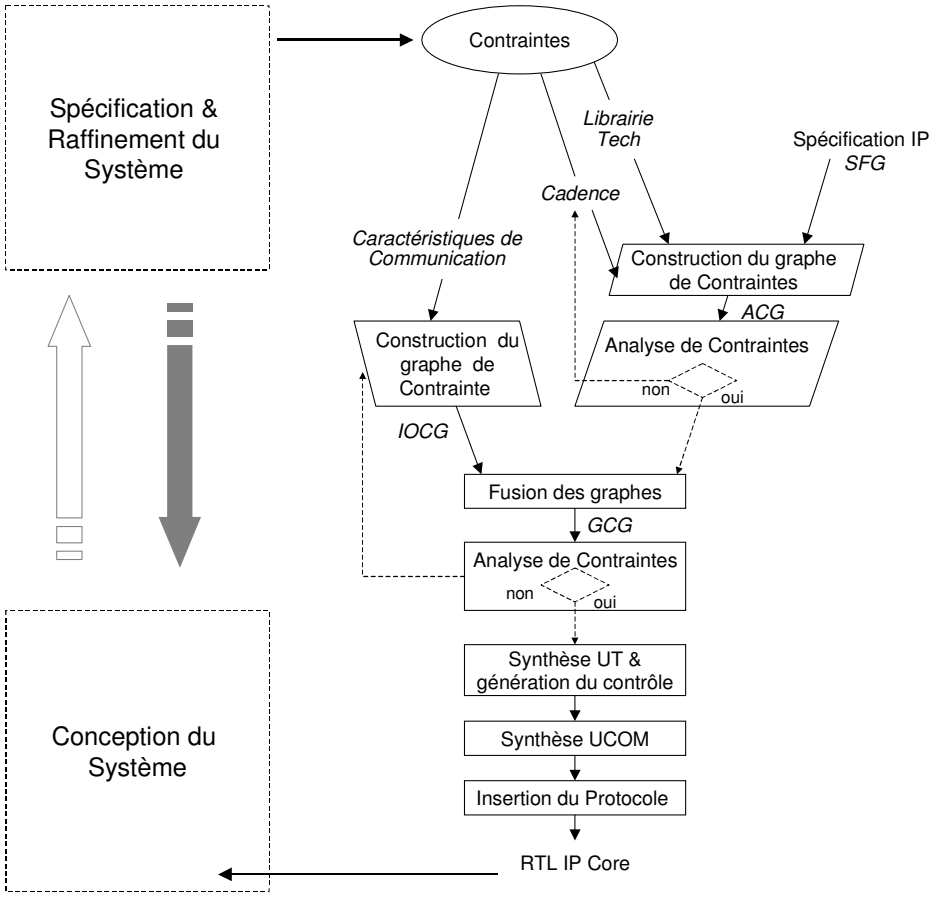


Figure 3-1 : Approche de conception

L'unité de communication est composée d'éléments mémorisants assurant la buffurisation des données et d'un ensemble de FSMs : une FSM par port, chacune implémentant le protocole d'échange de données spécifié par le concepteur du SoC et une FSM réalisant la synchronisation entre l'*UT* et l'*UCOM*. Seul le problème de modélisation des contraintes pour le dimensionnement des buffers sera traité dans ce document : les travaux concernant le protocole et les machines d'états ne seront pas abordés. Cette unité est synthétisée sous contrainte et a

posteriori de la synthèse de l'UT. Pour cela les modèles *IPERM* et *IOCG* sont utilisés afin de dimensionner et d'optimiser d'une part les éléments mémorisants réalisant l'adaptation temporelle des séquences d'E/S système aux séquences de l'UT et d'autre part de générer les multiplexeurs et démultiplexeurs aiguillant les E/S entre les bus internes, les bus système et les éléments mémorisants.

Le découpage en plusieurs unités fonctionnelles sur lequel nous basons l'architecture de nos composants virtuels permet la séparation de la communication et du traitement [ROW97]. La technique de synthèse utilisée, couplée au modèle d'architecture, garantie une intégration optimisée du composant virtuel puisque l'unité de traitement et l'unité de communication sont adaptées aux contraintes d'entrées/sorties imposées par le système.

1.2. Illustration d'un transfert d'E/S

Dans cette section, nous illustrons par un exemple introductif le comportement temporel d'un transfert de données que nous souhaitons exprimer à l'aide d'un graphe de contraintes d'E/S. La Figure 3-2 présente un graphe spécifiant la séquence de transfert a_0, a_1, a_2 Dans ce graphe, la date de transfert d'une donnée a_j est exprimée relativement au transfert la donnée précédente $a_{(j-1)}$. Le délai de transfert introduit par le média de communication est associé aux nœuds du graphe sous la forme d'une fenêtre d'incertitude. La Figure 3-3 représente le comportement temporel précédemment exprimé graphiquement. Les plages temporelles des transferts des données se répètent à une cadence de 2 cycles et sont présentées pour 2 itérations successives.

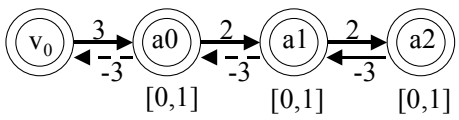


Figure 3-2 : IOCG : Exemple introductif

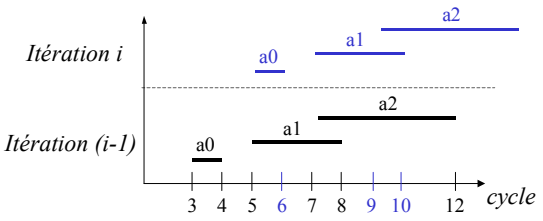


Figure 3-3 : Représentation des plages temporelles de transferts

La **Figure 3-4** représente le comportement temporel du graphe pour lequel à l'itération *(i-1)* les transferts des données a_0, a_1, a_2 sont supposés être réalisés au plus tôt relativement au transfert de leur prédécesseur. Pour l'itération *i* les transferts des données a_0, a_1, a_2 sont supposés être réalisés au plus tard relativement au transfert de leur prédécesseur.

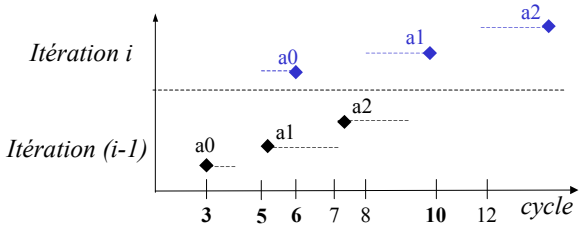


Figure 3-4 : Illustration des transferts à des dates fixées appartenant aux intervalles

2. Nouveau modèle unifié

2.1. Modélisation du Comportement d'un IP algorithmique

Le point d'entrée de notre outil de synthèse consiste en une description algorithmique spécifiant la fonctionnalité d'un composant virtuel dans un langage de description de niveau comportemental faisant abstraction des différentes solutions d'implémentation. Cette description initiale est ensuite compilée afin d'obtenir un modèle de représentation intermédiaire sous la forme d'un Graphe Flot de Signaux *GFS* définit comme suit :

Définition : Un graphe flot de signaux est un graphe polaire orienté *GFS(V, E)* où l'ensemble des nœuds $V = \{v_0, ..., v_n\}$ représente les opérations, v_0 et v_n étant respectivement le nœud source et le nœud puits. L'ensemble d'arcs $E = \{(v_i, v_j)\}$ représente les dépendances entre les nœuds opérations.

Le graphe flot de signaux contient $|V| = n+1$ nœuds. Un nœud représente l'une des opérations suivantes : arithmétique, logique, donnée ou délai. La différence entre un graphe flot de signaux et un graphe flot de données provient des opérations de type *délai* utilisées dans le domaine du traitement du signal pour exprimer l'utilisation d'une donnée dont la valeur est calculée dans une itération précédente de l'algorithme. Un arc $e_{ij} = (v_i, v_j)$ représente une dépendance de données $v_i \succ v_j$ entre les opérations v_i et v_j telle que pour toute itération de *GFS*, l'opération v_i doit démarrer son exécution avant celle de v_j. Pour les dépendances de données, l'exécution de v_j ne peut commencer qu'après la complétion de l'opération v_i.

Remarque : il existe un arc e_{ij} de poids w_{ij} si et seulement s'il existe une dépendance de donnée entre les nœuds v_i et v_j.

La communication entre le composant virtuel et son environnement se fait au travers de port et de canaux de communication par lesquels transitent les données consommées et produites par l'IP. Certains nœuds de données de l'algorithme représentent des opérations de lecture ou d'écriture. Nous définissons ainsi deux types de nœuds dans le graphe.

Définition : un *nœud entrée* v_i d'un graphe flot de signaux *GFS(V, E)* représente une donnée produite par le système sur un port d'entrée du composant virtuel. L'ensemble des nœuds *entrée* est noté $I \subseteq V$.

Un *nœud sortie* d'un graphe flot de signaux *GFS(V, E)* représente une donnée *consommée* par le système via un port de *sortie* du composant virtuel. L'ensemble des nœuds *sortie* est noté $O \subseteq V$.

2.2. Modélisation du graphe de contraintes algorithmiques

Les propriétés temporelles de l'algorithme sont dérivées de l'implantation physique du composant virtuel à partir de la bibliothèque d'opérateurs utilisée pour la synthèse. Dans un flot de synthèse comportementale, les temps d'exécution sont associés aux nœuds opérations durant la phase de sélection des ressources matérielles. Le délai d'une opération v_i est exprimé en nombre de cycles d'horloge, dont la période *T_cycle* est spécifiée par l'intégrateur, et représente la latence l_{opr} de l'opérateur sélectionné pour réaliser l'opération v_i.

Définition : Un graphe de contraintes algorithmiques est un graphe pondéré polaire orienté *ACG(V, E)* où l'ensemble des nœuds $V = \{v_0, ..., v_n\}$ représente les opérations, v_0 et v_n étant respectivement le nœud source et le nœud puits. L'ensemble d'arcs $E= \{(v_i, v_j)\}$ représente les contraintes temporelles entre les nœuds opérations. Un poids w_{ij}, associé à un arc e_{ij}, représente le temps d'exécution du nœud v_i et est noté $t(v_i)$.

$\sigma_{v0}(v_i)$ représente la date à laquelle une opération v_i peut être exécutée (de façon équivalente le cycle) relativement au début de l'exécution de l'algorithme défini par $\sigma_{v0}(v_0)$. Ainsi, $\sigma_{v0}(v_0)$ représente la référence temporelle absolue et a par définition la valeur $\sigma_{v0}(v_0)=0$. **Pour simplifier la lecture, $\sigma_{v0}(v_i)$ sera noté $\sigma(v_i)$ dans le reste du document.**

Nous définissons maintenant deux contraintes temporelles représentant la borne inférieure et supérieure entre les dates d'exécution de deux opérations.

Une contrainte temporelle minimum $l_{ij} \geq 0$ nécessite $\sigma_{v0}(v_j) \geq \sigma_{v0}(v_i) + l_{ij}$

Une contrainte temporelle maximum $u_{ij} \geq 0$ nécessite $\sigma_{v0}(v_j) \leq \sigma_{v0}(v_i) + u_{ij}$

Un premier graphe de contraintes temporelles, appelé *graphe de contraintes algorithmiques ACG(V, E)*, est dérivé du *GFS* en associant des contraintes temporelles minimum aux dépendances de donnée entre opérations arithmétiques et logiques et des contraintes temporelles maximum aux opérations de délai. Ainsi un nœud de type délai Z^n (vieillissement) dans le graphe modélisant l'algorithme est transformé dans le graphe de contrainte en une contrainte de type maximum représentée par un arc arrière dont le poids négatif à la valeur *–n*cadence* (voir Figure 3-5). Les contraintes temporelles sont donc introduites (voir Figure 3-6) dans le graphe flot de signaux, après l'étape de sélection des opérateurs, comme suit :

Pour chaque opérateur *opr*, ayant une latence l_{opr}, alloué à un nœud opération v_i ajout d'un poids w_{ij} positif ayant la valeur $w_{ij} = l_{opr} \geq 0$ sur *l'arc avant* (v_i, v_j).

Remarque : le poids w_{ij} associé à un arc e_{ij} représentant le temps d'exécution de l'opérateur alloué au nœud v_i est noté de façon équivalente dans le document $t(v_i)$

Pour un nœud délai Z^{-n} entre deux nœuds opérations v_i et v_j insertion d'un *Arc arrière* (v_j, v_i) de poids w_{ji} négatif ayant la valeur $w_{ji} = -n*cadence \leq 0$

Remarque : $w_{ji} = -u_{ij} \leq 0$, *car* $\sigma(v_j) \leq \sigma(v_i) + u_{ij}$ *implique* $\sigma(v_i) \geq \sigma(v_j) - u_{ij}$

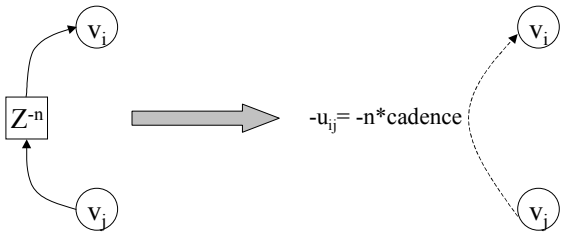


Figure 3-5: Transformation d'un nœud délai

Le graphe de contraintes algorithmiques résultant *ACG(V, E)* est constitué d'un ensemble *E* d'arcs avants E^f et arrières E^b tel que $E = E^f \cup E^b$

Le sous graphe avant $ACG^f(V, E^f)$ est acyclique (une contrainte minimum entre les nœuds v_i et v_j est considérée non valide s'il existe un chemin entre v_j et v_i) et contient uniquement les arcs avants représentant les contraintes temporelles minimums : les dépendances de données.

Le sous graphe arrière $ACG^b(V, E^b)$ contient uniquement les arcs arrières représentant les contraintes temporelles maximum.

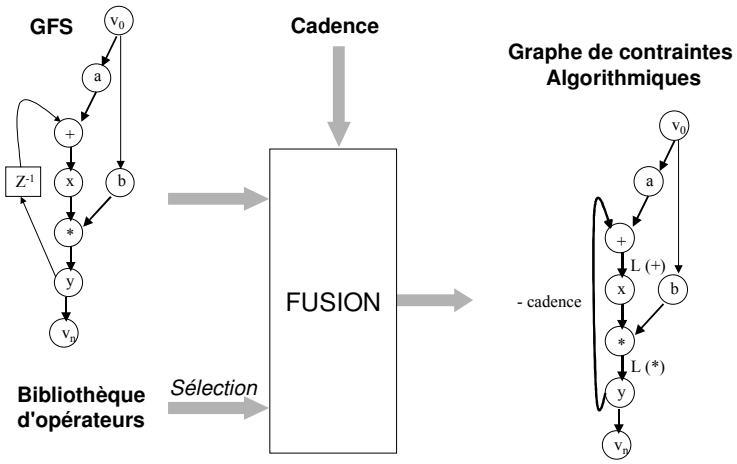


Figure 3-6: Obtention du graphe de contraintes algorithmiques

2.2.1. Connexion

La notion de *connexion* exprime une relation directe de dépendance de donnée (de précédence d'exécution d'opération) entre deux nœuds du graphe et est représentée graphiquement par un arc. Une connexion v_i vers v_j est dénotée par le couple nœud-arc (v_i, e_{ij})

2.2.1.1 Avant

Il y a *connexion avant* entre deux nœuds v_i et v_j s'il existe un arc e_{ij} dans le sous graphe avant $ACG^f(V,E^f)$ allant du nœud v_i au nœud v_j. Si l'arc a un poids $l_{ij} \geq 0$ alors $\sigma(v_j) \geq \sigma(v_i) + l_{ij}$. Le nœud v_j ne pourra débuter son exécution que w_{ij} cycles après l'initialisation de v_i (cf. Figure 3-7).

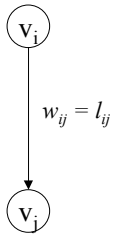


Figure 3-7 : Connexion avant

2.2.1.2 Arrière

Il y a *connexion arrière* entre deux nœuds v_j et v_i s'il existe un arc arrière e_{ji} dans le graphe arrière E^b du nœud v_j au nœud v_i. Si l'arc a un poids $w_{ji} = -u_{ij} \leq 0$ alors $\sigma(v_j) \leq \sigma(v_i) + u_{ij}$. Le nœud v_j devra commencer son exécution $-w_{ji}$ cycles après le début d'exécution de v_i (Figure 3-8).

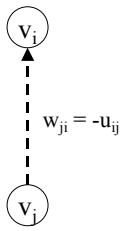


Figure 3-8 : Connexion arrière

La sémantique du nœud puits d'un arc arrière sera détaillée dans la section 2.4.3.2 (nœud référence).

2.2.2. Chemin

La notion de chemin exprime une relation indirecte entre deux nœuds du graphe et est représentée graphiquement par une chaîne de connexions reliants deux nœuds d'un graphe. Un *chemin avant* relie deux nœuds v_i et v_j s'il existe une suite de nœuds connectés dans le graphe avant ACG^f allant de v_i à. v_j. Un *chemin arrière* relie deux nœuds v_j et v_i s'il existe une suite de nœuds connectés dans le graphe arrière ACG^b allant de v_j à v_i.

2.2.2.1 Avant

Définition : soit $C^f_{vi,\ vj}$, l'ensemble des chemins allant de v_i à v_j dans le sous graphe avant $ACG^f(V, E^f)$. Un chemin $c = (V_c, E_c)$ avec $V_c \subseteq V$ et $E_c \subseteq E^f$ est un chemin avant allant de v_i vers v_j, si et seulement si

1. $v_i, v_j \in V_c$ et
2. $\exists! e_{ik} \in E_c$ et $(V_c \backslash v_i\ ,\ E_c \backslash e_{ik}) \in C^f_{vk,\ vj}$ ou $E_c = \{e_{ij}\}$ et $V_c = \{v_i, v_j\}$.

En d'autres termes, (voir Figure 3-9)

1. Le chemin c doit contenir les nœuds v_i et v_j.
2. Il existe un et un seul arc noté e_{ik} partant de v_i tel que le chemin c privé du nœud v_i et de l'arc e_{ik} est un chemin de v_k vers v_j.

Figure 3-9 : Chemin avant

La *longueur* du chemin avant c allant de du nœud v_i au nœud v_j dans le *graphe avant* ACG^f représente la durée *minimum*, pour ce chemin, après laquelle le nœud v_j pourra s'exécuter après la complétion du nœud v_i et est défini par $r^f(c) = \left(\sum_{e_{mk} \in E_c} w_{mk} \right) - t(v_i)$, $c \in C^f_{vi,\ vj}$ où $t(v_i)$ représente le temps d'exécution du nœud v_i.

2.2.2.2 Arrière

Définition : soit $C^b_{vj,\ vi}$, l'ensemble des chemins allant de v_i vers v_j dans le sous graphe arrière $ACG^b(V, E^b)$. Un chemin $c = (V_c, E_c)$ avec $V_c \subseteq V$ et $E_c \subseteq E^b$ est un chemin arrière allant de v_j vers v_i, si et seulement si

1. $v_i, v_j \in V_c$ et
2. $\exists! e_{jk} \in E_c$ et $(V_c \backslash v_j , E_c \backslash e_{jk}) \in C^b_{vk,\ vi}$ ou $E_c = \{e_{ji}\}$ et $V_c = \{v_j, v_i\}$.

En d'autres termes, (voir Figure 3-10)

1. Le chemin c doit contenir les nœuds v_i et v_j.
2. Il existe un et un seul arc noté e_{jk} partant de v_j tel que le chemin c privé du nœud v_j et de l'arc e_{jk} est un chemin de v_k vers v_i.

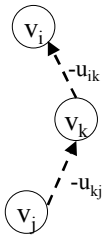


Figure 3-10 : Chemin arrière

La *longueur* du chemin arrière c allant de du nœud v_j au nœud v_i dans le *graphe arrière* ACG^b représente la durée *maximum*, pour ce chemin, séparant le début de l'exécution du nœud v_j et la complétion du nœud v_i et est définie par la valeur négative suivante :

$$r^b(c) = \left(\sum_{e_{mk} \in E_c} w_{mk} \right) + t(v_i) \ , \ c \in C^b_{vj,\ vi}$$

2.2.3. Sous graphe induit

La notion de sous graphe permet de représenter un ensemble de chemins allant du nœud v_i vers le nœud v_j. Le *sous graphe avant induit* par deux nœuds v_i et v_j est l'ensemble de chemins reliant v_i et v_j dans le graphe avant ACG^f et est noté $ACG^f_{vi,\ vj}$.

Le *sous graphe arrière induit* par deux nœuds v_i et v_j est l'ensemble de chemins reliant v_i et v_j dans le graphe arrière ACG^b et est noté $ACG^b_{vj,vi}$.

2.2.3.1 Avant

Définition : le sous graphe avant induit $ACG^f_{vi,\ vj} = (V_{ACG}, E_{ACG})$ est constitué d'un sous ensemble de nœuds $V_G = \bigcup_{c \in C^f_{vi,vj}} V_c$ et d'un ensemble d'arcs $E_G = \bigcup_{c \in C^f_{vi,vj}} E_c$.

La distance minimale séparant le nœud v_i du nœud v_j d'un sous graphe $ACG_{vi,\ vj}$ est égale à la plus grande longueur des différents chemins avant allant de v_i à v_j. Elle représente la durée minimum après laquelle le nœud v_j pourra s'exécuter après la complétion du nœud a et est définie par $\rho^f(v_i, v_j) = \underset{c \in C^f_{vi,vj}}{MAX} \{r^f(c)\}$ (voir Figure 3-11).

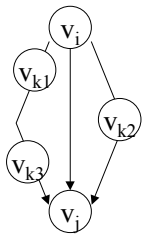


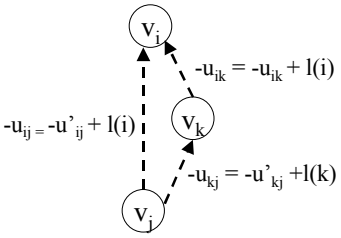
Figure 3-11 : Sous graphe induit avant

2.2.3.2 Arrière

Définition : le sous graphe arrière induit $ACG^b_{vi,\ vj} = (V_{ACG}, E_{ACG})$ est constitué d'un sous ensemble de nœuds $V_G = \bigcup_{c \in C^b_{vi,vj}} V_c$ et d'un ensemble d'arcs $E_G = \bigcup_{c \in C^b_{vi,vj}} E_c$.

La distance maximale séparant le nœud v_j du nœud v_i d'un sous graphe $ACG_{vi,\ vj}$ est égale à la plus grande longueur des différents chemins arrière allant de v_j à v_i. Elle représente la durée maximum entre le début de l'exécution du nœud v_j et la complétion du nœud v_i et est définie par $\rho^b(v_i, v_j) = \underset{c \in C^b_{vi,vj}}{MAX} \{r^b(c)\} = -\underset{c \in C^b_{vi,vj}}{MIN} \{|r^b(c)|\}$ (voir Figure 3-12).

Remarque : dans la suite du document G^f_{vi} représente le sous graphe contenant v_i et tous ses successeurs dans G^f et G^b_{vj} est le sous graphe arrière contenant v_j et tous ses successeurs dans G^b.



$$\rho^b(v_i, v_j) = \mathrm{Max}(-u_{ik} - u_{kj}, -u_{ij})$$

Figure 3-12 : Sous graphe induit arrière

2.3. Ordonnancement d'un graphe déterministe

Le problème d'ordonnancement, lorsque le poids des arcs est déterministe, est défini de la façon suivante

Définition : l'ordonnancement d'un graphe de contrainte *G(V, E)* est un étiquetage entier $\sigma : V \rightarrow Z^+$ d'un ensemble de nœuds V vers des entiers positifs Z^+, tel que $\sigma(v_j) \geq \sigma(v_i) + w_{ij}$ s'il existe un arc reliant v_i à v_j ayant un poids w_{ij}. Un ordonnancement minimum est un ordonnancement tel que $(\sigma(v_i) - \sigma(v_0))$ est minimum pour tous les $v_i \in V$.

2.4. Modélisation des contraintes d'E/S

2.4.1. Origine du non-déterminisme

Les opérations d'entrées et de sorties d'un composant virtuel sont réalisées au travers de ports et de canaux de communication. Dans certains cas les transferts de données entre le composant et son environnement peuvent introduire un indéterminisme sur les dates d'arrivées de données. L'indéterminisme peut avoir deux origines.

1. Le média de communication : la communication entre le composant et le système se fait au travers de canaux de communication partagés. Les conflits d'accès sont résolus lors des phases d'arbitrage (TDMA, priorité) : si un émetteur n'a pas l'autorisation d'émettre sur le média alors il doit attendre la phase d'arbitrage suivante pour soumettre une nouvelle requête.

2. Le traitement du reste du système : les calculs réalisés par les composants produisants les données, consommées par l'IP, contiennent des boucles dynamiques *(data dependent)* ou des structures de contrôle du type branchement conditionnel.

2.4.2. Modes de transfert des E/S et modèle associé

Pour modéliser les échanges de données entre l'environnement et le composant virtuel nous utilisons une modélisation graphique définie comme suit :

Définition : un graphe de contrainte d'entrée/sortie est un graphe hiérarchique pondéré polaire orienté *IOCG(V, E)* où l'ensemble des nœuds $V = \{v_0, ..., v_n\}$ représente des transferts de données ou des étapes de transfert. V_0 et v_n sont respectivement le nœud source et le nœud puits et symbolisent le début et la fin d'une séquence de transfert. L'ensemble d'arcs $E= \{(v_a, v_b)\}$ représente le séquencement des transferts. Un poids w_{ab}, associé à l'arc e_{ab}, représente une contrainte temporelle entre deux transferts de donnée v_a et v_b.

L'ensemble des nœuds V du graphe de contraintes d'E/S *IOCG* est exclusivement composé des nœuds entrée et sortie du graphe flot de signaux *SFG*: $V = I_{SFG} \cup O_{SFG}$ et de nœuds étape de transfert.

L'indéterminisme d'une date de transfert est modélisé à l'aide de contraintes temporelles représentant la borne inférieure et supérieure entre les dates d'exécution de deux opérations de transfert. Ce type de contraintes permet aussi la représentation des caractéristiques des modes de communication : rafale, série...

Une contrainte temporelle minimum $\lambda_{ij} \geq 0$ nécessite $\sigma(v_j) \geq \sigma(v_i) + \lambda_{ij}$

Une contrainte temporelle maximum $\upsilon_{ij} \geq 0$ nécessite $\sigma(v_j) \leq \sigma(v_i) + \upsilon_{ij}$

$\sigma(v_j)$ représentant la date d'émission de la donnée v_j.

L'indéterminisme lié aux transferts de données au travers d'un média de communication, est représenté par un délai de transfert borné (durée d'accomplissement) $\delta(v_a) \in [\delta^{min}(v_a), \delta^{max}(v_a)]$ associé au nœud v_a.

Ainsi, le poids associé à un arc peut être la combinaison d'un délai de transfert (cf. 1. Section 2.4.1) et d'une contrainte de traitement (cf. 2. Section 2.4.1) telle que : $w_{ij} = \lambda_{ij} = l'_{ij} + \delta(v_i)$ pour une contrainte minium et $w_{ij} = -\upsilon_{ij} = -u'_{ij} - \delta(v_i)$ pour une contrainte maximum. Ainsi, l'_{ij} (ou u'_{ij}) représente l'indéterminisme lié aux structures de contrôle de l'algorithme producteur et $\delta(v_i)$ l'indéterminisme introduit par le média de communication sur le transfert de la donnée v_i.

La modélisation des caractéristiques de l'architecture de communication (transfert série, parallèle...) est supportée par le graphe *IOCG* dans lequel elles sont exprimées à l'aide de nœuds hiérarchiques. La Figure 3-13 décrit le séquencement d'un transfert en rafale d'un vecteur de données *A*, et du transfert d'un scalaire *b*. Les caractéristiques des transferts telles que le nombre de cycles séparants deux données consécutives dans une rafale sont introduites dans les nœuds hiérarchiques à l'aide d'arcs avant et arrière entre les nœuds données *(a1, a2, a3)*. Dans ce cas le délai de transfert δ est associé au nœud hiérarchique lui-même en pondérant l'arc avant qui le relie à son prédécesseur dans le sous graphe avant et l'arc arrière le reliant à son successeur dans le sous graphe arrière.

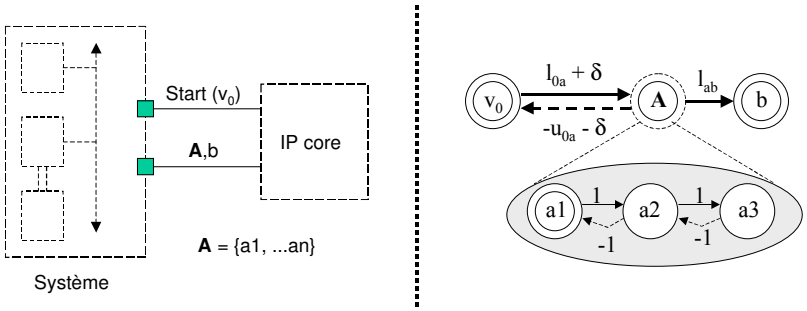
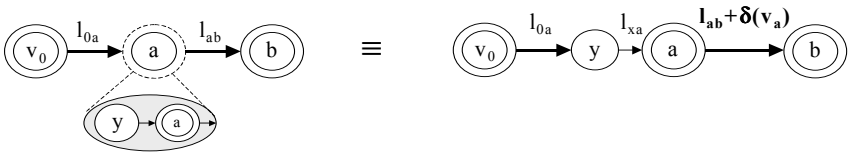


Figure 3-13: Nœud hiérarchique : transfert rafale

Les nœuds *étape de transfert* sont utilisés pour supprimer la corrélation temporelle existant entre deux transferts. La Figure. 3-14 (a) donne un exemple de deux transferts réalisés au travers d'un même port : l'émission de la donnée *b* n'est pas dépendante de la *bonne* réception de la donnée *a*. Remarque : un cumul de retards est possible lorsque plusieurs données sont émises en série sur un même port en utilisant un protocole de poignée de main nécessitant un acquittement de la part du consommateur.

Il faut noter que les nœuds hiérarchiques peuvent inclure des étapes de transfert. Un nœud étape de transferts peut aussi être utilisé pour exprimer explicitement une latence introduite, par exemple, par une conversion de protocole ou de format de donnée (cf. Figure. 3-14 (b))

(a)



(b)

Figure. 3-14: (a) (b) : Exemples d'utilisation de nœuds étape de transfert

La Figure 3-15 schématise l'obtention du graphe de contraintes d'E/S *IOCG* à partir des contraintes algorithmiques du composant producteur.

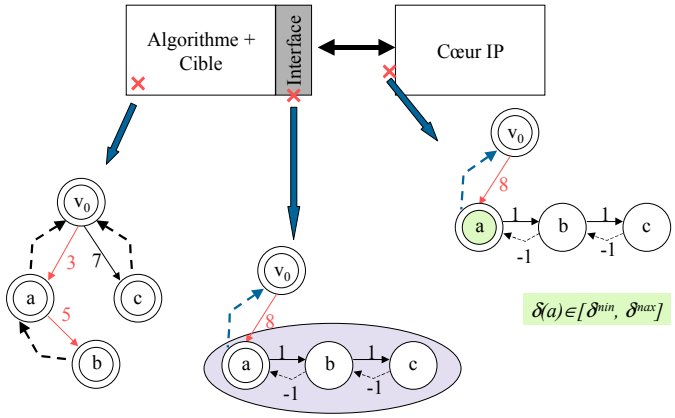


Figure 3-15: Modélisation d'un transfert mode rafale

2.4.3. Modélisation du non déterminisme

Les problèmes d'ordonnancement sous contraintes temporelles sont généralement définis et résolus pour des graphes soumis à des dates d'arrivées de données figées. Dans nos travaux, nous étendons ce problème aux graphes soumis à des dates d'arrivées variables et bornées. La notion de date d'arrivée d'un nœud v_a est exprimée (représentée) par un délai d'exécution représentant la durée de l'opération de lecture (temps d'acquisition et/ou de transfert sur le média de communication). Nous définissons ainsi un sous-ensemble de nœuds, appelés *nœuds de synchronisation* et *nœuds de référence*, servant de référence pour la spécification respective des dates de transfert minimum et maximum des entrées/sorties. Les nœuds de synchronisation sont des nœuds entrée ayant une date d'arrivée minimum *variable* sur les ports de l'IP.

2.4.3.1 Nœud de synchronisation

Définition : les *nœuds de synchronisation* d'un graphe de contrainte *IOCG(V, E)* sont composés du nœud source v_0 et de tous les nœuds entrées ayant une date d'arrivée minimum variable et sont notés $S \subseteq I_{SFG}$

Le nœud v_0 est traité comme un point de synchronisation puisque l'activation du graphe est analogue à la complétion d'un nœud source variable.

Définition : l'ensemble des nœuds de synchronisation d'un nœud v_i est le sous-ensemble des nœuds de synchronisation $S(v_i) \subseteq S$, tel que $a \in S(v_i)$ s'il existe un chemin dans $IOCG^f(V, E^f)$ de a vers v_i contenant au moins **un arc dont le poids variable est égal à $\delta(a) \in [\delta^{min}(a), \delta^{max}(a)]$**

En d'autres termes, un nœud de synchronisation a est dans l'ensemble des nœuds de synchronisation d'un nœud si ce nœud ne peut s'exécuter qu'après l'arrivée de la donnée a.

2.4.3.2 Nœud de référence

Définition : les nœuds *"référence"* d'un graphe de contrainte *IOCG(V, E)* sont composés de tous les nœuds servant de référence à une contrainte de temps maximum et sont notés *Rp*.

Définition : l'ensemble des nœuds *"référence"* d'un nœud v_i est le sous-ensemble des nœuds référence $Rp(v_i) \subseteq Rp$, tel que $a \in Rp(v_i)$ s'il existe un chemin dans $IOCG^b(V, E^b)$ de v_i vers a.

Remarque : un nœud *référence* peut ne pas avoir de dépendances de données avec le nœud v_i puisque la notion de référence est définie dans le sous graphe arrière, i.e. la sortie b est produite u_{ab} cycles après l'arrivée de l'entrée a.

2.5. Graphe de contrainte : algorithme + entrées/sorties

Le graphe de contraintes globales *GCG* est utilisé pour la synthèse de l'unité de traitement du composant virtuel sous contraintes d'intégration. Il est obtenu en fusionnant le graphe de contraintes algorithmiques *ACG* et le graphe de contraintes des E/S *IOCG (*Figure 3-16*)*. Un arc avant entre un nœud entrée v_i et un nœud sortie v_j du graphe *IOCG* représente la date minimum relative à v_i à partir de laquelle peut être lue la donnée v_j sur le port de l'IP : elle représente donc la date au plus tard à laquelle elle doit être produite par l'*UT*. Cette contrainte minimum de communication est donc transformée en une contrainte maximum de production pour la synthèse. Ainsi, un arc avant e_{ij} de poids positif w_{ij} entre un nœud entrée v_i et un nœud sortie v_j du *IOCG* est transformé en un arc arrière e_{ji} de poids négatif $-w_{ij}$ entre le nœud $v_j \in O_{ACG}$ et $v_i \in I_{ACG}$ dans le *GCG* (voir Figure 3-17).

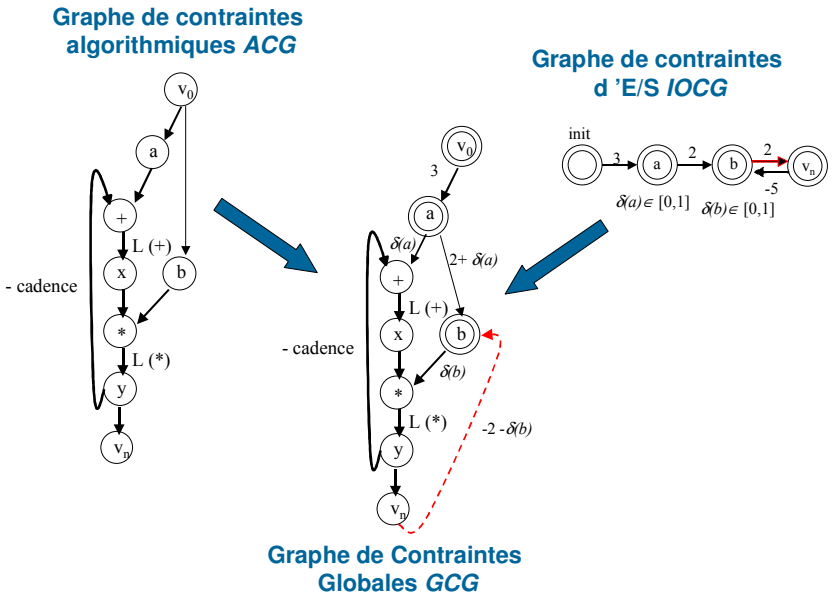


Figure 3-16 : Création du GCG par fusion

Le graphe de contraintes globales est définit de la façon suivante :

Définition : un graphe de contrainte global est un graphe pondéré polaire orienté *GCG(V, E)* où l'ensemble des nœuds $V = \{v_0, ..., v_n\}$ représente les opérations, v_0 et v_n étant respectivement le nœud source et le nœud puits. L'ensemble d'arcs $E = \{(v_i, v_j)\}$ représente les dépendances entre les nœuds. Un poids w_{ij} associé à un arc e_{ij} représente une contrainte temporelle entre les nœuds v_i et v_j.

Le graphe de contrainte contient $|V| = n+1$ nœuds. Un nœud représente l'une des opérations suivantes : arithmétique, logique, entrée ou sortie.

Un arc $e_{ij} = (v_i, v_j)$ représente

a) Une dépendance de données $v_i \succ v_j$ entre deux nœuds arithmétiques ou logiques v_i et v_j telle que pour toute itération de *GCG*, l'opération v_i doit être complétée avant que v_j ne puisse s'exécuter. *(relation opr ⇔ opr)*

b) Un séquencement $v_i \succ v_j$ entre les transferts des nœuds entrées/sortie v_i et v_j. *(relation I/O ⇔ I/O)*

c) Une dépendance temporelle $v_i \succ v_j$ entre un nœud entrée v_i et un nœud arithmétique ou logique v_j telle que pour toute itération de *GCG*, la donnée v_i doit avoir été transférée avant que v_j ne puisse commencer à s'exécuter. *(relation I ⇔ opr)*

d) Une dépendance temporelle $v_i \succ v_j$ entre un nœud opération v_i et un nœud sortie v_j telle que pour toute itération de *GCG*, l'opération v_i doit avoir terminé son exécution avant que v_j ne puisse être transféré. *(relation opr ⇔ O)*. La Figure 3-17 décrit un exemple de graphe de contrainte globale.

Remarque : Dans les cas a) et b) les arcs sont issus des graphes *ACG* et *IOCG*. Dans le cas c) le poids des arcs correspond à l'incertitude sur les dates de transfert des *E/S*. Le cas d) fait référence au temps de propagation des derniers opérateurs produisants les sorties. L'intégrateur ne peut donc en aucun cas spécifier directement des contraintes maximum impliquant un nœud de calcul et une variable interne (sauf opération de délai).

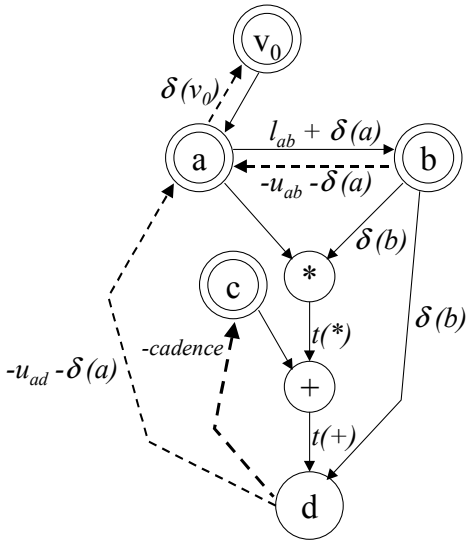


Figure 3-17: Exemple de Graphe de contrainte *GCG*

2.6. Composition de contraintes

Nous étendons le problème d'ordonnancement en présence de dates d'arrivées variables de données en introduisant le concept *d'offset* par rapport aux nœuds de synchronisation et de référence du graphe. Un offset représente la longueur d'un chemin, et est graphiquement symbolisé par une suite de connexions reliants une entrée non déterministe et un nœud opération. Un offset entre un nœud entrée et un nœud sortie sera considéré comme une contrainte minimale de synthèse suivant les hypothèses d'implémentation.

2.6.1. Connexion

2.6.1.1 Avant

Il y a *connexion avant* entre deux nœuds v_a et v_b s'il existe un arc e_{ab} dans le sous graphe avant $GCG^f(V,E^f)$ allant du nœud v_a au nœud v_b. Si l'arc a un poids $\lambda_{ab} \geq 0$ alors $\sigma(v_b) \geq \sigma(v_a) + \lambda_{ab}$. Le nœud v_b ne pourra être transféré que w_{ab} cycles après le transfert de v_a.

Remarquons que dans le *GCG*, la notion de *connexion avant* entre deux nœuds $v_a \in I \cup O$ et v_b $I \cup O$ exprime le séquencement des transferts de v_a et v_b, et est représentée graphiquement par un arc. Une connexion v_a vers v_b est dénotée par le couple nœud-arc (v_a, e_{ab}).

La Figure 3-18 illustre une connexion avant dans le *GCG* entre (a) deux nœuds v_a et v_b non synchronisation et (b) entre un nœud de synchronisation v_a et un nœud v_b.

E. **(b)**

Figure 3-18 : Connexion avant (a) entre nœuds non synchronisation, (b) avec nœud de synchronisation

2.6.1.2 Arrière

Il y a *connexion arrière* entre deux nœuds v_a et v_b s'il existe un arc arrière e_{ba} dans le sous graphe arrière $GCG^b(V^bE^b)$ du nœud v_a au nœud v_b. Si l'arc a un poids $w_{ba} = -\upsilon_{ab} \leq 0$ alors $\sigma(v_b) \leq \sigma(v_a) + \upsilon_{ab}$.

Lorsque $v_a \in I \cup O$ et v_b $I \cup O$, la connexion arrière spécifie que le nœud v_b devra être transféré $-w_{ba}$ cycles au maximum après le transfert de v_a.

La Figure 3-19 illustre une connexion avant dans le *GCG* entre (a) deux nœuds v_a et v_b non synchronisation, (b) entre un nœud v_a et un nœud de synchronisation v_b . et (c) entre un nœud v_a et un nœud de synchronisation v_b.

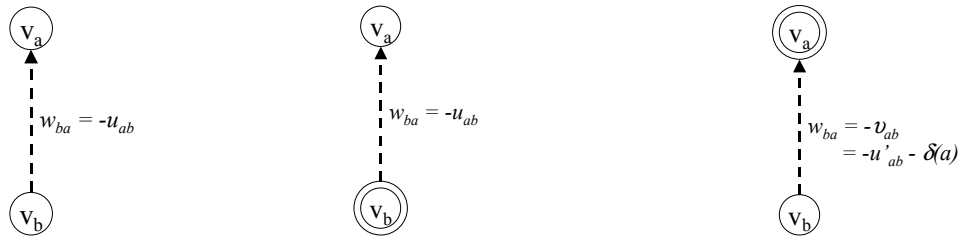


Figure 3-19 : Connexion arrière (a) entre nœuds non synchronisation, (b) issu d'un nœud de synchronisation (c) avec un nœud de synchronisation

2.6.2. Chemin

La notion de chemin exprime une relation indirecte entre deux nœuds du graphe et est représentée graphiquement par une chaîne de connexions reliants deux nœuds d'un graphe.

Un *chemin avant* relie deux nœuds v_i et v_j s'il existe une suite de nœuds connectés dans le graphe avant GCG^f allant de v_i à. v_j.

Un *chemin arrière* relie deux nœuds v_j et v_i s'il existe une suite de nœuds connectés dans le graphe arrière GCG^b allant de v_j à v_i.

Remarquons que dans le *GCG*, la notion de *chemin avant* entre nœuds *entrée* exprime une séquence de transferts alors qu'un chemin avant entre nœuds entrée et sortie représente (1) par les arcs initiaux du *ACG* une suite de traitements et (2) par les arcs initiaux du *IOCG* une séquence de transferts.

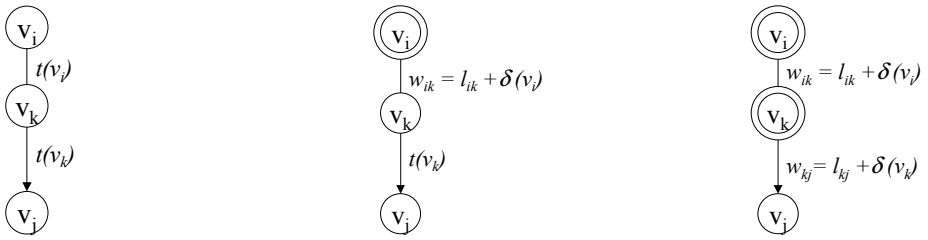
2.6.2.1 Avant

Définition : soit $C^f_{vi,\ vj}$, l'ensemble des chemins allant de v_i à v_j dans le sous graphe avant $GCG^f(V, E^f)$. Un chemin $c = (V_c, E_c)$ avec $V_c \subseteq V$ et $E_c \subseteq E^f$ est un chemin avant allant de v_i vers v_j, si et seulement si

1. $v_i, v_j \in V_c$ *et*
2. $\exists! e_{ik} \in E_c$ *et* $(V_c \backslash v_i , E_c \backslash e_{ik}) \in C^f_{vk,\ vj}$ *ou* $E_c = \{e_{ij}\}$ *et* $V_c = \{v_i, v_j\}$.

En d'autres termes, (voir Figure 3-20)

1. Le chemin c doit contenir les nœuds v_i et v_j.
2. Il existe un et un seul arc noté e_{ik} partant de v_i tel que le chemin c privé du nœud v_i et de l'arc e_{ik} est un chemin de v_k vers v_j.



(a) **(b)** **(c)**

Figure 3-20 : Chemin avant : (a) entre nœuds non synchronisation, (b) issu d'un nœud de synchronisation (c) contenant un nœud de synchronisation

La *longueur* du chemin avant c allant de du nœud v_i au nœud v_j dans le *graphe avant* ACG^f représente la durée *minimum*, pour ce chemin, après laquelle le nœud v_j pourra s'exécuter après la complétion du nœud v_i et est défini par $r^f(c) = \left(\sum_{e_{mk} \in E_c} w_{mk} \right) - t(v_i),\ c \in C^f_{vi,\ vj}$ où $t(v_i)$ représente (Figure 3-20(a)) un temps d'exécution lorsque v_i est une opération de traitement ou (Figure 3-20(b)(c)) un délai de transfert $\delta(v_i)$ lorsque v_i est une opération de lecture sur un port.

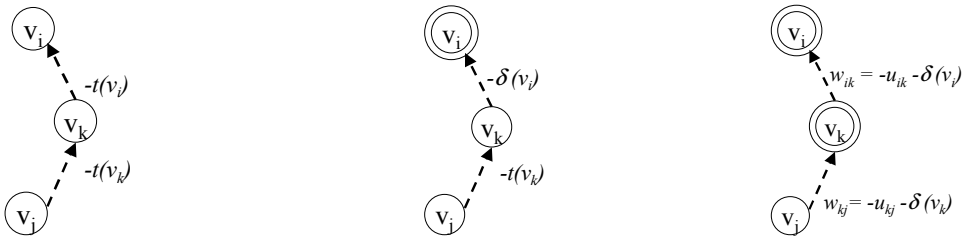
2.6.2.2 Arrière

Définition : Soit $C^b_{vj,\ vi}$, l'ensemble des chemins allant de v_i vers v_j dans le sous graphe arrière $ACG^b(V, E^b)$. Un chemin $c = (V_c, E_c)$ avec $V_c \subseteq V$ *et* $E_c \subseteq E^b$ est un chemin arrière allant de v_j vers v_i, si et seulement si

3. $v_i, v_j \in V_c$ *et*
4. $\exists! e_{jk} \in E_c$ *et* $(V_c \backslash v_j\ ,\ E_c \backslash e_{jk}) \in C^b_{vk,\ vi}$ *ou* $E_c = \{e_{ji}\}$ *et* $V_c = \{v_j, v_i\}$.

En d'autres termes, (voir Figure 3-21)

1. Le chemin c doit contenir les nœuds v_i et v_j.
2. Il existe un et un seul arc noté e_{jk} partant de v_j tel que le chemin c privé du nœud v_j et de l'arc e_{jk} est un chemin de v_k vers v_i.



(a) **(b)** **(c)**

Figure 3-21 : Chemin arrière (a) entre nœuds non référence, (b) issu d'un nœud référence (c) contenant un nœud référence

La *longueur* du chemin arrière *c* allant de du nœud v_j au nœud v_i dans le *graphe arrière* ACG^b représente la durée *maximum*, pour ce chemin, séparant le début de l'exécution du nœud v_j et la complétion du nœud v_i et est définie par la valeur négative suivante :

$r^b(c) = \left(\sum_{e_{mk} \in E_c} w_{mk} \right) + t(v_i)$, $c \in C^b_{vj,\, vi}$ où $t(v_i)$ représente (Figure 3-21(a)) un temps d'exécution lorsque v_i est une opération de traitement ou (Figure 3-21(b)(c)) un délai de transfert $\delta(v_i)$ lorsque v_i est une opération d'E/S.

2.6.3. Compétition entre chemins déterministes ayant un même nœud de synchronisation origine et une même destination

Nous définissons maintenant le concept *d'offset* minimum et maximum. Un *offset minimum* entre un nœud de synchronisation *a* et un nœud v_i représente la *durée minimum* séparant la complétion du nœud *a* du début de l'exécution du nœud v_i. Un *offset maximum* entre un nœud référence *a* et un nœud v_i représente la *durée maximale* séparant la complétion du nœud a de l'exécution du nœud v_i. La notion d'offset est équivalente à la notion de longueur de chemin précédemment définit pour un *ACG*.

2.6.3.1 Offset minimum

Soit $V_a \subseteq V$ le sous-ensemble des nœuds contenant *a* et tous ses successeurs et $GCG^f_a(V_a, E^f_a)$ le sous graphe avant induit par V_a supposant toutes les dates d'arrivées des nœuds de synchronisation fixées à $\sigma(v_0) = 0$. Un offset minimum $\sigma_a(v_j)$ représente la durée minimum après la complétion du nœud de synchronisation a, avant que v_j ne puisse s'exécuter.

Définition : l'*offset* d'un nœud $v_j \in V_a$ par rapport au nœud de synchronisation *a* est une valeur entière $\sigma_a(v_j)$ telle que $\sigma_a(v_j) \geq \sigma_a(v_i) + w_{ij}$ s'il existe un arc de poids w_{ij} du nœud v_i au nœud v_j dans le sous graphe avant $GCG^f_a(V_a, E^f_a)$, et $\sigma_a(a)$ est normalisé à 0. Si $\sigma_a(v_i)$ est la valeur minimum, alors c'est l'offset minimum de v_i relativement à *a* et est noté $\sigma_a^{min}(v_j) = \rho^f(a, v_j)$.

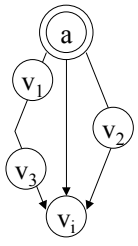


Figure 3-22

La Figure 3-22 décrit un exemple de compétition entre chemins déterministes ayant un même nœud de synchronisation origine et une même destination

2.6.3.2 Date d'exécution

Nous établissons la relation entre les offsets $\sigma_a(v_j)$ et la date d'exécution $\sigma(v_j)$ d'un nœud v_j en présence d'un unique nœud de synchronisation *a* comme suit :

$$\sigma(v_j) \geq \sigma(a) + \delta(a) + \sigma_a^{min}(v_j)$$

2.6.4. Compétition entre chemins déterministes ayant un même nœud source origine et un même nœud de référence destination

2.6.4.1 Offset maximum

Soit $V_a \subseteq V$ le sous-ensemble des nœuds contenant *a* et tous ses successeurs et $GCG^b{}_a(V_a, E^b{}_a)$ le sous graphe arrière induit par V_a supposant toutes les dates d'arrivées des nœuds de synchronisation fixées à $\sigma(v_0) = 0$. Un offset maximum $\sigma_a^{max}(v_j)$ représente la durée maximum après la complétion du nœud de référence a, pendant laquelle v_j doit démarrer son exécution.

Définition : L'*offset* maximum d'un nœud $v_j \in V_a$ par rapport au nœud de référence *a* est une valeur entière $\sigma_a^{max}(v_j) = \rho^b(a, v_j)$ telle que $\sigma_a^{max}(v_j) \geq \sigma_a^{max}(v_i) + w_{ji}$ s'il existe un arc de poids w_{ji} du nœud v_j au nœud v_i dans le sous graphe arrière $GCG^b{}_a(V_a, E^b{}_a)$, et $\sigma_a^{max}(a)$ est normalisé à 0.

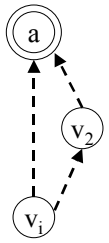


Figure 3-23

La Figure 3-23 décrit un graphe où il existe une compétition entre chemins déterministes ayant un même nœud source origine et un même nœud de référence destination.

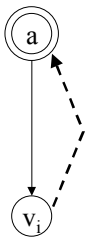
2.6.4.2 Date d'exécution

Nous établissons la relation entre les offsets $\sigma_a(v_j)$ et la date d'exécution $\sigma(v_j)$ d'un nœud v_j en présence d'un unique nœud de référence *a* par la relation suivante :

$$\sigma(v_j) \leq \sigma(a) + \delta(a) + \sigma_a^{max}(v_j)$$

Ayant défini pour les cas simples la date d'exécution d'un nœud, nous établissons maintenant par construction les relations entre les offsets et les dates d'exécutions minimum et maximum dans des graphes plus complexes où les nœuds ont plus d'un nœud de synchronisation et plus d'un nœud de référence.

2.6.5. Date d'exécution d'un nœud ayant un nœud de synchronisation et un nœud de référence identique



La Figure 3-24 décrit un exemple de graphe contenant un nœud sortie ayant un nœud de synchronisation et un nœud de référence identique.

Figure 3-24

Dans ce cas, l'arc avant allant du nœud de synchronisation v_a au nœud v_i implique :

$$\sigma(v_j) \geq \sigma(a) + \delta(a) + \sigma_a^{min}(v_j)$$

et l'arc arrière allant du nœud v_i au nœud référence v_a :

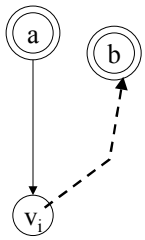
$$\sigma(v_j) \leq \sigma(a) + \delta(a) + \sigma_a^{max}(v_j)$$

Les deux relations précédentes induisent entre l'offset minimum et l'offset maximum l'inéquation suivante :

$$\sigma_a^{min}(v_j) \leq \sigma_a^{max}(v_j).$$

L'inéquation précédente implique l'inexistence de cycle positif dans le graphe. Cette propriété sera vérifiée par la phase d'analyse détaillée dans la section 3.1.

2.6.6. Date d'exécution d'un nœud ayant un nœud de synchronisation et un nœud de référence différent



La Figure 3-25 décrit un exemple de graphe contenant un nœud ayant un nœud de synchronisation et un nœud de référence différent.

Figure 3-25

Dans ce cas, l'arc avant allant du nœud de synchronisation v_a au nœud v_i implique :

$$\sigma(v_j) \geq \sigma(a) + \delta(a) + \sigma_a^{min}(v_j)$$

et l'arc arrière allant du nœud v_i au nœud référence v_b :

$\sigma(v_j) \leq \sigma(b) + \delta(b) + \sigma_b^{max}(v_j)$

Les deux relations précédentes induisent entre l'offset minimum relatif au nœud de synchronisation et l'offset maximum relatif au nœud de référence l'inéquation suivante :

$\sigma(a) + \delta(a) + \sigma_a^{min}(v_j) \leq \sigma(v_j) \leq \sigma(b) + \delta(b) + \sigma_b^{max}(v_j)$

2.6.7. Date d'exécution d'un nœud ayant plusieurs nœuds de synchronisation

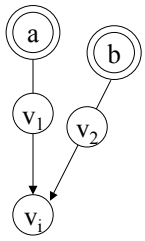


Figure 3-26

La Figure 3-26 décrit un exemple de graphe contenant un nœud ayant plusieurs nœuds de synchronisation

Dans ce cas de figure, nous définissons à partir de l'ensemble des arcs avant allant des nœuds de synchronisation de v_i, $r \in S(v_i)$ au nœud v_i la date d'exécution du nœud v_i comme étant :

$\sigma(v_i) \geq Max_{\,r \in S(vi)} \{\sigma(r) + \delta(r) + \sigma_r^{min}(v_i)\}$

2.6.8. Date d'exécution d'un nœud ayant plusieurs nœuds référence

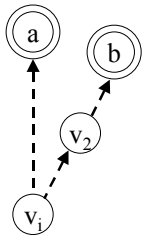


Figure 3-27

La Figure 3-27 décrit un exemple de graphe contenant un nœud ayant plusieurs nœuds références.

Dans ce cas de figure, nous définissons à partir de l'ensemble des arcs arrière allant du nœud v_i à ses nœuds référence $r \in Rp(v_i)$ la date d'exécution du nœud v_i comme étant :

$\sigma(v_i) \leq Min_{\,r \in Rp(vi)} \{\sigma(r) + \delta(r) + \sigma_r^{max}(v_i)\}$

2.6.9. Date d'exécution d'un nœud ayant plusieurs nœuds de synchronisation et plusieurs nœuds référence

Cette section généralise, grâce aux relations établies précédemment, la définition de la date d'exécution d'un nœud ayant plusieurs nœuds de synchronisation et plusieurs nœuds référence (voir Figure 3-28).

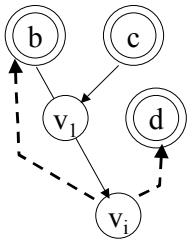


Figure 3-28

L'ensemble des arcs avant permet d'établir l'inéquation suivante :

$$\sigma(v_j) \geq Max_{\,r \in S(vj)} \{\sigma(r) + \delta(r) + \sigma_r^{min}(v_j)\}$$

L'ensemble des arcs arrière permet d'établir l'inéquation suivante :

$$\sigma(v_i) \leq Min_{q \in Rp(vi)} \{\sigma(q) + \delta(q) + \sigma_q^{max}(v_i)\}$$

La date d'exécution d'un nœud v_j doit vérifier les inéquations suivantes :

$$\sigma^{min}(v_j) \leq \sigma(v_j) \leq \sigma^{max}(v_j)$$

et

$$\sigma_a^{min}(v_j) \leq \sigma_a(v_j) \leq \sigma_a^{max}(v_j)$$

Si le sous graphe $GCG'_{vi}(E', V')$ est fortement connexe entre les nœuds entrées et sorties alors la date d'exécution de v_j est définie de façon récursive comme suit :

$$\sigma(v_i) = \{\sigma(a) + \delta(a) + \sigma_a(v_i)\}_{a \in I}$$

Remarque : s'il n'existe aucun nœud dont la durée d'exécution est variable, alors les dates d'exécution sont spécifiées en terme *d'offest* par rapport à v_0 ce qui réduit le problème à un ordonnancement traditionnel.

Aux dates d'exécution minimum et maximum nous ajoutons l'ensemble suivant de définitions:

La *date minimum absolue* d'exécution est définie de façon récursive comme suit *:*

$$\sigma^{min}(v_i) = Max_{\,r \in S(vi)} \{\sigma^{min}(r) + \delta^{min}(r) + \sigma_r^{min}(v_i)\}$$

La *date d'exécution minimum relative* d'un nœud opération est définie comme suit *:*

$$\sigma^{min}(v_i) = Max_{\,r \in S(vi)} \{\sigma(r) + \delta(r) + \sigma_r^{min}(v_i)\}$$

La *date d'exécution maximum absolue* du nœud v_i est définie de façon récursive comme suit :

$\sigma^{max}(v_i) = Min_{q \in Rp(vi)} \{\sigma^{max}(q) + \delta^{max}(q) + \sigma_q^{max}(v_i)\}$

La *date d'exécution* maximum *relative* d'un nœud opération est définie comme suit :

$\sigma^{max}(v_i) = Min_{q \in Rp(vi)} \{\sigma(q) + \delta^{max}(q) + \sigma_q^{max}(v_i)\}$

Remarque : l'expression des dates d'exécution relative permet de prendre en compte la corrélation temporelle qu'il existe entre les nœuds entrées. Cette corrélation est exprimée dans le graphe *IOCG* par des arcs de dépendance de type minimum (avant) et maximum (arrière).

2.7. Pertinence, synchronisation et référence

2.7.1. Nœud de synchronisation pertinent

Nous identifions les nœuds de synchronisation d'un nœud v_i pouvant directement affecter sa date d'exécution $\sigma(v_i)$ par **une dépendance de donnée** en introduisant le critère de *pertinence*.

Définition : un chemin avant *"pertinent"* d'un nœud de synchronisation $a \in S$ vers un nœud v_i, dénoté $c^f(a, v_i)$, est un chemin de a vers v_i dans $GCG^f(V, E^f)$ tel qu'il contient exactement un arc ayant un poids variable (égal à $\delta(a)$).

Définition : l'ensemble des nœuds de synchronisation *pertinents* d'un nœud v_i d'un graphe de contraintes $GCG^f(V, E^f)$ est l'ensemble des nœuds de synchronisation $R(v_i) = \{a \mid a \in S$ *tel qu'il existe un chemin avant pertinent* $c^f(a, v_i)\}$

2.7.2. Nœud de référence pertinent

Nous identifions les nœuds de référence d'un nœud v_i servant directement de référence dans le calcul de la date d'exécution $\sigma(v_i)$ en introduisant le concept de nœud de référence pertinent d'un nœud.

Définition : un chemin arrière « *pertinent* » d'un nœud v_i vers un nœud de référence $a \in S$, dénoté $c^b(v_i, a)$, est un chemin de v_i vers a dans $GCG^b(V, E^b)$ tel qu'il contient exactement un arc ayant un poids variable (égal à $\delta(a)$).

Définition : l'ensemble des nœuds de référence *pertinents* d'un nœud v_i d'un graphe de contrainte $GCG^b(V, E^b)$ est l'ensemble des nœuds de référence $RRp(v_i) = \{a \mid a \in Rp$ *tel qu'il existe un chemin arrière pertinent* $c^b(a, v_i)\}$

2.7.3. Répercussion sur le calcul des dates d'exécution

La date d'exécution minimum relative est définie de façon récursive comme suit :

$\sigma^{min}(v_i) = Max_{\ a \in R(vi)} \{\sigma(a) + \delta(a) + \sigma_a^{min}(v_i)\}$

La date d'exécution maximum *relative* est définie de façon récursive comme suit :

$\sigma^{max}(v_i) = Min_{\ a \in RRp\ (vi)} \{\sigma(a) + \delta(a) + \sigma_a^{max}(v_i)\}$

ATTENTION : afin de faciliter la lecture, dans la suite du document la non pertinence d'un nœud sera exprimée explicitement : la pertinence est donc implicite. Ainsi l'expression *nœud de synchronisation référence* est équivalente à l'expression *nœud de synchronisation pertinent et nœud de référence pertinent*.

3. Existence d'une solution

Une considération importante durant la phase d'ordonnancement est de savoir si une solution existe. Une analyse de la consistance des contraintes temporelles est présentée dans [CAM86] par Camposano et Kunzman. Dans cette approche, les graphes sont supposés contenir des nœuds dont les temps d'exécution sont fixes. Dans ce cas, un ordonnancement existe si le graphe ne contient aucun cycle dont la somme des poids est strictement positive : le cas contraire impliquerait que le délai requis pour produire un résultat serait supérieur au délai autorisé. Les algorithmes du Traitement du Signal et de l'Image (TDSI) sont caractérisés par des traitements itératifs (délais Z^{-n}) cadencés. Pour vérifier la cohérence entre la cadence d'itération, l'ensemble des chemins utilisant des opérations de délai, nous définissons la notion de faisabilité. Le comportement dynamique de l'algorithme sera quant à lui vérifié par une étude du graphe de contrainte globale.

3.1. Etude de faisabilité

Définition : une contrainte temporelle est *faisable* si elle peut être satisfaite lorsque toutes les dates d'arrivée des nœuds entrée sont fixées à 0, i.e. $\delta(a) = 0, \ \forall a \in S$. et $\sigma(a) = \sigma(v_0) = 0$.

Sinon elle est *non faisable*.

Dans le cas où le graphe ne contiendrait pas de nœuds de synchronisation, le critère de faisabilité est suffisant pour assurer qu'il existe une solution au problème d'ordonnancement. Ainsi, un *graphe de contraintes algorithmiques ACG* est faisable si toutes les contraintes sont faisables. Pour les graphes de contraintes algorithmiques, les dates d'arrivées des entrées sont égales à $\sigma(v_0) = 0$. Nous établissons la condition nécessaire et suffisante pour la faisabilité d'un graphe de contraintes dans le théorème suivant.

Le graphe de contraintes noté $ACG_0(V, E)$ suppose la date d'exécution des nœuds de synchronisation fixée à 0, *i.e.* $\delta(a) + \sigma(a) = \sigma(v_0) = 0, \forall a \in I$, *i.e.* $\delta(a) = 0 \ \forall a \in I$.

Théorème 1: un graphe de contraintes $ACG_0 (V, E)$ est *faisable* si et seulement si aucun cycle positif n'existe dans ACG_0. Sinon il est infaisable c'est à dire $\exists \sigma_a^{min}(v_j) > \sigma_a^{max}(v_j)$ avec v_j , $a \in V$ ou $\exists \rho^f(v_i, v_j) > \rho^b(v_i, v_j)$ avec $v_i, v_j \in V$.

Preuve : soit le graphe de contraintes noté $ACG_0(V, E)$ supposant la date d'exécution des nœuds de synchronisation fixée à 0, *i.e.* $\delta(a) + \sigma(a) = \sigma(v_0) = 0, \forall a \in I$. Nous prouvons dans un premier temps la condition nécessaire. Si le graphe ACG est faisable, alors toutes les contraintes dans ACG_0 doivent être faisables. Soit $\Omega = \{\sigma(v_i) \mid \forall v_i \in V\}$ dénotant un ordonnancement du graphe de contrainte $ACG_0(V, E)$ satisfaisant les contraintes. Considérons à présent un cycle dans le graphe, dénoté par $(v_1, v_2), (v_2, v_3), ..., (v_{s-1}, v_s), (v_s, v_1)$. Le système d'inégalités impliqué sur les contraintes par les arcs est le suivant :

$$\sigma(v_1) + w_{1,2} \leq \sigma(v_2)$$

$$\sigma(v_2) + w_{2,3} \leq \sigma(v_3)$$

$$\dots$$

$$\sigma(v_{s-1}) + w_{s-1,s} \leq \sigma(v_s)$$

$$\sigma(v_s) + w_{s,1} \leq \sigma(v_1)$$

En additionnant les inégalités précédentes nous obtenons

$$\sigma(v_1) + \text{somme (poids des arcs du cycle)} \leq \sigma(v_1)$$

Puisque toutes les contraintes sont consistantes, l'inégalité précédente doit aussi être satisfaite. La longueur du cycle doit donc être négative. Ceci est vrai pour tous les cycles du graphe ACG_0, et nous concluons qu'aucun cycle positif n'existe dans le graphe.

Inversement, supposons qu'aucun cycle positif n'existe dans le graphe. $\sigma_{v0}^{min}(v_i)$ représente la longueur du plus long chemin entre le nœud source v_0 et v_i. Nous allons montrer que $\{\sigma_{v0}^{min}(v_i), v_i \in V\}$ est un ensemble des solutions, c'est à dire pour tout arc $e_{ij} \in E$ avec un poids w_{ij}, l'inégalité $\sigma_{v0}^{min}(v_i) + w_{ij} \leq \sigma_{v0}^{min}(v_j)$ est satisfaite, qui implique que le graphe de contrainte est faisable. Supposons, dans l'absurde qu'il existe $e_{ij} \in E$ tel que la contrainte est violée :

$$\sigma_{v0}^{min}(v_i) + w_{ij} > \sigma_{v0}^{min}(v_j).$$

L'inégalité précédente implique que le plus long chemin entre v_0 et v_i ne passe pas par l'arc e_{ij}, sinon $\sigma_{v0}^{min}(v_i) + w_{ij}$ aurait été égale à $\sigma_{v0}^{min}(v_j)$. Plus particulièrement, le chemin allant de v_0 à v_i composé du plus long chemin entre v_0 et v_i, suivi de l'arc e_{ij}, est plus long que le plus long des chemins allant de v_0 à v_j. Ceci contredit la définition de $\sigma_{v0}^{min}(v_j)$, et l'inégalité précédente est donc fausse. Puisque les arguments précédents s'appliquent à tous les arcs, $\{\sigma_{v0}^{min}(v_i), v_i \in V\}$ est un ensemble de solutions et le graphe ACG est faisable.

Ce théorème garantit que le graphe de contraintes algorithmiques ne contient aucune contrainte maximum entre deux nœuds v_j et v_i strictement inférieure à la longueur du plus long chemin reliant v_i à v_j. La cadence spécifiée permet de respecter les opérations de délais (Z^{-n}) en fonction des dépendances de données et des temps de propagation des opérateurs sélectionnés.

3.2. Etude de bien posé

Nous considérons maintenant la consistance des contraintes en présence de dates d'arrivées variables pour les nœuds entrées. L'existence d'un ordonnancement peut être vérifiée statiquement puisque les fenêtres temporelles d'arrivée des nœuds entrées sont connues.

Définition : une contrainte temporelle est *bien posée* si elle peut être satisfaite pour toutes les valeurs des dates d'arrivées des nœuds de synchronisation $\forall a \in R \subseteq I$, i.e $\forall \delta(a) \in [\delta^{min}(a), \delta^{max}(a)]$ et $\sigma(a) \in [\sigma^{min}(a), \sigma^{max}(a)]$.

Réciproquement, une contrainte temporelle est *mal posée* si elle ne peut être satisfaite pour au moins une valeur de date d'arrivée.

Un graphe de contrainte faisable est bien posé si toutes les contraintes maximums sont bien posées.

Remarque : une contrainte temporelle maximum peut être définie par l'intégrateur de l'IP entre deux entrées, deux sorties ou une entrée et une sortie. En aucun cas, l'intégrateur ne peut spécifier une contrainte maximum impliquant un nœud de calcul ou une variable interne (voir section 2.5). La consistance des contraintes maximum entre deux sorties n'a pas à être vérifiée puisque qu'elle peut toujours être assurée par l'unité de communication. Ainsi, seules les contraintes maximums faisant référence aux entrées sont vérifiées.

La Figure 3-29 illustre un exemple d'ensemble de contraintes faisables mais mal posées. En effet, le graphe de la Figure 3-29 (a) ne contient aucun cycle positif mais la date d'exécution minimum relative au nœud de synchronisation *b* (voir Figure 3-29 (b)) peut dans certains cas (voir Figure 3-29 (c)) être supérieure à la date d'exécution maximum spécifiée relativement au nœud référence *a*.

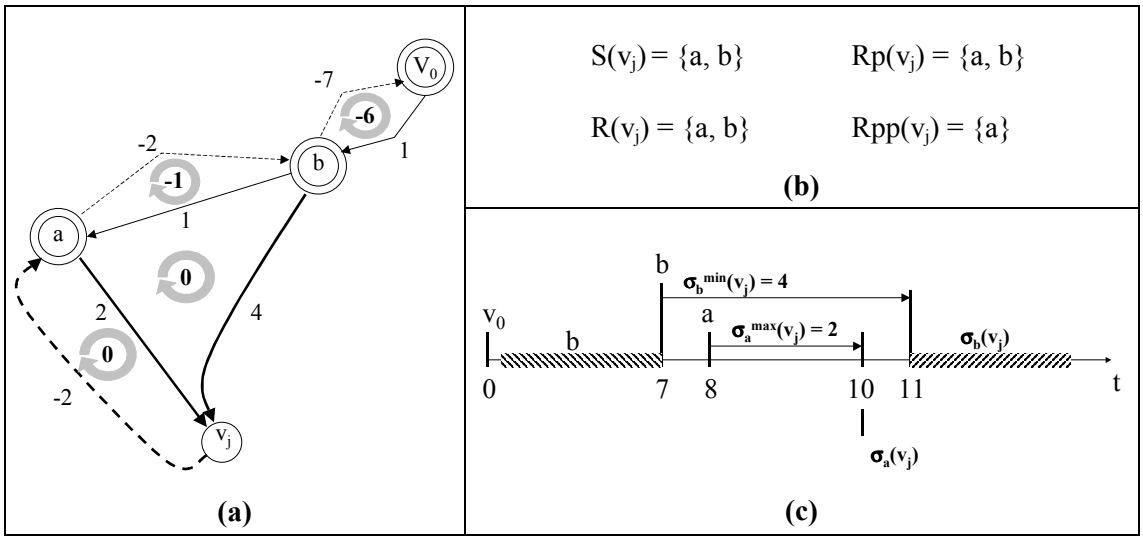


Figure 3-29 : Exemple de graphe faisable et mal posé

Lemme : Un graphe de contrainte faisable *GCG(V, E)* est *bien posé* si et seulement si

$$\sigma(v_j) \leq \sigma(a) + \delta(a) + \sigma_a^{max}(v_j) \quad \forall v_j \in V^b,\ a \in RRp(v_j),\ \forall \delta(a) \in [\delta^{min}(a), \delta^{max}(a)]$$

C'est à dire : $\sigma(b) + \delta(b) + \sigma_b^{min}(v_j) \leq \sigma(a) + \delta(a) + \sigma_a^{max}(v_j)$, $\forall v_j \in V^b$, $b \in R(v_j)$, $a \in RRp(v_j)$

L'étude d'un sous-ensemble de combinaisons garantit qu'une contrainte de temps maximum est bien posée.

Théorème 2 : un graphe de contrainte faisable est bien posé *si et seulement si* :

$\sigma^{max}(b) + \delta^{max}(b) + \sigma_b^{min}(v_j) \leq \sigma^{min}(a) + \delta^{min}(a) + \sigma_a^{max}(v_j)$ avec $\forall v_j \in V^b$, $\forall a \neq b$, $b \in R(v_j)$, $a \in RRp(v_j)$.

Remarque : le cas $a = b$ est couvert par l'étude de faisabilité : $\sigma_a^{min}(v_j) \leq \sigma_a^{max}(v_j)$

Preuve : supposons, dans l'absurde qu'il existe un chemin critique $\sigma_b^{min}(v_i)$ *avec* $b \in R(v_i)$ tel que la contrainte est violée :

$\sigma^{max}(b) + \delta^{max}(b) + \sigma_b^{min}(v_j) \leq \sigma^{min}(a) + \delta^{min}(a) + \sigma_a^{max}(v_j)$ avec $\forall v_j \in V^b$, $\forall a \neq b$, $b \in R(v_j)$, $a \in RRp(v_j)$.

Nous démontrons par construction que le théorème 2 est nécessaire et suffisant. Supposons un graphe de contrainte global *GCG* contenant deux points de synchronisation référence tel que décrit dans la Figure 3-30.

Les dates de transfert des nœuds *a* et *b* sont :

$$\sigma(v_0) + \delta(v_0) + \sigma_{v0}^{min}(a) \leq \sigma(a) \leq \sigma(v_0) + \delta(v_0) + \sigma_{v0}^{max}(a)$$

$$\sigma(v_0) + \delta(v_0) + \sigma_{v0}^{min}(b) \leq \sigma(b) \leq \sigma(v_0) + \delta(v_0) + \sigma_{v0}^{max}(b)$$

La date d'exécution du nœud v_j doit vérifier les relations suivantes:

$\sigma(v_i) \geq Max_{\ r \in R(vi)} \{\sigma(r) + \delta(r) + \sigma_r^{min}(v_i)\}$

$\sigma(v_i) \leq Min_{q \in RRp(vi)} \{\sigma(q) + \delta(q) + \sigma_q^{max}(v_i)\}$

qui correspond pour le nœud de synchronisation *b* à :

$$\sigma(v_i) \geq \sigma^{max}(b) + \delta^{max}(b) + \sigma_b^{min}(v_i) \geq \sigma(v_0) + \delta(v_0) + \sigma_{v0}^{max}(b) + \delta^{max}(b) + \sigma_b^{min}(v_i) \quad \textbf{(1)}$$

et pour le nœud de synchronisation référence *a* à :

$\sigma^{min}(a) + \delta^{min}(a) + \sigma_a^{min}(v_i) \leq \sigma(v_i) \leq \sigma^{min}(a) + \delta^{min}(a) + \sigma_a^{max}(v_i)$

soit

$$\sigma(v_0)+\delta(v_0)+\sigma_{v0}^{min}(a)+\delta^{min}(a)+\sigma_a^{min}(v_i) \leq \sigma(v_i) \leq \sigma(v_0)+\delta(v_0)+\sigma_{v0}^{min}(a)+\delta^{min}(a)+\sigma_a^{max}(v_i) \quad \textbf{(2)}$$

Les inéquations (1) et (2) nous donnent l'inéquation suivante :

$\sigma(v_0) + \delta(v_0)+\sigma_{v0}^{max}(b)+ \delta^{max}(b) +\sigma_b^{min}(v_i) \leq \sigma(v_i) \leq \sigma(v_0)+\delta(v_0)+\sigma_{v0}^{min}(a)+ \delta^{min}(a)+\sigma_a^{max}(v_i)$

$\Leftrightarrow \sigma_{v0}^{max}(b) + \delta^{max}(b) + \sigma_b^{min}(v_i) \leq \sigma(v_i) \leq \sigma_{v0}^{min}(a)+ \delta^{min}(a)+\sigma_a^{max}(v_i)$

Un raisonnement similaire peut être appliqué pour les cas où la date de transfert de b dépend de la date de transfert de *a* et inversement.

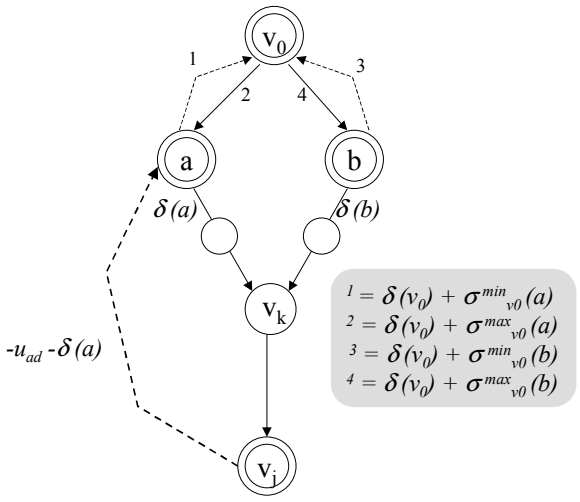


Figure 3-30 : Exemple de *GCG*

4. Conclusion

Dans ce chapitre nous avons proposé une modélisation des contraintes supportant l'expression de fenêtres temporelles pour les transferts, de caractéristiques de l'architecture de communication et de modes de transferts. Nous avons présenté un ensemble de phases d'analyse effectuées avant la synthèse. Ces différentes étapes permettent de vérifier que les contraintes de synthèse de l'algorithme offre une solution au problème d'ordonnancement. Ainsi, lorsque l'intégrateur système impose des contraintes trop fortes sur la synthèse d'un l'algorithme, les phases d'analyse déterminent les nœuds entrées et sorties dont les contraintes sur les dates de transfert doivent être relâchées. La modélisation de l'indéterminisme de transfert que nous proposons peut être utilisée pour représenter l'indéterminisme de traitement (structure de contrôle) dans le modèle algorithmique de l'IP. Des travaux sur la modélisation du contrôle et la synthèse architecturale sont actuellement en cours au sein du LESTER. Il convient de noter que le formalisme précédemment décrit est aisément réutilisable dans divers environnements de conception. Il est donc indépendant de l'outil de synthèse *GAUT* qui a été utilisé afin de concrétiser et de valider la modélisation et l'approche proposée. Le chapitre suivant détaille la prise en compte des contraintes temporelles dans le flot de synthèse, d'une unité de traitement, de l'environnement logiciel *GAUT*.

Chapitre 4

Implémentation et Optimisation

Dans une première partie de ce chapitre, nous rappelons le modèle architectural des composants virtuels que nous ciblons. Nous présentons une technique de conception et d'optimisation de l'unité de traitement qui repose sur un ensemble de transformation formelle et d'ordonnancement du Graphe de Contraintes Globales GCG. Nous proposons ensuite une modélisation des contraintes générées par la synthèse de l'unité de traitement qui couplée avec la modélisation des contraintes d'intégration permettra à terme la génération de l'unité de communication. Dans une deuxième partie nous présentons l'environnement logiciel GAUT dans lequel ont été intégrés ces travaux de recherche. Nous détaillons dans un premier temps le flot de conception et les fondements théoriques sur lesquels est basée la version originale de l'outil. Nous décrivons ensuite l'ensemble des modifications qui ont du être apportées pour prendre en compte les contraintes temporelles durant la synthèse de l'unité de traitement.

1. Stratégies de synthèse

La synthèse de haut niveau est l'association de techniques de transformation d'une spécification comportementale, et d'un modèle générique cible pour l'implémentation dont nous rappelons l'architecture pour débuter ce chapitre.

Nous mettons en avant le problème du choix de l'architecture permettant de garantir le respect des contraintes relatives à la production des sorties. En effet, comme nous le détaillerons dans les paragraphes suivants, lorsque le graphe de contraintes comporte plusieurs points de synchronisation référence, il peut s'avérer nécessaire de partitionner le graphe *GCG* et de synthétiser séparément chacun des sous graphes.

1.1. Rappel du modèle architectural cible

1.1.1. Unités fonctionnelles

L'architecture des composants comportementaux que nous considérons est constituée, comme indiqué sur la Figure 4-1, de trois unités fonctionnelles : une unité mémoire *(UM)* que nous ne détaillerons pas dans ce chapitre, une unité de communication *(UCOM)* et une unité de traitement *(UT)*. Chaque unité possède sa propre unité de contrôle *(UC)*.

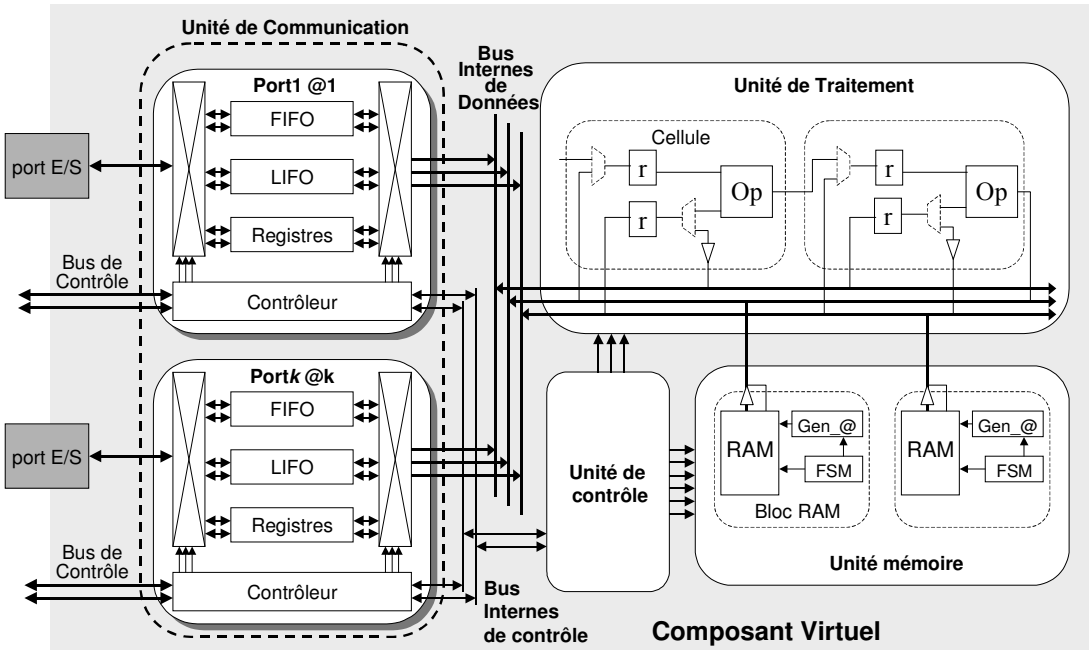


Figure 4-1 : Modèle architectural visé

L'unité de communication contient, des éléments mémorisants (FIFO, LIFO, registres), et leurs structures de contrôle associées, qui sont dédiées à chaque port d'E/S et ce en fonction des types des transferts (lecture ou écriture). Elle contient de plus un ensemble de FSMs : une FSM par

port, chacune implémentant le protocole d'échange de données spécifié par le concepteur du SoC et une FSM réalisant la synchronisation entre l'*UT* et l'*UCOM*.

L'unité de traitement *(UT)* implémente la partie calcul de l'algorithme en utilisant des cellules qui réalisent les opérations arithmétiques ou logiques élémentaires de la spécification algorithmique. Chaque cellule est composée d'un opérateur, d'un ensemble de registres, de multiplexeurs, de démultiplexeurs et de buffers trois états. Les registres permettent le stockage temporaire des données et la synchronisation des transferts de données au sein de l'*UT*. Les multiplexeurs, démultiplexeurs et buffers trois états ont en charge l'aiguillage des données. Leur présence optionnelle dans une cellule résulte du partage des registres et des opérateurs, déterminé par la phase d'optimisation de l'implémentation. Les cellules communiquent au travers d'un réseau de bus parallèles. Lors du processus de synthèse, et après les phases de sélection et d'allocation, l'étape d'ordonnancement affecte une date d'exécution à chacune des opérations arithmétiques ou logiques. Comme nous allons le voir dans la suite de ce chapitre, cette affectation est réalisée en tenant compte d'une part des dépendances de données et d'autre part des contraintes définies sur les entrées/sorties. Cette étape fondamentale permet de satisfaire les contraintes temporelles (E/S, Cadence, latence de l'application).

1.1.2. Unité de contrôle associée à l'*UT*

L'unité de contrôle, qui pilote l'*UT,* est constituée d'une machine d'états finis *linéaire* qui résulte de l'ordonnancement. En effet, l'étape d'ordonnancement affecte une date d'exécution, de façon équivalente une étape de contrôle (cstep), à chacune des opérations en tenant compte d'une part des dépendances de données et d'autre part des contraintes d'E/S. Toutefois, bien que la FSM contienne des états de synchronisation, elle suppose, par sa linéarité, un séquencement total, dans l'unité de traitement, des opérations utilisant des données d'entrée/sortie. Ce mode de contrôle requiert donc dans certains cas, un ordonnancement préalable des opérations d'E/S dans les graphes de contraintes *GCG* contenant plusieurs nœuds de synchronisation référence. Ainsi, comme nous le montrons dans les sections suivantes, lorsque aucun séquencement total n'est possible, le *GCG* doit être scindé en plusieurs sous graphes appelés *partitions*. Dans ce cas, le composant résultant est constitué d'un ensemble d'unités de traitement (chemins de données) qui fonctionnent en parallèle : chaque *UT* possède ses propres ressources matérielles et son propre contrôleur (FSM linéaire).

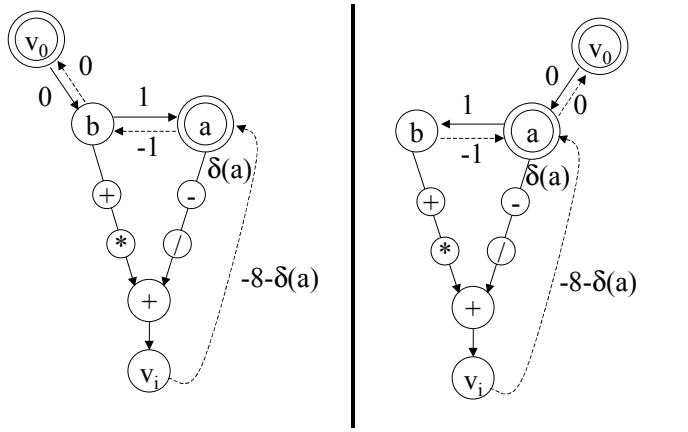
1.2. Introduction à l'implémentation

1.2.1. Cas 1 : point de synchronisation référence unique

Lorsque le cardinal de l'ensemble des nœuds de synchronisation référence est inférieur ou égal à 1 (hors v_0) alors *une* unique FSM et *un* unique chemin de données suffisent à garantir les contraintes temporelles d'intégration.

Afin de faciliter la lisibilité et la compréhension des prochains paragraphes, nous ne n'incluons pas le nœud source v_0 dans le dénombrement des nœuds de synchronisation référence. De plus, les graphes de contraintes considérés dans les exemples sont supposés bien posés.

La Figure 4-2(a) donne un exemple de graphe qui ne contient qu'une seule contrainte maximum sur la production de la donnée v_i et ce relativement au point de synchronisation référence *a* transmis après l'entrée *b* (*a* est transmis avant l'entrée *b* dans la Figure 4-2(b)).



Tp(+) = 1, Tp(-) = 1, Tp(*) = 3, Tp(/) = 6 $\quad \delta(a) \in [1, 4]$

(a) **(b)**

Figure 4-2 : Graphes contenant un unique point de synchronisation référence (hors v_0)

Quelle que soit la plage temporelle d'arrivée de l'entrée *a*, le chemin de données réalisant ce graphe peut être piloté par une FSM linéaire.

La Figure 4-3(a) présente le comportement temporel du graphe de la Figure 4-2(a) dans lequel l'entrée *b* (qui n'est pas un point de synchronisation) est utilisée à une date fixe après le début de l'exécution du calcul (représenté par le nœud v_0). La FSM (voir Figure 4-3(b)) comporte un état de synchronisation dont la transition est franchie dès l'arrivée de l'entrée *a*.

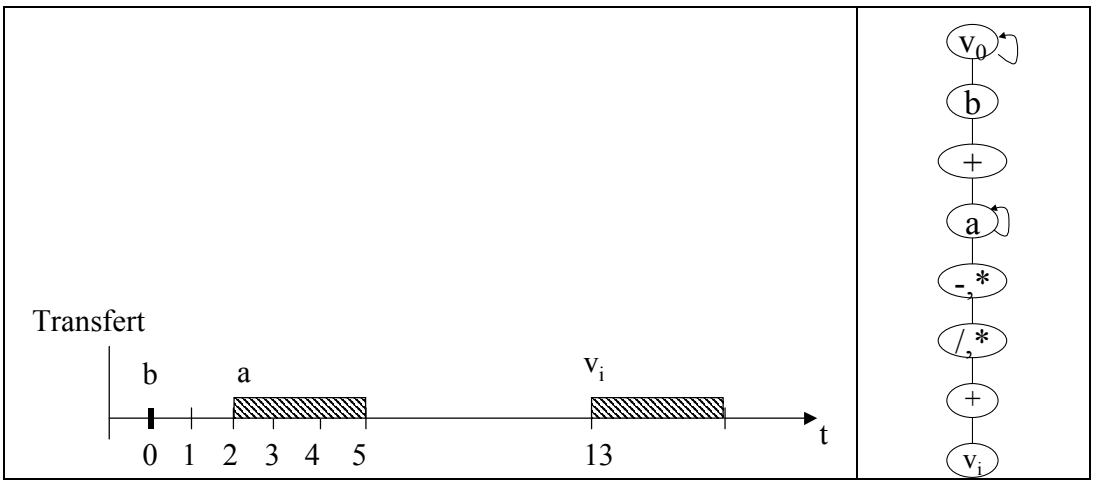


Figure 4-3: (a) Représentation temporelle des transferts de la Figure 4-2(a), (b) FSM linéaire

La Figure 4-4(a) présente le comportement temporel du graphe de la Figure 4-2(b) dans lequel le point de synchronisation référence *a* est utilisé à une date variable après le début de l'exécution du calcul (représenté par le nœud v_0). Le graphe étant bien posé, la FSM (voir Figure 4-4(b)) comporte un unique état de synchronisation dont la transition est franchie dès l'arrivée de l'entrée *a*. L'entrée *b* ne nécessite pas l'utilisation d'un état de synchronisation dans la FSM puisqu'elle est transférée à une date fixe relativement à la donnée *a*.

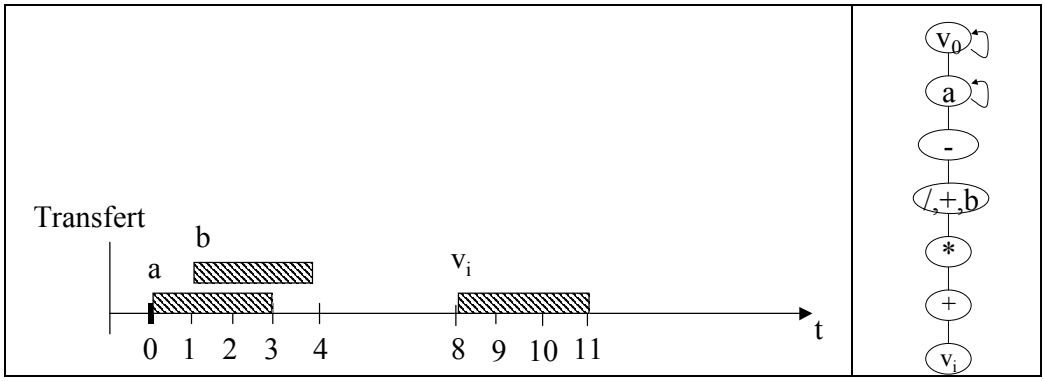


Figure 4-4: (a) Représentation temporelle des transferts Figure 4-2(b), (b) FSM linéaire

1.2.2. Cas 2 : ensemble de points de synchronisation référence

Lorsqu'il existe plusieurs points de synchronisation référence, l'utilisation d'une unique FSM linéaire pilotant un chemin de données peut, dans certains cas, aboutir à des architectures qui ne respectent pas les contraintes temporelles maximum relatives. Théoriquement, le nombre de point de synchronisation définit le nombre de partitions permettant de réaliser une architecture respectant les contraintes. Cependant, une étude des fenêtres temporelles d'arrivées des entrées et des fenêtres temporelles de production des sorties permet de réduire, lorsque cela est possible, le nombre de partition et ainsi de partager les ressources matérielles.

1.2.3. Sous cas 2.1 : séquencement naturel

Lorsque les fenêtres temporelles de production de sorties et les fenêtres temporelles d'arrivées des entrées sont disjointes c'est à dire que les contraintes d'E/S spécifient une suite de couple (point de synchronisation référence, sorties contraintes) alors la synthèse d'un unique chemin de données, pilotée par une unique FSM comportant plusieurs nœuds de synchronisation est réalisée.

La Figure 4-5 représente un graphe de contrainte dans lequel le transfert du point de synchronisation référence *b* est réalisé avant celui du point de synchronisation référence *a*. Les dates de production des données *c* et *d* sont spécifiées de façon relative respectivement aux points *b* et *a*. La production de la donnée *c* est requise avant l'acquisition du point de synchronisation référence *a* : il y a donc un séquencement total des données d'E/S.

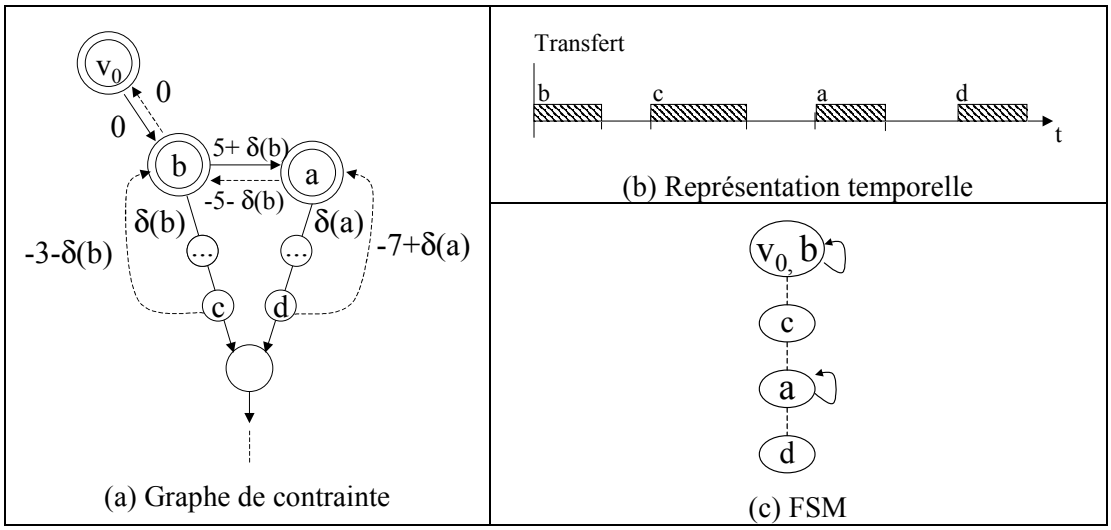


Figure 4-5: Exemple sous cas 2.1

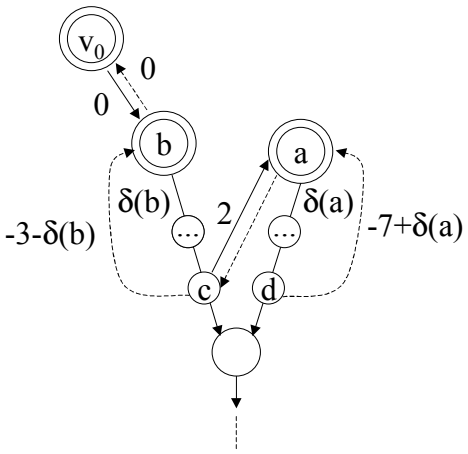


Figure 4-6: Séquence de transfert (b,c,a,d)

Notons que la séquence de transfert (b, c, a, d) aurait pu être modélisée de façon équivalente par le graphe décrit dans la Figure 4-6.

1.2.4. Sous cas 2.2 : séquencement imposé pour la synthèse

Lorsque les fenêtres temporelles de production et d'arrivées de données sont non disjointes et qu'un séquencement total est possible alors la synthèse d'un unique chemin de données, pilotée par une unique FSM comportant plusieurs nœuds de synchronisation peut aussi être réalisée. En effet, lorsqu'il n'existe pas de séquencement naturel de transfert des E/S, celui ci peut être créé en modifiant le graphe *GCG*.

La Figure 4-7(a) représente un graphe de contrainte dans lequel le transfert du point de synchronisation référence *b* est réalisé avant celui du point de synchronisation référence *a*. Comme dans l'exemple précédent, les dates de production des données *c* et *d* sont spécifiées de façon relative respectivement aux points *b* et *a*. La production de la donnée *c* est cependant requise à une date pouvant être contenue dans la fenêtre temporelle d'acquisition du point de

synchronisation référence *a*. Toutefois la contrainte temporelle sur la date de production de la donnée *c* autorise l'attente de l'entrée *a* : un séquencement des E/S est donc possible lors de la synthèse ce qui résulte sur une FSM linéaire comme le décrit la Figure 4-7(c).

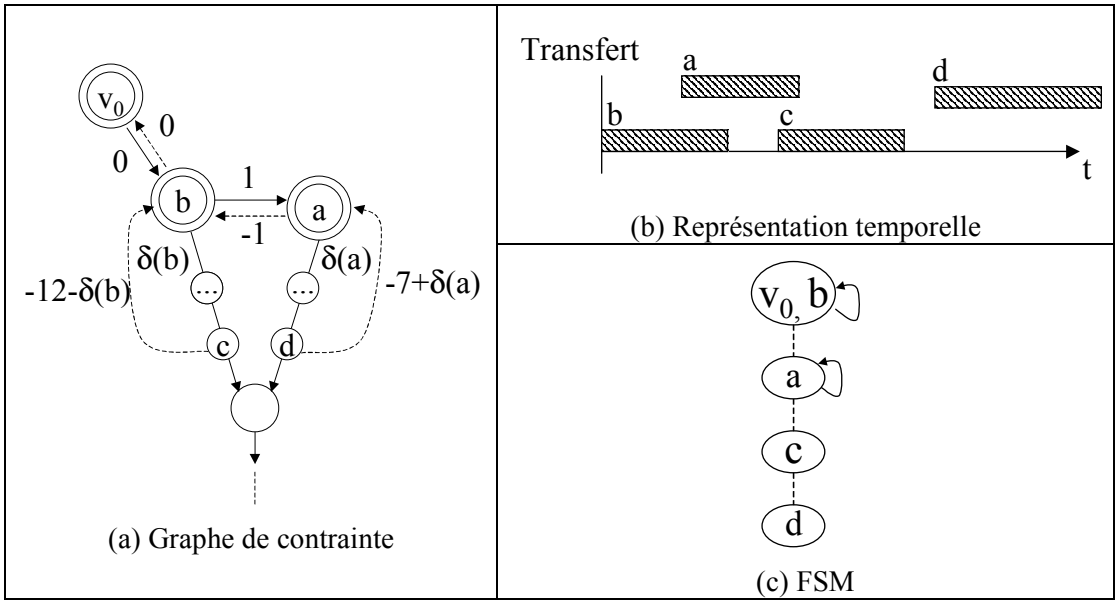


Figure 4-7: Exemple sous cas 2.2

1.2.5. Sous cas 2.3 : Séquencement impossible pour la synthèse

Lorsque les fenêtres temporelles de production et d'arrivée de données sont non disjointes et qu'aucun séquencement total n'est possible il faut partitionner le GFS représentant l'IP en plusieurs sous graphes. La Figure 4-8(a) décrit un graphe dans lequel le transfert de l'entrée *a* peut être effectué durant le transfert de la sortie *c* (Figure 4-8(b)). Toutefois, la contrainte spécifiée sur la date de production de la donnée *c* n'autorise pas l'attente de l'entrée *a*.

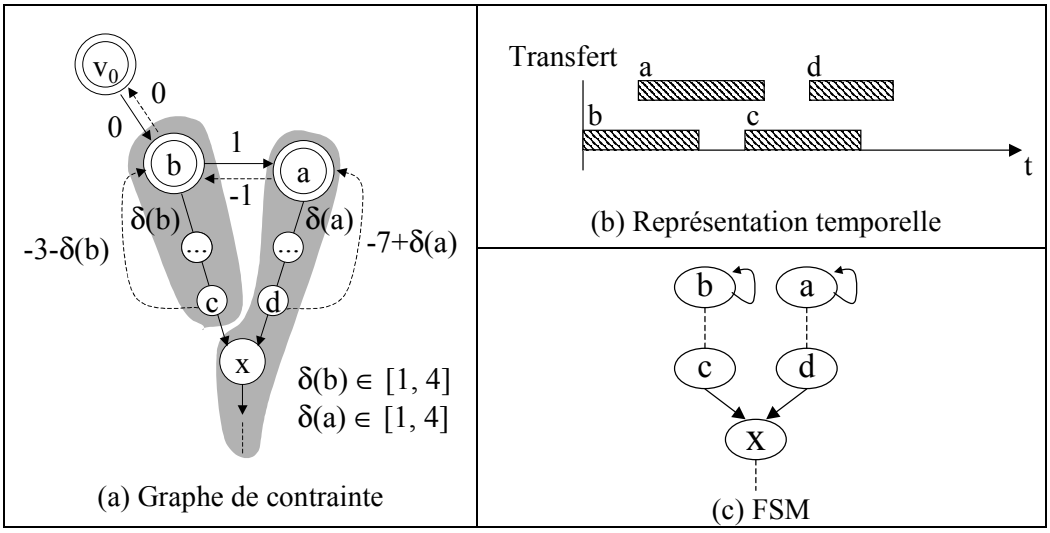


Figure 4-8: Exemple sous cas 2.3

Dans le cas ou le graphe doit être partitionné, chacun des sous-graphes est synthétisé pour produire un chemin de donné qui est piloté par une FSM dédiée. La synchronisation des différents contrôleurs est réalisée par l'utilisation de signaux *done* et *enable* comme le décrit la Figure. 4-9 : un nœud conjonction ne pourra donc s'exécuter que lorsque tous ses prédécesseurs auront terminé leur exécution.

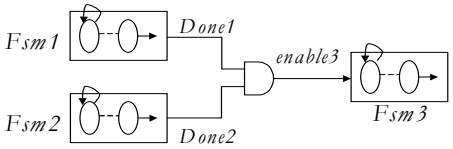


Figure. 4-9: Technique de synchronisation des FSMs

Ce modèle de contrôleur n'est actuellement pas implémentable dans GAUT à cause des problèmes d'allocation, d'ordonnancement et d'assignation. Les modifications nécessaires à la réalisation d'un contrôle distribué seront réalisées dans des travaux futurs concernant la synthèse par parties.

1.3. Implémentation du GCG

1.3.1. Problématique

Certains nœuds de synchronisation référence peuvent être utilisés dans l'unité de traitement à une date supérieure à leur date maximum d'arrivée, et ce tout en garantissant le respect des contraintes imposées sur les dates de production des résultats. Cette réduction du nombre de points de synchronisation permet dans certains cas d'éviter une synthèse par parties de l'algorithme. En effet, la suppression de certains nœuds de synchronisation référence permet un séquencement de l'utilisation des données d'E/S dans l'UT et donc l'utilisation d'une FSM linéaire pour piloter l'architecture. Les figures suivantes illustrent les effets de la suppression d'un point de synchronisation sur la machine d'état pilotant le chemin de données.

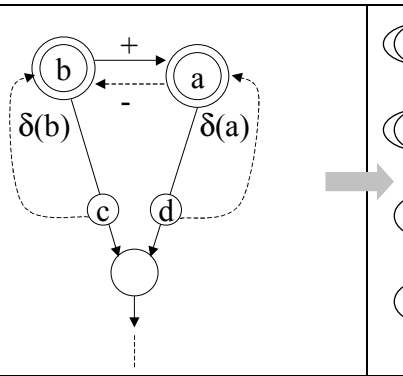


Figure 4-10 : GCG

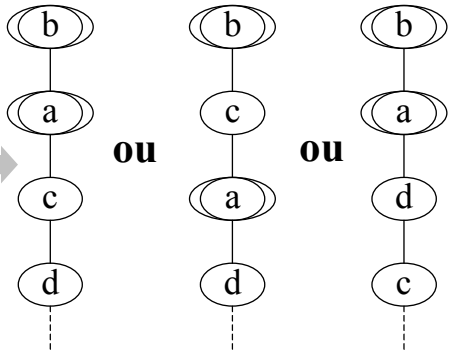


Figure 4-11 : FSMs linéaires possibles

L'implémentation linéaire d'un *GCG* nécessite donc la définition d'un critère permettant la sélection des nœuds de synchronisation référence candidats pour une suppression. Elle requiert de plus un ensemble de transformations formelles du graphe afin **d'ordonnancer** les opérations d'E/S.

1.3.2. Ordonnancement du *GCG*

1.3.2.1 Critère de suppression d'un nœud de synchronisation : la redondance

Une nouvelle propriété concernant les nœuds de synchronisation référence est introduite : la *redondance*. Cette notion va servir par la suite, de critère à la suppression de ce type de nœud et donc à la *"sérialisation"* du graphe de contraintes. Les nœuds de synchronisation référence redondants peuvent être traités par l'UT à une date s telle que $\sigma(a) \geq \sigma^{max}(a) + \delta^{max}(a)$.

Définition : un nœud de synchronisation référence pertinent pour un nœud v_i est *redondant* s'il existe un autre nœud de synchronisation pertinent pour un nœud v_i qui contraint plus la date d'implémentation de v_i.

Théorème 3 : soit a et b deux nœud de synchronisation référence pertinent de v_i, $b \in R(v_i) \cap RRp(v_i)$ est *redondant* par rapport à a si et seulement si :

$$\sigma^{max}(b)+\delta^{max}(b)+\sigma_b^{min}(v_i) \leq \sigma^{min}(a)+\delta^{min}(a)+\sigma_a^{min}(v_i)$$

et

$$\sigma^{max}(a)+\delta^{max}(a)+\sigma_a^{max}(v_i) \leq \sigma^{min}(b)+\delta^{min}(b)+\sigma_b^{max}(v_i)$$

avec $v_i \in V^b$, $a \in RRp(v_i)$, $b \in R(v_i) \cap RRp(v_i)$

Autrement, b est un nœud de synchronisation non redondant de v_i.

L'ensemble des nœuds de synchronisation non redondants de v_i est dénoté par $IR(v_i) \subseteq Rp(v_i)$.

Preuve : la date d'exécution de v_i est (cf chapitre 3 §2.6.3) par rapport à a

$$\sigma(a) + \delta(a) + \sigma_a^{min}(v_i) \leq \sigma(v_i) \leq \sigma(a) + \delta(a) + \sigma_a^{max}(v_i)$$

et par rapport à b

$$\sigma(b) + \delta(b) + \sigma_b^{min}(v_i) \leq \sigma(v_i) \leq \sigma(b) + \delta(b) + \sigma_b^{max}(v_i)$$

le nœud v_i n'est contraint que par a si et seulement si les contraintes imposées par le nœud a sont plus fortes que celles imposées par le nœud b c'est à dire si

$$\sigma(b) + \delta(b) + \sigma_b^{min}(v_i) \leq \sigma(a) + \delta(a) + \sigma_a^{min}(v_i)$$

et

$\sigma(a) + \delta(a) + \sigma_a^{max}(v_i) \leq \sigma(b) + \delta(b) + \sigma_b^{max}(v_i)$

donc en particulier si et seulement si

$\sigma^{max}(b) + \delta^{max}(b) + \sigma_b^{min}(v_i) \leq \sigma^{min}(a) + \delta^{min}(a) + \sigma_a^{min}(v_i)$

et

$\sigma^{max}(a) + \delta^{max}(a) + \sigma_a^{max}(v_i) \leq \sigma^{min}(b) + \delta^{min}(b) + \sigma_b^{max}(v_i)$

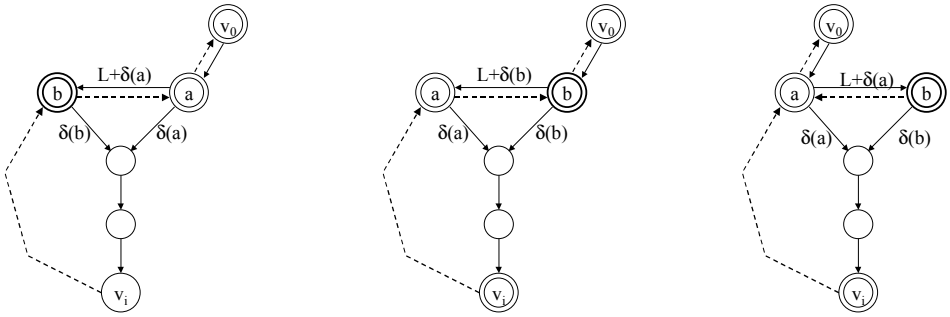
Remarque : un nœud de synchronisation *référence b* peut servir de référence à plusieurs contraintes maximum i.e. $b \in Rp(v_j) \cap Rp(v_i)$. Il peut ainsi être redondant pour un nœud v_i (v_j) et non redondant pour un nœud v_j (v_i) *i.e.* $b \notin IR(v_i) \cap IR(v_i)$.

Définition : un nœud est *redondant absolu* s'il est redondant pour tous les nœuds de sortie auxquels il sert de référence.

Théorème 4 : soit *GCG(V, E)* un graphe de contrainte bien posé. Tout nœud de synchronisation pertinent référence non pertinente redondant *b* peut être supprimé si et seulement si $b \notin IR(v_j)$ et ce $\forall v_j \in V^b$.

I.e. le nœud doit être *redondant absolu*.

Le critère de suppression des nœuds que nous venons de définir ne vise que les nœuds de synchronisation référence. En effet, l'analyse du *GCG,* permettant d'affirmer qu'il existe un ordonnancement si le graphe est bien posé, est réalisée en fixant les valeurs des dates d'arrivées des **nœuds de synchronisation** à leur maximum. Ainsi, quelle que soit la date réelle d'arrivée d'un point de synchronisation aucune contrainte temporelle maximum n'est violée : un nœud de synchronisation **non référence**, peut toujours être supprimé : le nœud de synchronisation *b* de la Figure 4-12(c)) peut être supprimé car il n'appartient pas à l'ensemble des nœuds de référence *Rp* du nœud v_j.

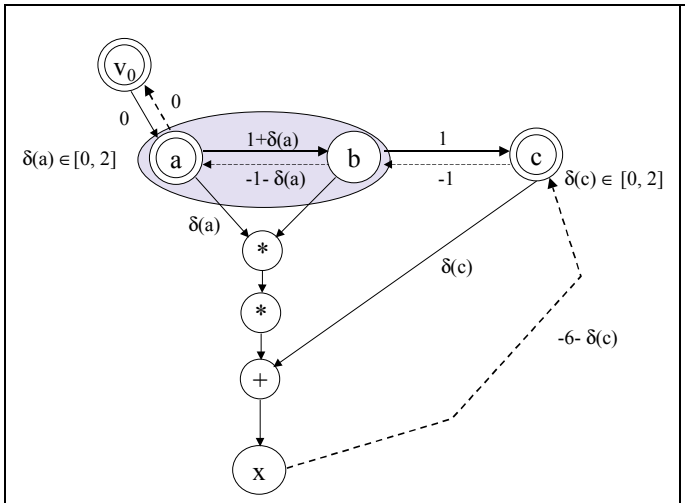


(a) **(b)** **(c)**

Figure 4-12 : b : nœud de synchronisation pertinent (a) référence pertinente, (b) référence **non** pertinente et (c) non référence

1.3.2.2 Exemple de suppression de nœud

Les Figure 4-13 et Figure 4-14 illustrent les modifications opérées dans le graphe lors de la suppression d'un nœud de synchronisation référence *c*.



Rp(c) = {a, v_0} S(c) = {a, v_0}

Rpp(c) = {a} R(c) = {a}

Rp(x) = {c, a, v_0} S(x) = {a, v_0, c}

Rpp(x) = {c} R(x) = {a, c}

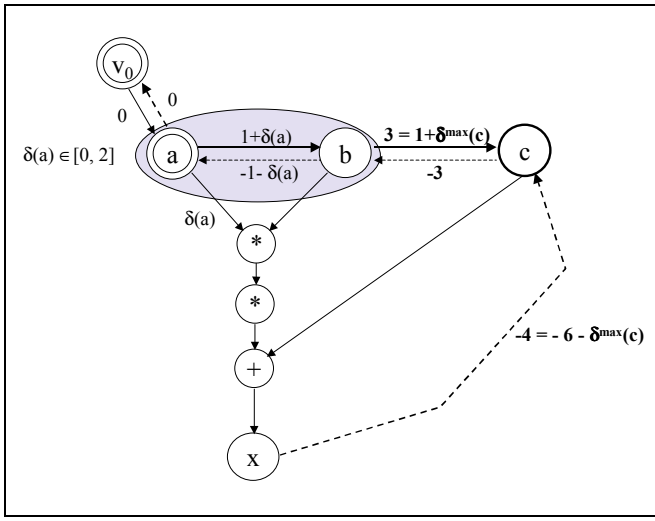
$\delta^{max}(c) = 2$

$\sigma_c^{max}(x) = -6 - \delta(c)$

Figure 4-13 : GCG avant suppression du nœud de synchronisation C

La date d'exécution du nœud *x* dans la figure précédente est :

$\sigma(x) \geq \sigma(a) + \delta(a) + 2*t(*) + t(+)$ et $\sigma(c) + \delta(c) + t(+) \leq \sigma(x) \leq \sigma(c) + \delta(c) + 6$



Rp(c) = {a, v_0} S(c) = {a, v_0}

Rpp(c) = {a} R(c) = {a}

Rp(x) = {a, v_0} S(x) = {a, v_0}

Rpp(x) = {a} R(x) = {a}

$\sigma_c^{max}(x) = -6 - \delta^{max}(c) = -4$

Figure 4-14 : GCG après suppression du nœud de synchronisation C

La date d'exécution du nœud *x* dans la figure précédente est

$$\sigma(c) + \delta^{max}(c) + t(+) \leq \sigma(x) \leq \sigma(c) + 6 - \delta^{max}(c)$$

1.3.2.3 Transformation du graphe lors de la suppression d'un nœud

La suppression d'un nœud de synchronisation *c* demande une modification du graphe de contrainte constituée des transformations suivantes :

(1) Modification du poids des a*rcs avant (b, c)* pour tout nœud ayant une connexion avant avec *c* : la valeur $\delta^{max}(c)$ est ajoutée au poids positif w_{bc}.

(2) Modification du poids des *arcs arrière (c, b)* pour tout nœud ayant une connexion arrière avec *c* : la valeur $-\delta^{max}(c)$ est ajoutée au poids négatif w_{cb}.

(3) Modification du poids des *arcs arrière* (v_i, *c)* pour tout nœud v_i ayant *c* dans son ensemble de nœud référence pertinent $\forall v_i \text{ tel que } c \in RRp(v_j) : \delta(c) = -\delta^{max}(c)$

(4) Modification du poids des *arcs avant* (*c*, v_i) pour tout nœud v_i ayant *c* dans son ensemble de nœud de synchronisation pertinent $\forall v_i \text{ tel que } c \in Rp(v_j) : \delta(c) = 0$

(5) Mise à jour de l'ensemble des nœuds de synchronisation pertinents des nœuds v_j tel que $c \in R(v_j) : R(v_j) = R(v_j) \setminus c + R(c)$

(6) Mise à jour de l'ensemble des nœuds de référence pertinents des nœuds v_j tel que $c \in RRp(v_j) : RRp(v_j) = RRp(v_j) \setminus c + RRp(c)$

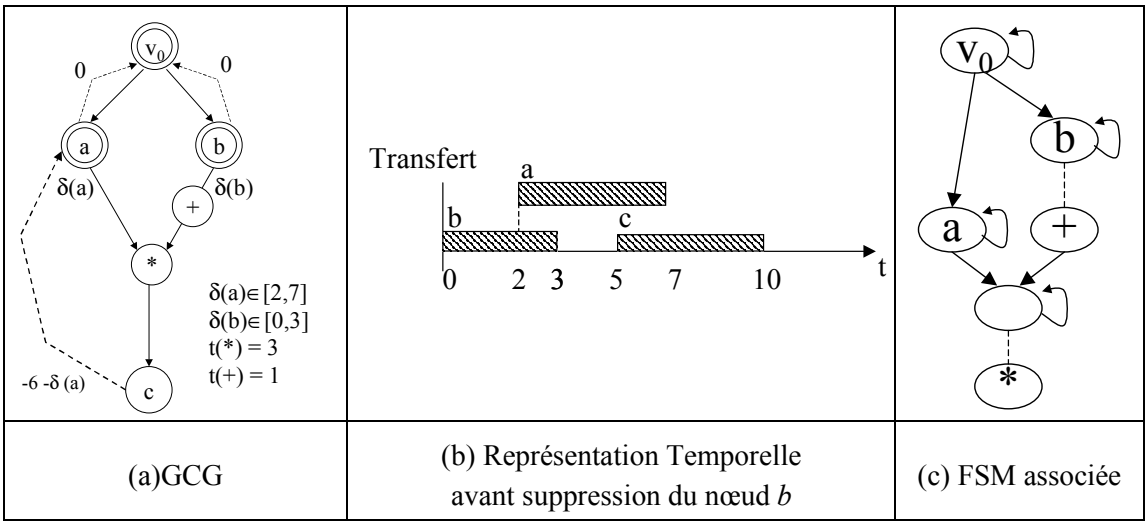
(7) Mise à jour de l'ensemble des nœuds de synchronisation $\{S\} = \{S\} \setminus c$

(8) Mise à jour de l'ensemble des nœuds de référence $\{Rp\} = \{Rp\} \setminus c$

1.3.3. Sérialisation pour l'obtention d'une FSM linéaire

Bien qu'il soit nécessaire, le critère de redondance n'est pas suffisant dans certains cas pour réaliser une architecture contenant une unique partition pilotée par une FSM linéaire. En effet, certaines spécifications des échanges entre l'IP et le système requièrent une *sérialisation* des transferts des données dans le *GCG. Cette transformation générique de forte complexité n'est pas formalisée dans ce document et sera réalisée dans de futurs travaux de recherche.* Afin d'illustrer le problème, nous présentons dans cette section un exemple où seule la suppression d'un point de synchronisation ne permet pas l'utilisation d'un contrôleur linéaire.

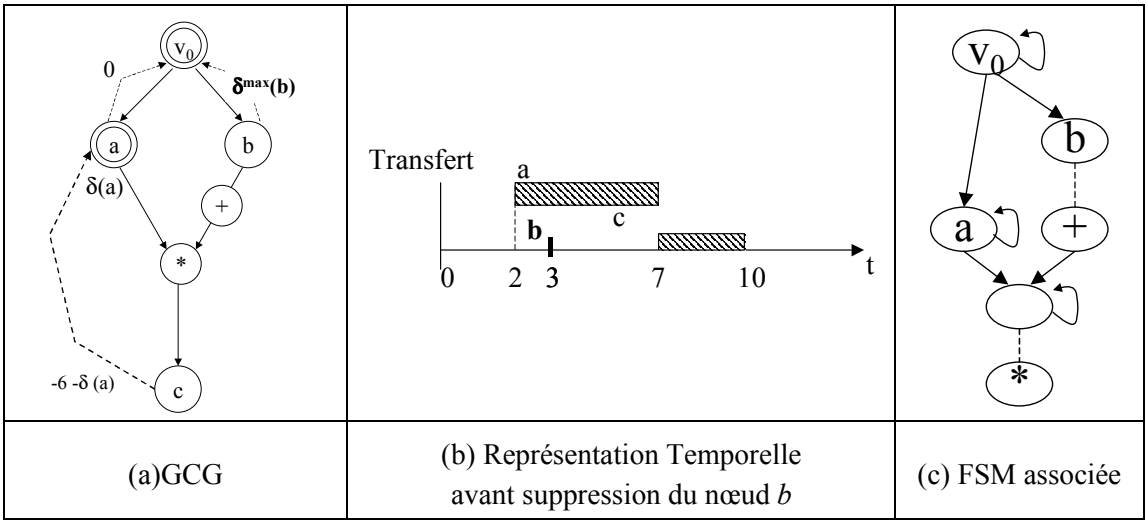
Ainsi, dans la Figure 4-15, le nœud entrée *b* peut être supprimé puisqu'il est point de synchronisation.



(a)GCG	(b) Représentation Temporelle avant suppression du nœud *b*	(c) FSM associée

Figure 4-15 : Exemple avant suppression d'un nœud de synchronisation

La Figure 4-16 décrit le *GCG* après suppression du nœud de synchronisation *b*. Cette suppression ne permet cependant pas le pilotage du chemin de données par une FSM linéaire. En effet, l'attente du point de synchronisation référence *a* est durant l'exécution d'une partie des traitement dépendant de l'entrée *b (voir* Figure 4-16*(b) et (c)).*

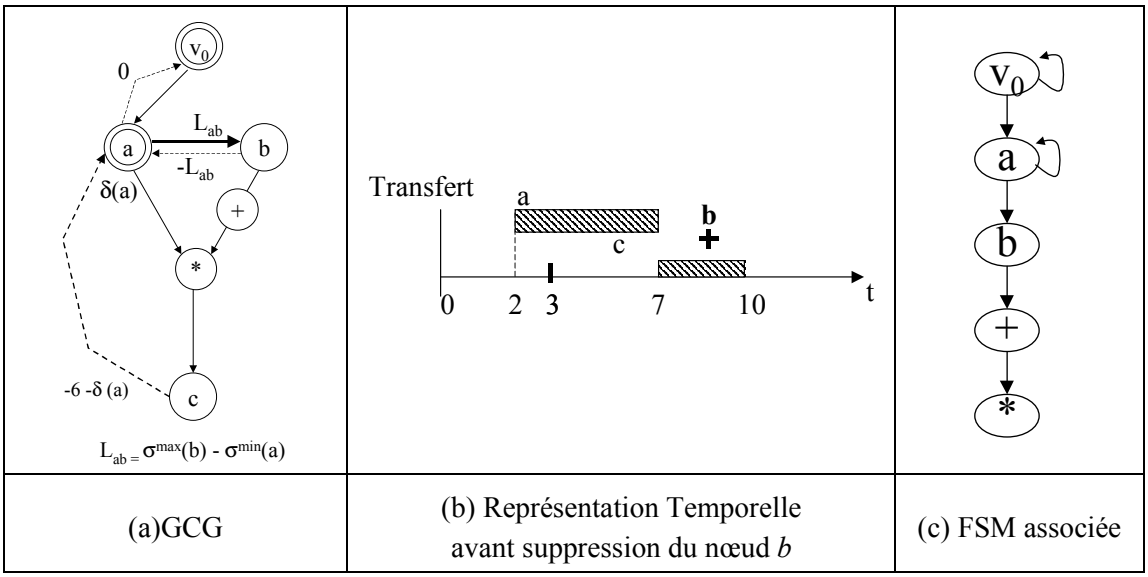


(a)GCG	(b) Représentation Temporelle avant suppression du nœud *b*	(c) FSM associée

Figure 4-16 : Exemple après suppression d'un nœud de synchronisation

L'utilisation d'une machine d'état linéaire demande une modification supplémentaire du GCG : la *sérialisation* des nœuds *a* et *b*. La Figure 4-17 illustre les modifications apportées au *GCG* en terme de connexion et de pondération des connexions. Le retard, impliqué par le séquencement de l'opération d'acquisition de *a* sur la date d'exécution de l'opération d'addition utilisant l'entrée *b,* autorise toujours la production du résultat *c* à une date inférieure à la date imposée par *a*

$$\sigma(c) \leq \sigma(a) + \delta(a) + \sigma_a^{max}(c)$$



(a)GCG	(b) Représentation Temporelle avant suppression du nœud *b*	(c) FSM associée

Figure 4-17 : Exemple après sérialisation du *GCG*

Comme nous l'avons montré, le nombre de partition peut être réduit en supprimant certains points de synchronisation référence et en sérialisant le *GCG*. Ces transformations, qui minimisent directement le nombre de partitions, réduisent en contrepartie la mobilité des opérations que ces partitions contiennent : la synthèse doit respecter des contraintes plus fortes et peut ainsi générer une architecture dont le coût matériel est supérieur à une architecture contenant un plus grand nombre de chemins de données. Inversement, comme nous le montrons dans la section suivante, la synthèse par parties peut, elle aussi, générer une architecture dont le coût est supérieur à la synthèse d'une seule partition. Ce surcoût matériel vient du non partage des ressources entre les différentes partitions.

1.3.4. Exemple de synthèse par parties

Comme nous le montrons maintenant sur l'exemple d'un filtre elliptique du cinquième ordre [DEW85], la synthèse en plusieurs partitions d'un algorithme aboutit à un accroissement du matériel devant être mis en œuvre pour réaliser l'architecture finale. La description algorithmique contient 8 opérations de multiplications, 26 opérations d'additions et 16 opérations d'E/S. Les dépendances de données sont montrées dans la Figure 4-18. La description initiale non partitionnée a un chemin critique de 17 cycles en supposant les temps d'exécution des additionneurs et des multiplieurs de la bibliothèque respectivement égaux à 1 et 2 cycles. Nous avons supposé des valeurs de contraintes nécessitant respectivement une synthèse de cet algorithme en 1, 2 et 3 partitions.

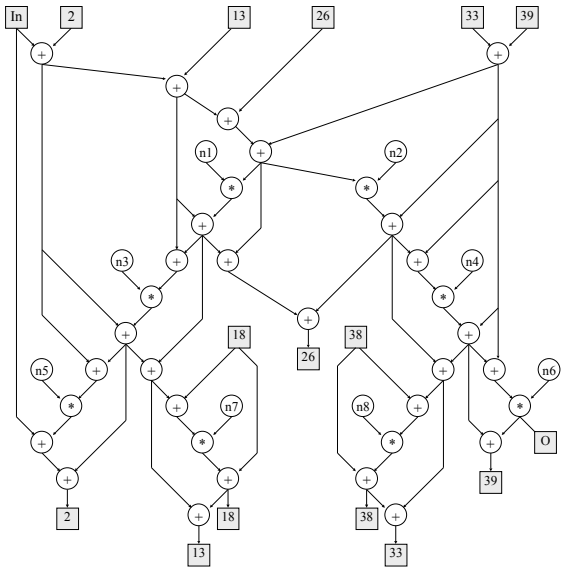


Figure 4-18: Partition unique

Ainsi, la Figure 4-19 présente un graphe *IOCG* qui contient un unique point de synchronisation référence pertinent (nœud 38) permettant la spécification relative de la date de production du nœud *O*. Les contraintes de production des autres résultats sont toutes exprimées relativement à cette première sortie *O*.

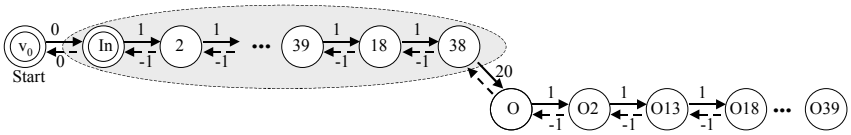


Figure 4-19: Contraintes générant une partition

La Figure 4-20 contient, quant à elle, deux points de synchronisation référence pertinents (nœud 39 et 38) permettant respectivement la spécification relative des dates de production des nœuds *O* et *O18*. Les contraintes de production des autres résultats sont toutes exprimées relativement à ces deux nœuds sortie.

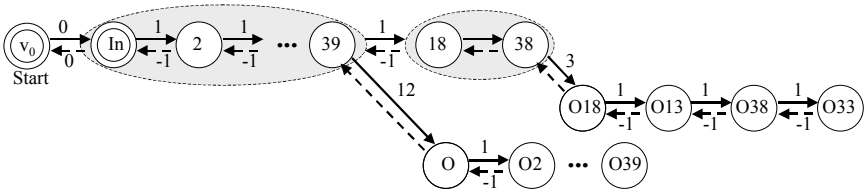


Figure 4-20: Contraintes générant deux partitions

La Figure 4-21 illustre le résultat du partitionnement du filtre en deux parties.

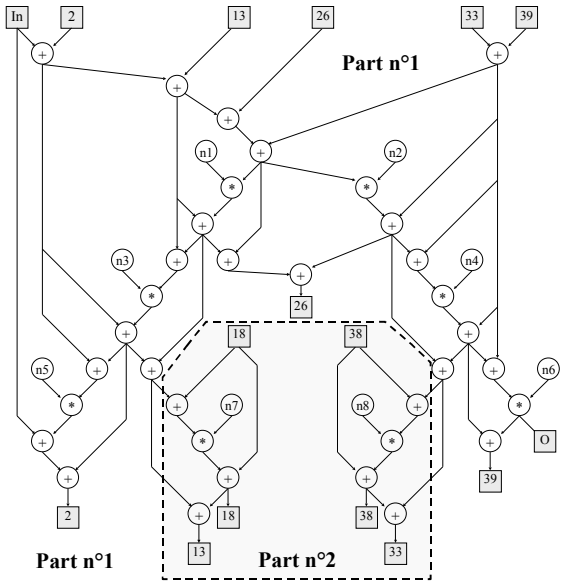


Figure 4-21: Deux partitions

La Figure 4-22 contient trois points de synchronisation référence pertinents (nœud 39, 18 et 38) permettant respectivement la spécification relative des dates de production des nœuds *O*, *O18* et *O38*. Les contraintes de production des autres résultats sont toutes exprimées relativement à ces trois nœuds sortie.

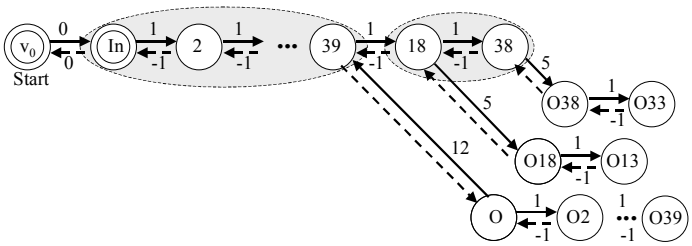


Figure 4-22: Contraintes générant trois partitions

La Figure 4-23 illustre le résultat du partitionnement du filtre en trois parties.

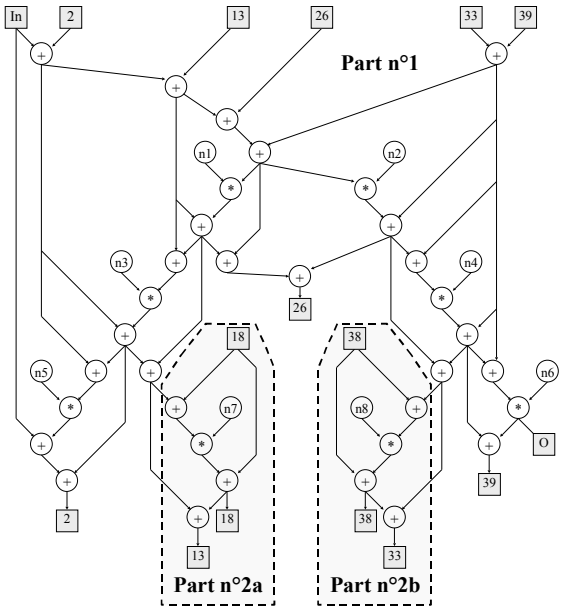


Figure 4-23: Trois partitions

Le Tableau 4-1 présente les résultats de synthèse en nombre d'opérateurs et de registres contenus dans les différentes partitions. Le partitionnement du graphe en un ensemble de sous graphe permet le respect des contraintes d'intégration mais augmente le coût de l'architecture finale. Il en résulte une augmentation du temps d'exécution du chemin critique (latence pour produire les sorties 13, 18, 33, 38) puisque les registres permettant la synchronisation entre les partitions ajoutent un cycle de latence.

Nombre de partition		**Additionneur.**	**Multiplieur.**	**Registre**	**Chemin critique**
1	**Total**	**3**	**1**	**30**	**17**
2	*Part 1*	2	1	21	16
	Part 2	2	1	12	5
	Total	**4**	**2**	**33**	**18**
3	*Part 1*	2	1	21	16
	Part 2a	1	1	6	5
	Part 2b	1	1	6	5
	Total	**4**	**3**	**33**	**18**

Tableau 4-1 : Résultat de synthèse du filtre elliptique

1.4. Conclusion

L'étude que nous avons présentée dans cette section a mis en avant la nécessité d'un choix de conception : synthèse d'une partition unique versus synthèse par parties. En effet, dans certains cas, la synthèse par parties augmente le nombre d'opérateurs puisque les ressources ne sont pas partagées entre les partitions. Cependant, bien que la réduction du nombre de partitions permette de minimiser le coût des ressources matérielles de l'architecture, elle peut aussi avoir un coût supérieur à la synthèse d'un plus grand nombre de partitions.

Bien que le critère de partitionnement soit présenté dans ce manuscrit, nous n'avons pas intégré la synthèse par parties d'un algorithme dans l'outil GAUT. Ainsi, dans le cadre du travail effectué durant cette thèse et intégré dans l'outil GAUT, seuls les graphes contenant un unique point de synchronisation après les phases de transformation et de suppression de point de synchronisation ont été étudiés. Nous n'avons pas tenu compte du surcoût engendré par la suppression des points de synchronisation référence. La sélection de l'architecture obtenue en supprimant les points de synchronisation ou générée par une synthèse par partie sera considérée dans des travaux futurs.

2. Conception des unités fonctionnelles

Cette section présente une technique permettant une synthèse de l'unité de traitement et de l'unité de communication qui respecte les contraintes d'intégration.

2.1. Synthèse de l'unité de traitement

En plus des contraintes spécifiées par l'intégrateur système, la synthèse de l'unité de traitement requiert la prise en compte des temps d'accès aux éléments mémorisants de l'unité de communication et ce pour une opération de lecture et d'écriture des E/S (voir Figure 4-24).

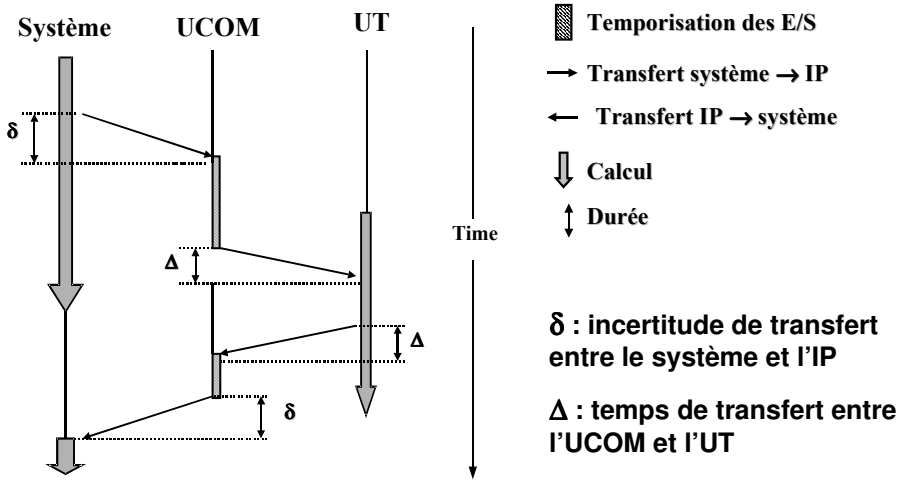


Figure 4-24 : Temps de transfert Système / IP

Exemple introductif :

La Figure 4-25 illustre les plages temporelles des E/S dans l'unité de communication pour le graphe de contraintes *GCG* présenté dans la Figure 4-14.

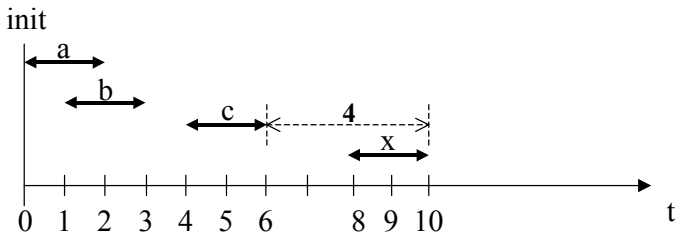


Figure 4-25 : Plages temporelles de transfert

Comme décrit dans la Figure 4-26, les dates d'utilisation et de production considérées pour la synthèse de l'unité de traitement doivent prendre en compte les temps d'accès à l'unité de communication.

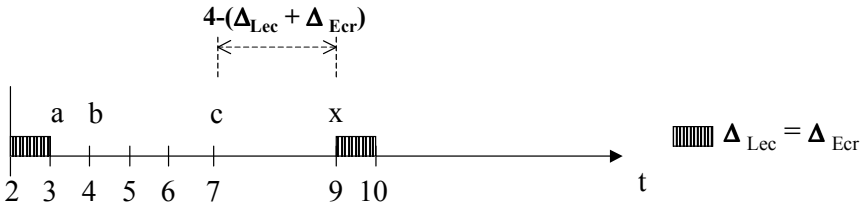


Figure 4-26 : Dates de consommation et de production considérées pour la synthèse l'UT

2.1.1. Temps d'accès à l'unité de communication

La synthèse (par partie ou non) de l'unité de traitement est réalisée en fonction de la mobilité des opérations. Cette mobilité est fournie par le calcul des dates ASAP et ALAP des E/S du CGC réduit. Les dates d'utilisation et de production des E/S dans l'unité de traitement découlent de l'inéquation introduite dans le théorème 2 (voir section 3.2 du chapitre 3) garantissant qu'un graphe de contraintes faisable est bien posé. Les dates de consommation et de production utilisées pour cette vérification, représentent le pire cas de contraintes sur les d'E/S. Elles permettent de générer un ensemble de dates ASAP et ALAP utilisées pour le calcul des mobilités et ainsi de garantir la synthèse d'une architecture respectant l'ensemble des contraintes. Ainsi,

- les nœuds de synchronisation non référence sont utilisés au plus tôt dans l'unité de traitement (*ASAP_UT*) à leur date ALAP calculée relativement à leur prédécesseur dans le graphe de contraintes GCG réduit (*ALAP_UCOM*);

- les nœuds de synchronisation référence sont utilisés au plus tôt dans l'unité de traitement (*ASAP_UT*) à leur date ASAP calculée relativement à leur prédécesseur dans le graphe de

contraintes *GCG* réduit (*ASAP_UCOM*). Ceci permet d'exprimer la mobilité la plus réduite pour les opérations;

- les sorties sont produites au plus tard par l'unité de traitement (*ALAP_UT*) à leur date ALAP calculée relativement à leur de nœud de référence dans le graphe contraintes *GCG* réduit (*ASAP_UCOM*).

Le temps d'accès aux données, échangées entre l'unité de communication et l'unité de traitement, est modélisé par une latence d'écriture et de lecture : Δ_{ECR} et Δ_{LEC}. Les transformations pour le calcul des dates *ASAP_UT* et *ALAP_UT* sont donc les suivantes :

Pour les nœuds entrée :

- $ASAP\_UT(a) = ALAP\_UCOM(a) + \Delta_{LEC} \ \forall a \in R(v_i), v_i \in I$

- $ASAP\_UT(a) = ASAP\_UCOM(a) + \Delta_{LEC} \ \forall a \in RRp(v_i), v_i \in I$

Pour les nœuds sortie :

- $ALAP\_UT(v_j) = ASAP\_UCOM(v_j) - \Delta_{ECR} \ \forall v_j \in O$

2.1.2. Contraintes liées à la synthèse de l'unité de traitement

- **La mémorisation des données d'entrée/sortie :** dans le cas où, le mode de communication du composant avec les autres composants du système est asynchrone, le stockage des données échangées avec l'extérieur doit être effectué dans des mémoires tampons. Deux approches de synthèse sont possibles. Dans la première, la conception et l'optimisation de ces mémoires sont entièrement assurées par l'unité de communication. Dans la deuxième, leur conception est réalisée avec le processus de synthèse de l'unité de mémorisation c'est à dire avec les mémoires de données hors E/S traitées par l'UT. Cette dernière approche est avantageuse car elle vise une optimisation globale des éléments mémoires. Cependant sa mise en œuvre devient vite complexe avec le non déterminisme des échanges externes.

- **La structure des données :** les données d'entrée/sortie sont souvent structurées (i.e scalaire ou vectoriel) et les messages échangés entre le composant et le reste du système peuvent avoir des formats variables (sous forme de trames). Ainsi, les fonctionnalités de conversion de format sont laissées à la charge de l'unité de communication.

- **Protocoles de communication :** d'une façon similaire, l'implantation et la gestion des protocoles de communication seront réalisées dans l'unité de communication.

2.2. Synthèse de l'unité de communication

La synthèse de l'unité de communication doit dimensionner le nombre de registres nécessaire à la temporisation des données d'E/S. Afin de réaliser une communication sans perte, le calcul des durées de vie des données dans l'unité de communication est réalisée de la façon suivante :

Les entrées sont mémorisées dans l'unité de communication de leur *ASAP_UCOM* jusqu'à leur dernière date d'utilisation T_{ut} dans l'unité de traitement. Les sorties sont mémorisées dans l'unité de communication depuis leur date de production T_{ut} définie par la synthèse de l'unité de traitement jusqu'à *ALAP_UCOM*. La Figure 4-27 illustre les plages temporelles de mémorisation pour l'exemple de la Figure 4-25.

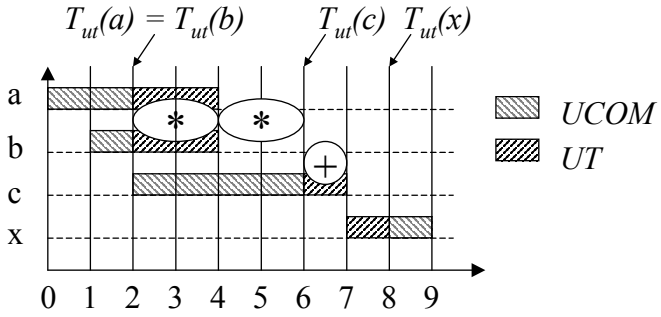


Figure 4-27 : Plage de mémorisation des E/S dans l'UCOM

La synthèse de l'*UCOM* utilise donc les contraintes d'E/S spécifiées dans le *IOCG* par l'intégrateur système et les contraintes provenant de la synthèse de l'*UT*.

2.2.1. Flot de conception

Comme nous l'avons décrit dans le chapitre précédent et le rappelons dans la Figure 4-28, la synthèse de l'unité de communication est réalisée a posteriori de celle de l'unité de traitement dans le flot de conception basé sur l'utilisation du logiciel *GAUT*.

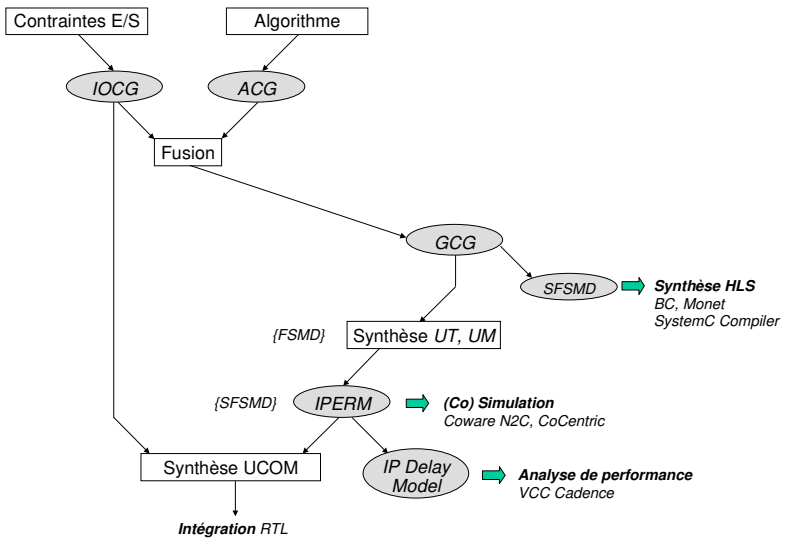


Figure 4-28 : Rappel du flot de conception

Afin de concevoir l'unité de communication, nous proposons un modèle *IPERM (IP Execution Requirements Model)* pour la spécification des contraintes imposées par la synthèse de l'unité de traitement. Cette modélisation est utilisée conjointement avec les contraintes d'intégration *IOCG* pour la synthèse des modules de stockage de données, des bus et des éléments d'interconnexion. L'unité de communication permettra à terme la synchronisation de l'unité de traitement et du système, la temporisation des données et la conversion entre le protocole basique de l'unité de traitement et celui standard ou non du système.

2.2.2. Modélisation des contraintes résultant de la synthèse de l'unité de traitement

La synthèse sous contraintes de l'*UT*, l'*UM* et l'*UC,* génère des descriptions de niveau RTL. Ainsi, l'unité de traitement est représentée par un modèle de type FSMD qui sera nommé dans le reste du document M_{UT}.

2.2.2.1 Machine d'état fini de type *FSMD*

Une *FSMD* est une machine d'états finis (FSM) étendue par un chemin de données représenté par un quintuplet *(S, INB, ONA, f, h)*

- S ensemble d'états
- I : ensemble d'entrées
- O : ensemble de sorties
- B α STAT
- A α ASG
- VAR un ensemble de variables de stockage du chemin de données
- ASG un ensemble assignation de stockage ASG = {X <= e | X γ VAR, e γ EXP}
- EXP un ensemble d'expressions avec EXP = {f(x, y, …) | x, y, z γ VAR }
- STAT un ensemble de variables de statut définies par des relations entre deux expressions de l'ensemble EXP : STAT = {Rel(a, b) |a, b γ EXP}
- f représente la fonction de transition décomposée en une fonction calculant l'état suivant f_c et une fonction calculant la valeur suivante des variables f_d définies comme suit :
  - $f_c : S \times I \times STAT \rightarrow S$
  - $f_v S \times I \times STAT \times EXP \rightarrow VAR$
- f représente la fonction de génération $g : S \times I \times STAT \times EXP \rightarrow O$

2.2.2.2 Génération du modèle IPERM *(IP Execution Requirements Model)*

La synthèse par partie n'étant pas supportée actuellement, la FSMD M_{UT} contient en plus du nœud v_0

- un unique point de synchronisation référence

ou

- un ensemble de points de synchronisation référence. Dans ce cas, les sous-graphes induits (ou chemin de données) par les sorties contraintes sont exécutés de façon totalement séquentielle et partagent les mêmes ressources matérielles de l'architecture du composant.

L'unité de mémorisation est modélisée par un ensemble de *FSMD* $M^{UM} = \{M_{UM1}, \ldots, M_{Umi}\}$ dans lequel M_{Umi} représente le $i^{ème}$ élément de stockage avec $1 \leq i \leq N_{IP}$ et N_{IP} représente le nombre de bus connectant l'unité mémoire et l'unité de traitement. *L'UT* et *l'UM* sont ainsi modélisées par un ensemble de *FSMD* communicantes (voir Figure 4-29).

La **première étape** permettant de générer le modèle *IPERM* fusionne l'ensemble des états du modèle M^{UM} avec ceux du modèle M_{UT} pour obtenir une unique machine d'état FSMD nommée M_{IP} (voir Figure 4-30). La **deuxième étape** fusionne les états séquentiels du M_{IP} n'ayant aucune dépendance avec une donnée d'entrée/sortie pour former un nouvel état nommé *super état*. Un super état représente un ensemble de calculs et d'accès mémoire réalisés entre deux transferts d'entrée/sortie (voir Figure 4-31).

Le modèle final *IPERM* est une *SFSMD (Super FSM with Datapath)* annotée contenant :

- un ensemble de super états produit par des étapes de fusion d'états de FSMD,
- des dates auxquelles les transferts entre l'*UT* et l'*UCOM* sont réalisés ou de façon équivalente un temps d'exécution associé à chaque super état.

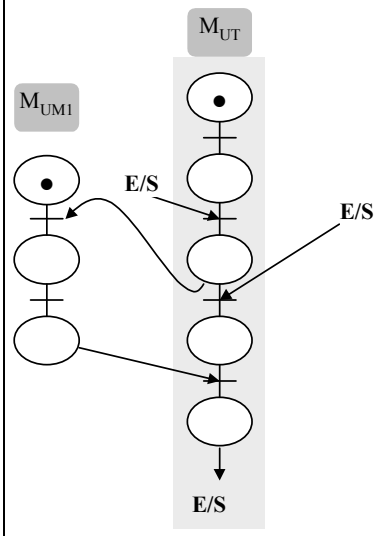


Figure 4-29 : Modèles *UT*, *UM*

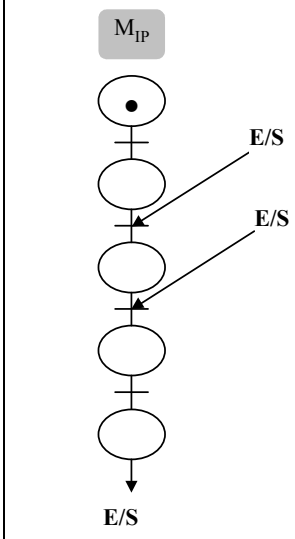


Figure 4-30 : Résultat de la première étape

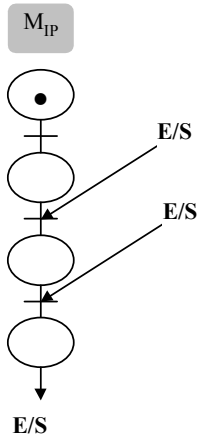


Figure 4-31 : Modèle IPERM

Une SFSMD est formellement définie comme une FSMD. La différence entre les deux modèles réside dans la sémantique associée à un état (ou super état) : une FSMD modélise le comportement au cycle près d'une architecture décrite au niveau RTL en associant à chaque état un temps d'exécution de un cycle exactement contrairement au modèle SFSMD dont le nombre de cycles séparant deux transitions successives est indéterminé. Le modèle IPERM que nous proposons, diffère d'une SFSMD classique dans le sens où le nombre de cycle entre deux transitions est connu et spécifié. Il se différencie de plus du modèle FSMD puisque le temps d'exécution d'un état peut être différent de un cycle d'horloge.

Ainsi, l'unité de traitement étant décrite au niveau RTL, les transferts d'E/S avec l'*UCOM* sont complètement spécifiés relativement aux points de synchronisation. Le temps d'exécution d'un super état est défini durant la construction par fusion des états des FSMD M_{UT} et M^{UM}. Ces informations temporelles sont combinées avec les dates minimums de transfert σ^{min} des entrées et les dates de transfert maximum σ^{max} des sorties. Cette combinaison permet donc la spécification des dates de production et d'acquisitions T_{UT} des données relativement à leur point de synchronisation et de référence. Les durées de vie des données dans l'unité de communication peuvent donc être calculées et utilisées afin de dimensionner les éléments mémorisants servant à la temporisation des données d'E/S dans l'*UCOM* (voir [BAG97]).

Première étape

1) ***Pour*** *tous les états de* M_{UT}
2) ***Pour*** *chaque dépendance de données de l'état courant avec un état de* M^{UM}
3) *Fusionner l'état du* M^{UM} *ayant une dépendance avec l'état courant dans un nouvel état du modèle* M^{IP}
4) ***Fin pour***
5) ***Fin pour***

Deuxième étape

1) ***Tant*** *qu'il existe des états dans MIP sans dépendances de données de type E/S*
2) ***Pour*** *tous les états de MIP*
3) ***Si*** *le prochain état n'a pas de dépendances de données de type E/S*
4) ***Alors*** *Le fusionner avec l'état courant*
5) ***Fin si***
6) ***Fin pour***
7) ***Fin tant que***

Figure 4-32 : Pseudo code de l'algorithme générant le modèle IPERM

Exemple

La Figure 4-33 décrit un SoC composé d'un composant virtuel dont l'unité de traitement a été synthétisée sous contraintes. L'IP est constitué des quatre unités fonctionnelles précedemment décrites et reçoit les données du système sur un unique bus par une séquence $S_{IP} = (X,Z,Y)$ i.e. $t_x < t_z < t_y$.

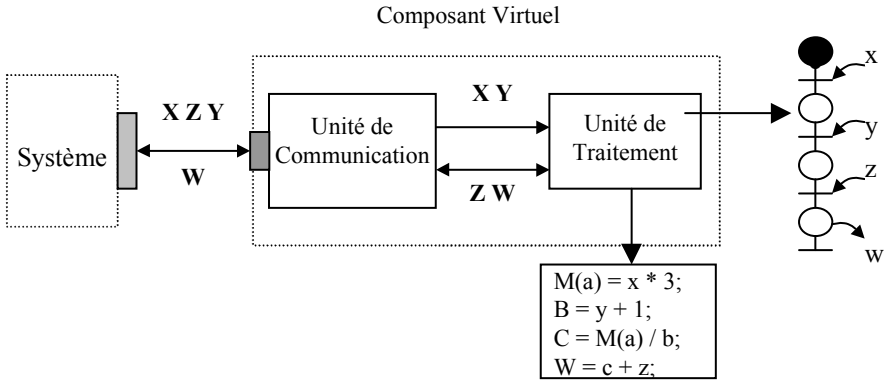


Figure 4-33 : Exemple

La Figure 4-34 décrit l'ensemble de FSMD communicantes du composant virtuel précédemment décrit. Le M^{um} est composé d'un unique banc mémoire. L'unité de traitement lit les entrées X et Y sur un bus interne n°1 et lit Z et écrit W sur le bus interne n°2. La FSMD modélisant l'unité de traitement contient 5 états. La Figure 4-35 et la Figure 4-36 illustrent respectivement les modèles générés par la première et la seconde étape de fusion. Le model IPERM M_{IP} est composé de trois super états. La Figure 4-37 décrit l'ensemble de contraintes utilisé pour la synthèse de l'unité de communication et modélisé par un graphe *IOCG*.

Dans cet exemple, les dates d'arrivée des entrées sont définies relativement au point de synchronisation référence *x* comme suit : $\sigma(x) = 0$, $\sigma(z) = 30$, $\sigma(y) = 60$, $\sigma(w) = 210$. La synthèse de l'unité de traitement spécifie les dates d'utilisation des entrées relativement à *x* : $T_{ut}(x) = 0$, $T_{ut}(y) = 60$, $T_{ut}(z) = 90$, $T_{ut}(w) = 150$.

Les durées de vie dans l'unité de communication sont donc :

$D(x) = [0, 30]$, $D(y) = [60, 90]$, $D(z) = [30, 120]$ et $D(w) = [180,210]$.

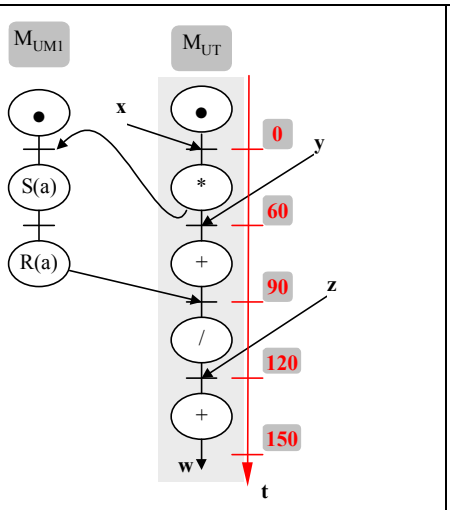


Figure 4-34: Modélisation *UT + UMEM*

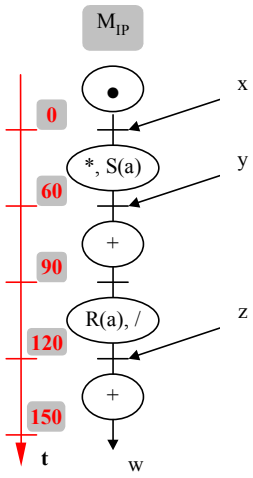


Figure 4-35: Résultat de la première étape

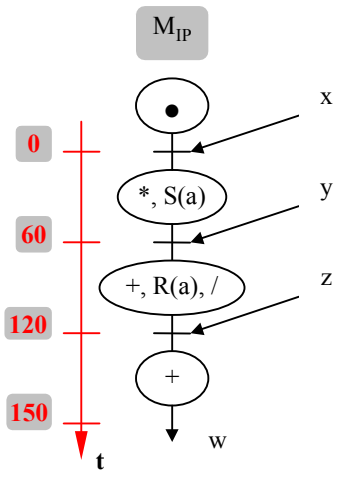


Figure 4-36: Résultats de la deuxième étape

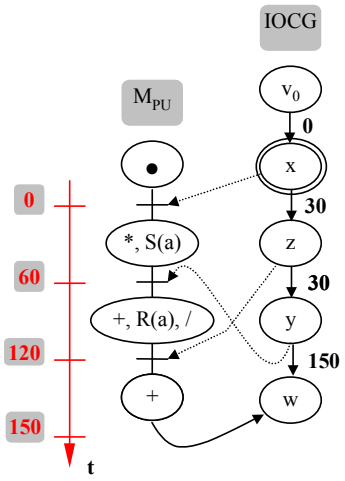


Figure 4-37: Contraintes de synthèse *UCOM*

De plus amples détails sur le calcul de la durée de vie des données, l'utilisation de structure FIFO, LIFO ou registres sont fournis dans le manuscrit de thèse d'Adel Baganne [BAG97].

2.2.2.3 Extensions possibles

Comme nous l'avons mentionné dans le chapitre 1, l'organisation VSIA a établi une liste des délivrables associés à chaque classe de composants virtuels. Ces délivrables comprennent notamment un ou plusieurs modèles simulables du composant et un modèle synthétisable. Les modèles simulables fournis avec un composant virtuel répondent à trois objectifs :

1. Analyse de performances du composant (IP DELAY Model)

2. Vérifier après achat que le modèle synthétisable est correct (IPERM, RTL)

3. Accélérer la simulation fonctionnelle et temporelle du système en cours de conception en y insérant un modèle comportemental exact au cycle près du composant (IPERM, SFSMD)

4. Source RTL synthétisable

Nous proposons ici trois extensions possibles pour la génération de modèles de l'IP (voir Figure 4-28).

A. Génération d'un modèle de performances (IP Delay Model)

Comme nous l'avons vu dans le chapitre 1, l'utilisation d'un modèle décrivant un IP à haut niveau d'abstraction permet au concepteur système de simuler et d'évaluer le comportement du composant durant les phases d'analyse de performances avec des outils tel que VCC. Dans ce cas, la description fonctionnelle de niveau système de l'IP est accompagnée d'un modèle de performances qui décrit ses caractéristiques temporelles. Ceci permet de vérifier les problèmes de synchronisation et de raffiner la communication. Nous proposons de générer un *IP Delay Model* à partir du IOCG et du modèle IPERM décrivant le comportement aux E/S et le

traitement. Dans l'environnement VCC ce modèle peut être décrit à l'aide du langage *"Delay Script Language"* dérivé du C (voir Figure 4-38). Ce modèle permettra à terme l'évaluation d'un composant virtuel algorithmique.

```
...
delay_model () {
input (x); input(y);input(z);              /*Lecture des entrées */
run();                                                  /*Calcul */
delay('180e-9');       /*Attente avant la production du résultat */
output(w);                                 /*Ecriture des sorties */
...}
```

Figure 4-38 : Exemple de IP Delay Model décrit en DSL

B. Génération d'un modèle de l'IP pour la synthèse

La modélisation IPERM *(SFSMD)* peut être utilisée pour représenter le comportement au cycle près E/S du composant virtuel. En effet, en intégrant les aspects liés aux transferts des E/S, nous pourrions générer un modèle de type machine d'état comportementale utilisé dans les outils tel que BC ou Monet (voir la thèse de G. Savaton [SAV02]). L'acquisition des points de synchronisation sera spécifiée à l'aide de fonctions utilisant une structure de contrôle de type *While* et d'instructions de synchronisation *wait.* La description étant de niveau comportemental, ce nouveau modèle permettra à terme une simulation plus rapide de l'algorithme que le modèle RTL. Il pourra de plus être couplé à un modèle de performance brièvement décrit dans la section suivante.

C. Génération d'un modèle de l'*UT* pour la simulation

Le modèle IPERM peut aussi être utilisé pour la génération d'un modèle comportemental simulable de l'unité de traitement au niveau RTL. Une thèse est actuellement en cours sur les modèles permettant la cosimulation multi-niveau de système utilisant des composants virtuels de niveau RTL [GAI03].

3. L'environnement logiciel *GAUT*

Bien que le critère de partitionnement ait été présenté, nous n'avons pas intégré la synthèse par parties d'un algorithme dans GAUT. Ce problème sera traité dans une thèse dédiée à la synthèse par partie et à la sérialisation de graphe contenant plusieurs points de synchronisation référence. Ainsi, les travaux réalisés sur l'outil concernent uniquement les graphes contenant un point de synchronisation après les phases de transformation et de suppression de point de synchronisation.

3.1. Synthèse de l'unité de traitement

Comme nous l'avons mentionné dans le chapitre 2, l'architecture ciblée par le flot de synthèse de *GAUT* est composée de trois unités fonctionnelles : l'unité mémoire, l'unité de traitement et l'unité de communication. Dans cette première partie, nous détaillons le flot de synthèse, sous contrainte de cadence d'itération, de l'unité de traitement d'un algorithme.

Dans l'environnement *GAUT*, cinq grandeurs de temps sont utilisées pour effectuer la synthèse de l'unité de traitement :

- *Tfonc* : le temps de fonctionnement des opérateurs défini en bibliothèque
- *Phase_min* : le temps de cycle de l'horloge
- $Tcycle = \mathrm{Max}\{Tfonc\}_{opérateurs\ sélectionnés}$: le plus grand temps de traversée d'un opérateur
- *Tcadence* : le temps de cadence pour l'exécution de l'application
- *Tlatence* : le temps d'exécution de l'algorithme (chemin critique) qui, s'il est supérieur à *Tcadence*, impose la mise en œuvre d'une architecture de traitement pipeline

3.1.1. Sélection / Allocation

La synthèse de l'unité de traitement débute par la sélection des opérateurs. Cette phase permet d'associer des délais *Tfonc* aux opérations qui ont été, au préalable, ajustés pour être des entiers multiples de la fréquence d'horloge *phase_min*. La mobilité des opérations est ensuite calculée en utilisant les dates au plus tôt *(ASAP)* et au plus tard *(ALAP)* (voir [GAJ91]) de chaque nœud du graphe. Pour cela, les données d'entrées ont une date au plus tôt fixée à t=0, et les sorties une date au plus tard fixée à *Tlatence*. L'allocation dans *GAUT* consiste à allouer des opérateurs sur chaque tranche de pipeline : le nombre d'opérations de chaque type est donc comptabilisé par tranche. Afin de connaître la quantité d'opérateurs à mettre en œuvre pour satisfaire le parallélisme moyen de l'application, le nombre moyen d'opérateurs par tranche est calculé comme suit :

$$\mathrm{Nb\_opr}(type, \mathrm{Tranche}) = \frac{\mathrm{Nb\_ops}(type, \mathrm{Tranche}) * \mathrm{tps\_fonc}(type)}{\mathrm{Tcadence}}$$

Si cette allocation est respectée durant l'ordonnancement alors la meilleure solution en terme de surface est obtenue. Cependant si le nombre d'opérateurs alloués est insuffisant, on observe une élongation du graphe qui entraîne l'ajout de tranche de pipeline et la création de nouvelles ressources. La complexité de la phase d'allocation est en *O(n)*, où n représente le nombre de nœuds du graphe. Le nombre de tranches *Tranche* de pipeline est défini par :

$$\mathrm{Tranche} = \mathrm{Tlatence} / \mathrm{Tcadence}$$

où *Tlatence* représente la date maximum au plus tôt (ASAP) des nœuds opérations.

3.1.2. Ordonnancement / Assignation

Afin de réduire au maximum la complexité de l'ordonnancement, l'algorithme est de type glouton : une décision n'est prise qu'en fonction des événements passés et ne peut être remise en cause par la suite. Un des objectifs de l'ordonnancement implanté dans *GAUT* est de réguler le parallélisme du *GFS* au parallélisme moyen tout en limitant l'élongation du graphe. En effet plus l'allongement est important plus le nombre de tranches et donc le nombre d'opérateurs est grand. L'algorithme utilisé, de type *list scheduling* (voir [GAJ91]), est basé sur la gestion des priorités des opérations à exécuter et permet de maîtriser l'élongation du *GFS*. Un autre objectif de l'ordonnancement est d'optimiser l'utilisation des opérateurs. Pour cela, une priorité d'utilisation est associée aux opérateurs qui sont ainsi réutilisés dès leur libération. La progression du temps se fait donc par événement et non de manière discrète.

Nous allons maintenant décrire les traitements réalisés par la phase d'ordonnancement (voir Figure 4-39). Une première étape initialise des structures parmi lesquelles se trouvent deux ensembles de files dont les fonctions sont primordiales.

Le premier ensemble permet la gestion des opérations et de leurs différents états : *"exécutables"* et *"en cours"* d'exécution. Une file est attribuée à chaque état et pour chaque type de fonction (addition, soustraction, multiplication...). Lorsque la phase d'ordonnancement débute, l'ensemble des opérations exécutables est initialisé avec l'ensemble des opérations dont la date au plus tôt est égale à 0. Lorsqu'une opération de fonction *f* devient exécutable, c'est à dire lorsque l'ensemble de ces prédécesseurs, de type opération, a été réalisé, elle est enfilée dans la liste des opérations de type *f*, de l'ensemble des opérations *exécutables*. Au moment où cette opération est assignée à un opérateur, elle est défilée de l'ensemble des opérations *exécutables*, et enfilée dans l'ensemble des opérations de type *f en cours d'exécution*.

Le deuxième ensemble permet la gestion des opérateurs et de leurs différents états: *"sommeil", "en attente", "disponible"* et les sous états "*libre"* et *"occupé"* de l'état *disponible.* Un opérateur *disponible* est *libre* à un instant *t* s'il peut exécuter une opération à cet instant. Il sera *occupé* pendant toute la durée de l'exécution. Les opérateurs en *sommeil* sont des opérateurs qui ont été crées par l'allocation mais qui n'ont encore jamais été utilisés. Les opérateurs qui ont déjà été utilisés (*libres*) sont prioritaires par rapport aux autres opérateurs (en *sommeil*). Les opérateurs sont crées dans l'état *sommeil* à chaque nouvelle tranche d'après le taux de parallélisme calculé lors de l'allocation : l'ensemble des opérateurs *disponibles* est donc vide. Lorsqu'un opérateur est utilisé pour la première fois il passe de l'état *sommeil* à l'état *disponible et occupé*. Lorsqu'un opérateur est *"usé"*, c'est à dire lorsque son temps d'utilisation total devient supérieur au temps de cadence, il est retiré de l'ensemble des opérateurs *disponibles.* Lorsqu'il n'existe plus d'opérateurs *disponibles,* on va chercher l'opérateur souhaité dans l'ensemble des opérateurs en *sommeil*. Les opérateurs *disponibles* sont triés par état et par date de première utilisation, donc par priorité décroissante. Un opérateur est *en attente* à l'instant *t* s'il ne peut terminer l'opération avant le prochain *Tcycle.* Le *Tcycle* est utilisé comme point de (re)synchronisation des opérations dans l'unité de traitement. Ceci limite le nombre de combinaisons d'opérations dans un *Tcycle* et aboutit à la réduction du nombre d'états de la FSM.

Les quatre étapes suivantes réalisent le cœur d'ordonnancement et d'assignation de *GAUT*.

1. **Sélection d'une opération.** Les opérations exécutables sont triées suivant leur marge croissante. La marge d'une opération correspond au temps qu'il reste entre l'instant courant et sa date au plut tard :

$$\mathrm{Marge}(s) = D_{Tard}(s) - t$$

Ainsi, plus la marge est grande moins l'opération est prioritaire.

2. **Sélection d'un opérateur.** On tente d'assigner chaque opération exécutable à un opérateur suivant la priorité des opérateurs.

   - Opérateur déjà mis en œuvre

   - Opérateur minimisant le nombre de liaisons avec les opérateurs réalisant les opérations précédant l'opération à allouer. Cette gestion des opérateurs permet une optimisation des liaisons inter cellules et réduit donc la complexité des interconnexions.

Lorsqu'une opération n'a pu être assignée, elle est conservée dans l'ensemble des opérations exécutables. Elle sera considérée lors de la prochaine phase d'ordonnancement et ce pour la date définie par la fonction de progression du temps. Une opération à marge négative est dite *immédiate* et est assignée à un opérateur en *sommeil* s'il n'existe plus d'opérateur *libre*.

3. **Progression du temps.** La progression du temps est particulière. Dans un premier temps on scrute l'ensemble des opérations en cours d'exécution pour sélectionner le plus petit temps *t1* des dates de relâchement. Cette date de relâchement est égale à la date de fin pour les opérations assignées à des opérateurs non pipeline. Par contre, pour une opération assignée à un opérateur pipeline, la date de relâchement est différente de la date de fin, et correspond à l'instant *t2* où (bien que l'opération ne soit par terminée) le premier étage de pipeline de l'opérateur doit être relâché. Le nouveau temps t est alors égal à $t = \min(t1, t2)$. Lorsque le temps courant *t* dépasse l'instant de la tranche de pipeline suivante *(Tranche*Tcadence)*, on crée alors les opérateurs prévus par l'allocation pour cette tranche. Le temps courant est égal au *Tcycle* suivant lorsqu'il ne reste que des opérations exécutables sur des opérateurs en *attente*.

4. **Libération des opérateurs.** Après avoir fait progresser le temps, on recherche parmi les opérations en cours celles qui se terminent ou relâchent un opérateur pipeline. Dans ce cas, les opérateurs *occupés* passent dans un l'état *disponible.* Lorsqu'une opération *s* est réellement terminée, c'est à dire lorsque sa date de fin est égale au temps courant, on cherche les nouvelles opérations exécutables engendrées par la terminaison de *s*. Pour cela, on recherche parmi tous les successeurs de *s*, les opérations pour lesquelles l'ensemble des prédécesseurs a été réalisé. Les opérateurs *usés* sont retirés de l'ensemble des opérateurs *disponibles*.

La Figure 4-39 décrit l'algorithme d'une tentative d'ordonnancement. Il est important de noter qu'avant de commencer la phase d'assignation, il existe une phase de création éventuelle d'opérateurs n'ayant pas été prévus par l'allocation. En effet, certaines opérations ne peuvent être allouées dans la tranche de pipeline où elles étaient initialement prévues. On observe dans ce cas une élongation du graphe qui nécessite la création d'opérateurs supplémentaires pour traiter ce *parallélisme dur*.

```
Sélection et Allocation;
Initialisation de l'ordonnancement;
Tant qu'il reste des opérations à exécuter faire
    Création éventuelle d'opérateur pour traiter le parallélisme dur;
    Chercher à assigner toutes les opérations exécutables;
    Progression du temps;
    Rechercher les opérations en cours d'exécution terminant ou relâchant leurs
    opérateurs;
    Libérer les opérateurs assignés aux opérations se terminant;
    Déterminer les nouvelles opérations exécutables;
    Si nouvelle tranche
       Créer les opérateurs prévus par l'allocation;
    Détecter les opérations en attente;
    Supprimer les opérateurs usés;
Fin tant que;
```

Figure 4-39 : Algorithme d'une phase d'ordonnancement

3.1.3. Traitement de la mémorisation

Les fonctions d'ordonnancement et d'assignation tentent de produire la meilleure solution en terme de surface en respectant les contraintes de performances spécifiées par l'utilisateur. Le critère de surface peut prendre en compte les registres, mais l'algorithme utilisé considère en priorité le coût global des opérateurs. La prise en compte des registres se fait localement au cours du choix des opérateurs *disponibles* durant la phase d'assignation. La phase d'optimisation des registres est donc réalisée a posteriori de l'ordonnancement. Elle réduit le nombre de registres utilisés en ajoutant des composants d'interconnexion (multipléxeurs, démultiplexeurs, modules trois états). Différents algorithmes peuvent être utilisés dans l'environnement *GAUT* pour optimiser le nombre de registres de l'unité de traitement : LeftEdge, Branch&Bound ou Branch&Bound avec des heuristiques [MAR92].

Ainsi, la mémorisation des données est réalisée soit par des registres dans l'unité de traitement soit par des mémoires dans l'unité de mémorisation [CHI97A], [CHI97B], [SEN97B]. Le critère de sélection de l'emplacement des variables se base sur une notion de localité définie comme suit :

Une variable est *locale* si sa durée de vie dans l'unité de traitement dépasse un certain *seuil*, sinon elle est *globale*.

La valeur du *seuil* correspond à la somme du temps nécessaire pour réaliser l'écriture puis la lecture d'une donnée en mémoire, et du temps de fonctionnement de l'opérateur du plus lent. Il y a *rebalancement* lorsque qu'une variable *locale* devient *globale*.

Une thèse est actuellement en cours au laboratoire LESTER et permettra à terme la prise en compte des contraintes sur le nombre de bancs mémoires disponibles pour réaliser l'ordonnancement [COR03A] [COR03B].

3.1.4. Contraintes liées aux techniques de synthèse de l'unité de traitement

Les principales caractéristiques de la synthèse de l'unité de traitement sont

- **L'allocation du nombre d'opérateurs :** la technique d'allocation, basée sur le nombre moyen d'opérateurs est adaptée aux synthèses contraintes par les ressources matérielles.

- **Le fonctionnement pipeline :** la synthèse de l'UT peut imposer un mode d'exécution pipeline. L'allocation en moyenne du nombre d'opérateurs à mettre en œuvre, l'utilisation en priorité des opérateurs déjà utilisés ainsi que le retardement de certaines opérations, lié au *Tcycle,* font que le nombre de tranches de pipeline n'est connu qu'après l'ordonnancement.

- **La progression du temps :** La progression du temps se fait uniquement sur les événements de relâchements des opérateurs et en fonctions des *Tcycles*. Les données lues sur les ports d'entrées sont supposées être présentes aux dates assignées par l'ordonnancement. Ainsi, les contraintes temporelles d'entrée/sortie ne sont pas sont pas prises en compte lors de la synthèse.

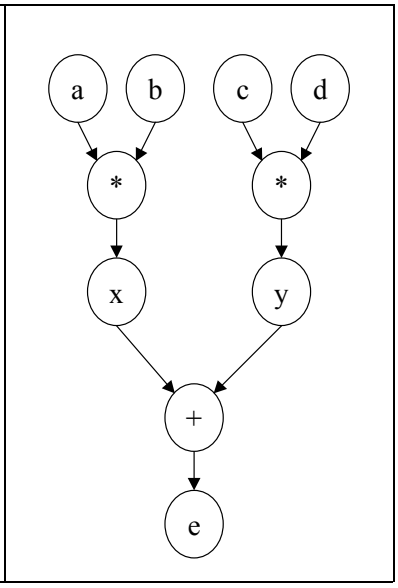
Exemple de synthèse

Dans ce paragraphe, nous illustrons les conséquences liées aux techniques de synthèse utilisées dans la version non modifiée de *GAUT* par deux exemples pédagogiques.

Le premier dont l'algorithme est illustré dans la Figure 4-40 (a) met en avant les conséquences de l'utilisation d'un module d'allocation en moyenne sur l'ordonnancement. La Figure 4-40 (b) décrit la représentation interne de l'algorithme, obtenue après compilation.

```
Entity exemple1 is
        port (a, b, c, d :in integer; e :out integer);
end exemple1;

architecture exemple_arch of exemple1 is
        constant Tcadence : time := 120 ns;
begin
        process
        Variable x, y : integer;
        Begin
                x := a*b;
                y := c*d;
                e <= x + y;
                wait for Tcadence;
        end process;
end exemple_arch;
```



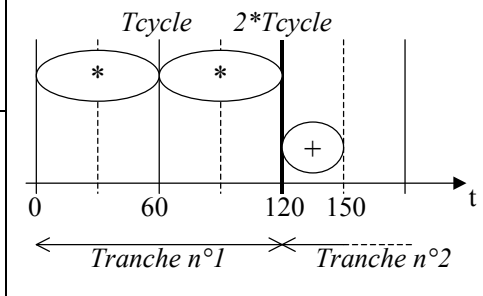
(a) **(b)**

Figure 4-40 : (a) Description VHDL de l'exemple (b) Représentation interne de l'exemple

La description initiale est accompagnée d'une contrainte de cadence *Tcadence* de 120 ns (voir Figure 4-41 (a)). La période d'horloge *phase_min* du composant est de 30ns et les temps de traversée *Tfonc* des opérateurs de multiplication et d'addition sélectionnés sont respectivement de 60 ns et 30ns. Le chemin critique *Tlatence* du circuit est de 90 ns et l'architecture contient a priori une unique tranche de pipeline. Le temps de cycle *Tcycle* est de 60 ns. Le module d'allocation indique qu'un multiplieur et un additionneur sont requis afin de satisfaire le parallélisme moyen de l'application. Les dépendances de données entre les opérations nécessitent la complétion des deux opérations de multiplication afin de pouvoir exécuter l'opération d'addition. Les dépendances de données couplées au nombre et au type d'opérateurs alloués résultent en une architecture contenant après synthèse deux tranches de pipeline (voir Figure 4-41 (b)). Les données *a* et *b* doivent être disponibles dans l'unité de communication à t = 0, et les données c et d à t = 60ns. Le résultat est fourni toutes les 120 ns.

Tfonc(*) = 60ns, Tfonc(+) = 30ns
Phase_min = 30ns
Tlatence = 90ns
Tcadence = 120ns

Nb_opr(*) = (2*60)/120 = 1
Nb_opr(+) = (1*30)/120 = 1
Avant synthèse
  Tranche = 1
Après synthèse
  Tranche = 2



(a) **(b)**

Figure 4-41 : (a) contraintes et résultat de synthèse (b) résultat de l'ordonnancement

Ainsi, le mode d'allocation choisi peut faire aboutir à des architectures ne respectant pas les contraintes imposées sur les données d'entrée/sortie même si ces dernières sont réalisables en fonction des dépendances de données de l'algorithme et des contraintes technologiques. En effet, si la date imposée pour la production de la sortie e avait été de 90ns (temps d'exécution du chemin critique) alors la technique d'allocation aurait aboutit à la violation de la contrainte.

Le deuxième exemple, dont la Figure 4-42 (a) décrit la représentation interne après compilation, est utilisé afin de mettre en évidence les conséquences de l'utilisation du *Tcycle* sur l'ordonnancement des opérations. La cadence spécifiée correspond à la latence du chemin critique égale à 150ns. Les contraintes de synthèse ainsi que le nombre et le type d'opérateurs alloués ont des valeurs identiques à celles du premier exemple. La Figure 4-42 (b) montre l'état d'attente qui est inséré entre la date de fin de l'opération d'addition et le début de l'exécution de la deuxième multiplication. **Cette gestion du temps aboutit à une architecture pipeline qui pourrait violer une contrainte fixée sur la date de production de la sortie *e* même si celle ci est bien posée au regard du chemin critique de l'application et des contraintes sur les données d'entrée.** En d'autres termes, une contrainte de production du résultat en 150ns, en supposant les dates d'arrivées des entrées à 0ns est impossible car la latence de l'architecture synthétisée est de 180ns.

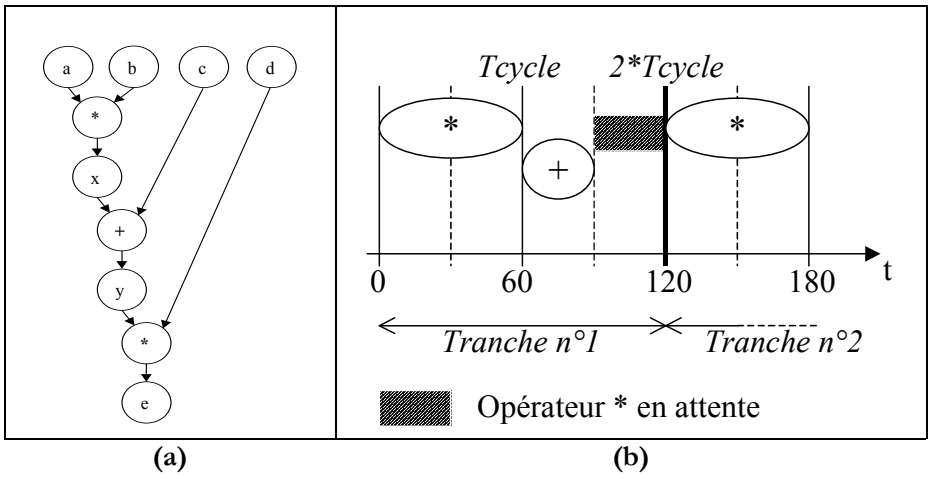


Figure 4-42 : (a) Représentation interne de l'exemple 2 (b) résultat d'ordonnancement

Ainsi, la prise en compte des contraintes temporelles sur les E/S nécessite un ensemble **d'adaptations du cœur de synthèse de l'unité de traitement**. Les modifications concernent la prise en compte des dates d'E/S, le mode d'allocation en moyenne et la technique de progression du temps.

3.2. Synthèse sous contraintes

Dans cette section nous allons décrire comment la gestion des contraintes d'entrée/sortie a été intégrée dans *GAUT*. Nous abordons dans un premier temps les orientations permettant d'adapter le module d'allocation en moyenne pour réaliser une synthèse sous contraintes temporelles. Nous présentons ensuite les modifications qui ont été apportées au cœur d'ordonnancement et qui concernent la gestion du temps et les phases d'initialisation.

3.2.1. Calcul des mobilités

Le calcul de mobilité est réalisé classiquement dans *GAUT* par les fonctions ASAP (voir Figure 4-43) et ALAP (voir Figure 4-44). Pour cela, l'outil suppose que les données lues sur les ports d'entrées sont toujours présentes. Cette hypothèse n'est plus valide lors de la prise en compte des contraintes d'entrée/sortie.

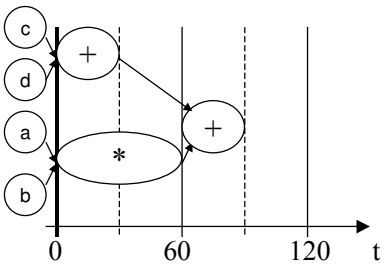


Figure 4-43 : Ordonnancement ASAP

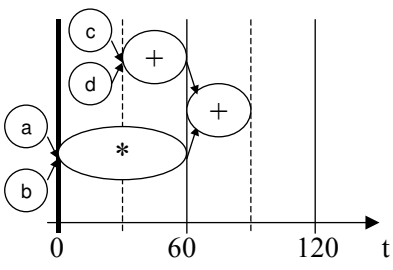


Figure 4-44 : Ordonnancement ALAP

Comme nous l'avons mentionné dans la section précedente, le temps d'accès en lecture et en écriture à l'unité de communication doit être pris en compte lors de la synthèse de l'unité de traitement. Ainsi (voir chapitre 3), un point de synchronisation v_i n'est accessible dans l'unité de traitement qu'à une date $ASAP\_UT(v_i) = ALAP\_UCOM(v_i) + \Delta_{LEC}$, un point de référence v_i n'est accessible dans l'unité de traitement qu'à une date $ASAP\_UT(v_i) = ASAP\_UCOM(v_i) + \Delta_{LEC}$ et une sortie v_j doit être produite par l'unité de traitement avant la date $ALAP\_UT(v_j) = ASAP\_UCOM(v_j) - \Delta_{ECR}$.

La fonction originale ASAP, qui ignore les données d'entrée, a été modifiée pour utiliser les dates ASAP des entrées dans le calcul des dates ASAP des nœuds opération et des nœuds sortie (voir Figure 4-45). La fonction ALAP, utilisant uniquement le chemin critique, a quant à elle été modifiée pour intégrer les dates ALAP des sorties dans le calcul des dates ALAP des nœuds opération et entrée(voir Figure 4-46).

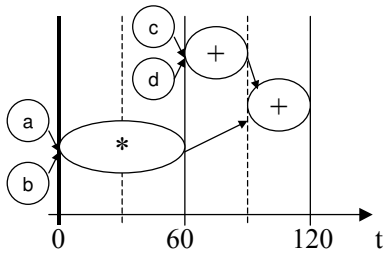


Figure 4-45 : Ordonnancement ASAP modifié

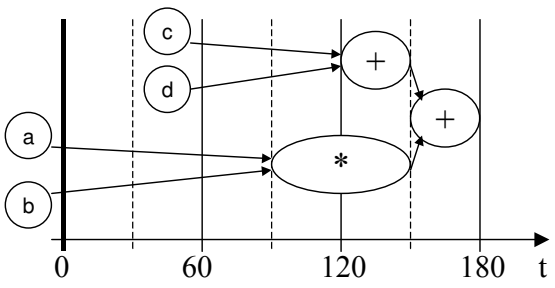


Figure 4-46 : Ordonnancement ALAP modifié

3.2.2. Allocation

Comme nous l'avons précédemment mentionné, le module originel d'allocation de *GAUT* fournit le nombre moyen d'opérateurs garantissant, quand cela est possible, une synthèse optimale en terme de surface. Nous avons apporté des modifications à ce mode d'allocation qui, couplé à la technique d'ordonnancement de type liste, n'est plus adapté à la synthèse de circuits sous contraintes temporelles. **Il est important de noter que la solution proposée a permis, dans un premier temps, de prendre en compte les contraintes dans l'outils *GAUT* afin de valider l'approche de conception. Toutefois, une approche par estimation du nombre d'opérateurs à allouer avant synthèse peut aussi être envisagée pour résoudre le problème de l'allocation avant celui de l'ordonnancement ([DIG96], [SEN96A]).**

Ainsi, nous avons ajouté un module d'allocation dynamique dans le cœur d'ordonnancement. Ce module crée un opérateur lorsqu'une opération est immédiate (sa marge est égale à 0) et qu'il n'existe plus d'opérateur *disponible* ou en *sommeil*. Bien qu'elle soit fonctionnellement correcte, cette solution fournit dans certains cas des résultats sous optimaux. En effet, afin d'utiliser au maximum le nombre moyen d'opérateurs alloués, l'ordonnancement retarde les opérations lorsqu'il n'y a plus d'opérateurs disponibles ce qui a pour effet de regrouper les opérations urgentes. La solution apportée par l'allocation dynamique aboutit dans ce cas à des pics de création d'opérateurs qui sont bien souvent sous utilisés. Afin d'éviter ce problème, nous utilisons un algorithme qui effectue une recherche dichotomique entre le nombre d'opérateurs de type *f* alloués en moyenne *x(f)* et le nombre d'opérateurs résultant de l'ordonnancement effectué avec une allocation dynamique *y(f)*.

Comme l'indique l'algorithme de la Figure 4-47, lorsque la synthèse sous contrainte ne peut être réalisée avec un nombre moyen d'opérateurs, nous réalisons un ordonnancement avec allocation dynamique puis réduisons, quand cela est possible, le nombre d'opérateurs sélectionnés par une recherche dichotomique.

```
Si l'ordonnancement sous contraintes de temps avec une allocation en moyenne est possible alors
        STOPPER car solution optimale;
Sinon
        Ordonnancer sous contrainte de temps avec allocation dynamique;

// Alloc_moy = nombre d'opérateur résultant de l'allocation en moyenne
// Alloc_dyn = nombre d'opérateur résultant de l'ordonnancement utilisant l'allocation dynamique
        Allocation_Ordonnancement_Dichotomique(Alloc_moy, Alloc_dyn, premier_f);
```

Figure 4-47 : Algorithme Allocation / Ordonnancement modifié

La Figure 4-48 présente l'algorithme récursif de la recherche dichotomique. L'algorithme réalise un ordonnancement pour l'ensemble des solutions d'allocation et ce de façon récursive. Il retourne une fois le parcours terminé la meilleure solution relative au coût de la synthèse. Le critère de coût utilisé peut être la surface des opérateurs, la consommation ou le nombre d'opérateurs maximum.

```
Allocation_Ordonnancement_Dichotomique (X_appel, Y_appel, f)

X(f) = X_appel(f); // nombre minimum d'opérateur réalisant la fonction f
Y(f) = Y_appel(f); // nombre maximum d'opérateur réalisant la fonction f

Tant que (X(f) est différent de Y(f)) //deux valeurs identiques indiquent la fin de la dichotomie
        Y'(f) = PartieEntInf [y(f) + x(f)/2] // Dichotomie
        Si f n'est pas la dernière fonction
                Si(Allocation_Ordonnancement_Dichotomique (X,Y', f_suivant) est valide)
                        Y = Y' // on diminue le nombre maximum d'opérateur de type f
                Sinon
                        X = Y' // on augmente le nombre minimum d'opérateurs de type f
        Sinon
                Si Ordonnancement(Y') est valide
                        Si le coût de cet ordonnancement est inférieur au coût précédent
                        Mise à jour du coût
                        Indiquer la validité de la synthèse
```

Figure 4-48 : Algorithme récursif de recherche de la meilleure solution

Complexité

Remarque : l'hypothèse (nécessaire pour le calcul de complexité) supposant un type d'opérateur différent pour chaque opération est envisageable dans les algorithmes de faible complexité mais reste peu probable pour les algorithmes complexes. Ainsi, la relation existant entre le nombre total de nœuds du graphe *n* et le nombre total de type de fonction *T* implique que le produit maximum ne peut dépasser *n/T*.

La complexité de l'algorithme d'un ordonnancement de type liste est en *O(n)* où n représente le nombre d'opérations du graphe. La recherche est réalisée pour chaque type *f* d'opérations et si

les opérations du graphe ont toutes un type d'opérateur différent, alors sa complexité est égale au nombre total de combinaisons parmi tous les types :

$$\text{Nb\_total\_combi\_possible} = \prod_{f=1}^{T} (y(f) - x(f) + 1) \leq \left(\frac{n}{T}\right)^{T} \leq n^{T}$$

En utilisant une recherche de type dichotomique nous réduisons la complexité et obtenons

$$\prod_{f=1}^{T} [y(f) - x(f)] \leq \left[\log(\frac{n}{T})\right]^{T}$$

La complexité totale de l'algorithme d'ordonnancement couplé à l'allocation dichotomique est en $O(n[log(n/T)]^T)$.

Le Tableau 4-2 compare, à titre d'exemple, les résultats de synthèse que nous obtenons sur un filtre elliptique du cinquième ordre avec ceux fournis par les algorithmes d'ordonnancements *Force Directed Scheduling [PAU89], SALSA [NES96] et ASAP[PAU89]*. Les additionneurs et les multiplieurs ont une latence respective de un et deux cycles. L'ordonnancement est réalisé sans chaînage d'opération et les opérateurs ne sont pas pipelines.

Algorithme	Latence	Multiplieur	Additionneur
FDS $O(n^2)$	17	3	3
	18	2	3
	21	1	2
SALSA *O(n) +recuit simulé*	17	3	3
	18	2	3
	21	1	2
ASAP *O(n)*	17	4	4
	18		
	21		
Dichotomie $O(n[log(n/T)]^T)$.	17	3	3
	18	2	3
	21	1	2

Tableau 4-2 : Filtre elliptique: Différentes approches pour l'allocation sous contraintes

3.2.3. Ordonnancement

Plusieurs modifications ont été nécessaires pour rendre compatible l'algorithme d'ordonnancement par liste avec la prise en compte des contraintes temporelles. Comme nous l'avons vu *GAUT* dans sa version originale suppose que les données lues sur les ports d'entrées sont toujours présentes : une opération de lecture sur un port est interprétée par l'outil comme une opération déjà ordonnancée. Ainsi, la progression du temps n'est réalisée qu'en fonction des événements de sortie des opérateurs. Cette gestion du temps n'est plus valide lorsque des dates d'arrivées sont associées aux entrées : la progression du temps doit être réalisée en fonction des

événements de sortie des opérateurs mais aussi des événements sur les ports d'entrées. Les modifications apportées concernent

1. Recherche de la première date (temps initial) à laquelle l'ordonnancement peut commencer à exécuter les opérations du *GFS* qui ne dépendent que des nœuds entrée.

2. Rechercher des événements sur les ports d'entrée pour détecter l'arrivées de données durant l'exécution d'une opération de latence supérieure à un temps de cycle.

3. Gestion des opérations qui dépendent d'une entrée et du résultat d'un calcul.

4. Modification de la gestion du temps : la notion de *Tcycle* a été rendue optionnelle afin d'éviter la violation de certaines contraintes temporelles sur les sorties (voir section 3.1.4).

La Figure 4-49 décrit l'algorithme de la fonction modifiée de l'ordonnancement (version initiale Figure 4-39).

```
Initialisation de l'ordonnancement;
①Recherche du temps initial;
Tant qu'il reste des opérations à exécuter faire
    Création éventuelle d'opérateur pour traiter le parallélisme dur;
    Chercher à allouer toutes les opérations exécutables;
    Progression du temps;
        ②Recherche d'événements sur les Entrées;
    Rechercher les opérations en cours d'exécution terminant ou relâchant leurs opérateurs;
    Libérer les opérateurs assignés aux opérations se terminant;
    Déterminer les nouvelles opérations exécutables;
        ③Mise à jour des opérations exécutables en fonction des événements d'entrées
    Si nouvelle tranche
       Créer les opérateurs prévus par l'allocation;
    ④/* Détecter les opérations en attente; */
    Supprimer les opérateurs usés;
Fin tant que;
```

Figure 4-49 : Algorithme d'une phase d'ordonnancement sous contraintes

3.2.4. Machine d'état

La génération du contrôleur pilotant l'unité de traitement est une FSM linéaire. Cette dernière contient pour chaque point de synchronisation, un état de lecture dont la transition est validée par l'acquisition de la donnée dans l'unité de communication. Ce protocole simple permet une synchronisation de l'unité de traitement et de l'unité de communication, et ce, indépendamment du protocole de synchronisation utilisé entre le système et l'unité de communication. Dans le cas où les données seraient stockées dans des FIFO c'est l'état de cet élément qui permet la génération du signal de synchronisation. Lors de l'utilisation d'un protocole de type poignée de

main c'est l'acquittement qui permet la génération du signal de synchronisation. De plus amples détails sont fournis dans le rapport technique [BOM03].

4. Bilan et Conclusions

Dans une première partie de ce chapitre, nous avons montré que le nombre de partitions pouvait être réduit en supprimant certains points d'E/S. Dans ce contexte, nous avons défini un critère de *redondance* servant à la suppression des nœuds de synchronisation référence et présenté l'ensemble de transformations formelles, correspondantes, opérées sur le graphe de contraintes globales. Nous avons illustré par l'exemple l'insuffisance de ce critère pour réaliser, dans certains cas, une architecture contenant une unique partition pilotée par une FSM linéaire. En effet, certaines spécifications des échanges entre l'IP et le système requièrent en plus de la suppression de nœuds, une *sérialisation* des transferts des données dans le *GCG*. Une étude menée sur l'ordonnancement et la sérialisation des opérations d'E/S du *GCG*, a mis en évidence la nécessité d'un choix d'implémentation : synthèse d'une partition unique versus synthèse par parties. En effet, ces transformations qui minimisent le nombre de partitions et autorisent la génération d'un contrôleur linéaire, réduisent en contrepartie la mobilité des opérations. Inversement, la synthèse par parties peut, elle aussi, générer une architecture dont le coût est supérieur à la synthèse d'une seule partition. *La définition d'une méthode sélectionnant l'implémentation d'une l'architecture mono ou multi partitions sera réalisée dans de futurs travaux de recherche.* Finalement, nous avons proposé une modélisation des contraintes générées par la synthèse de l'unité de traitement, qui couplée avec la modélisation des contraintes d'intégration, permettra à terme la synthèse de l'unité de communication.

Dans la deuxième partie de ce chapitre nous avons présenté l'outil de synthèse architecturale *GAUT*. Dans sa version, originale, cet outil permet une synthèse automatique des unités de traitement, de mémorisation et de contrôle sous la contrainte d'une cadence d'itération de l'algorithme. Pour cela, aucune hypothèse n'est faite sur les caractéristiques temporelles des données d'entrée/sortie. Cette hypothèse a abouti à l'utilisation d'une technique d'ordonnancement particulière basée sur une progression du temps réalisée en fonction des événements de sorties des opérateurs. Comme nous l'avons remarqué, la sélection en moyenne du nombre d'opérateurs à mettre en œuvre peut faire aboutir à une architecture pipeline à posteriori de la synthèse alors qu'à priori elle ne l'était pas. Nous avons décrit les modifications apportées d'une part au calcul de la mobilité et d'autre part à la progression du temps afin de synthétiser un *GCG* ne contenant qu'un unique point de synchronisation référence. Nous avons proposé une recherche dichotomique restant à coder dans l'outil pour qu'il sélectionne et ordonnance les opérations de façon à optimiser le nombre d'opérateurs en présence de contraintes. Nous avons présenté les modifications à apporter à la génération de la machine d'états finis de l'unité de contrôle afin qu'elle synthétise un *GCG* contenant plusieurs points de synchronisation référence totalement séquencés.

Chapitre 5

Application au traitement du signal, de l'image et aux Télécommunications

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 la prise en compte des contraintes d'intégration dans un flot de conception basé sur la synthèse comportementale en terme d'optimisation du nombre de registres et d'opérateurs. La deuxième expérience compare les résultats obtenus en appliquant l'approche d'intégration proposée dans ce manuscrit avec les résultats des méthodes dites "classiques". La dernière expérience est un exemple d'utilisation de notre méthodologie appliqué à un composant virtuel comportemental complexe dans un contexte applicatif temps réel.

1. Choix des applications

Dans ce chapitre nous exposons les résultats de synthèses d'unités de traitement réalisées sous contraintes temporelles. Le choix des applications à synthétiser a été guidé par l'expérience acquise au LESTER dans les domaines du traitement du signal, de l'image et des télécommunications.

Nous débutons ce chapitre par la présentation d'une fonction de transformation : la Transformée de Fourier Rapide. Cette application est générique et se retrouve dans de très nombreuses applications du traitement du signal. Cette première expérience met en avant l'intérêt de la prise en compte des contraintes d'E/S dans un processus d'intégration de composant virtuel algorithmique par la synthèse haut niveau. Elle permet de plus d'illustrer la démarche d'allocation dichotomique proposée dans le chapitre précédent.

La deuxième expérience compare les résultats obtenus par une méthode dite "classique" d'intégration (composant virtuel RTL et adaptateur), avec les résultats obtenus par la méthode proposée dans ce manuscrit. Nous utilisons pour cela un algorithme de Transformée en Cosinus Discrète *(DCT)* sur laquelle sont basées entre autres les méthodes de compression d'images et de vidéo telle que *JPEG* [ISO94A] *et MPEG*-2 [ISO94B].

Nous terminons enfin ce chapitre par une application orientée télécommunications en présentant les résultats de synthèse obtenus sur une fonction de décodage convolutif basée sur le calcul du Maximum A Posteriori *(MAP)*. Cette fonction est intégrée dans les systèmes corrigeant les erreurs de transmission tels que les turbo-décodeurs [BER93] utilisés dans la norme DVB-RCS [DVB00]. Cette expérience montre l'intérêt de la modélisation des contraintes de communication que nous avons proposée : La synthèse du composant virtuel est pilotée par les caractéristiques de l'architecture de communication au travers de laquelle il échange des données avec son environnement et permet d'obtenir un composant virtuel adapté au contexte d'intégration. Enfin, cette expérience illustre la capacité que présente notre méthode et outil à traiter des problèmes industriels de grande complexité.

2. Transformée de Fourier Rapide

2.1. Présentation

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 5-1).

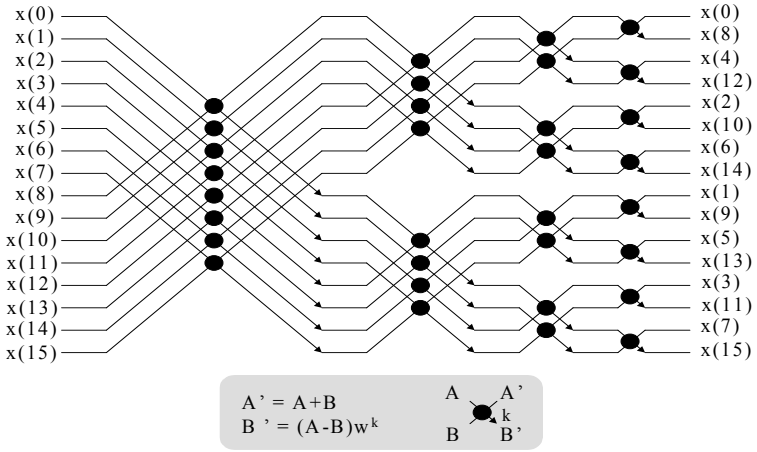


Figure 5-1: FFT Radix 2 Dif

2.2. Résultats de synthèse sans et sous contraintes d'E/S

2.2.1. Synthèse sans contraintes d'E/S

Nous avons dans un premier temps synthétisé à l'aide de l'outil GAUT une FFT 128 points à entrelacement temporel avec une contrainte de latence égale à dix fois le temps du chemin critique de l'algorithme soit 10x27 cycles = 270 cycles, et une période d'horloge de 30ns. Dans ce cas, la latence représente le temps entre la date d'acquisition de la première entrée et la date de production de la dernière sortie (voir Figure 5-2). Le système est supposé fournir les 128 données consommées par la FFT à une cadence d'un échantillon par cycle. Aucune contrainte

n'est formulée sur la cadence de production des résultats. La technologie utilisée comporte des additionneurs/soustracteurs et des multiplieurs ayant des latences respectives égales à 1 et 2 cycles.

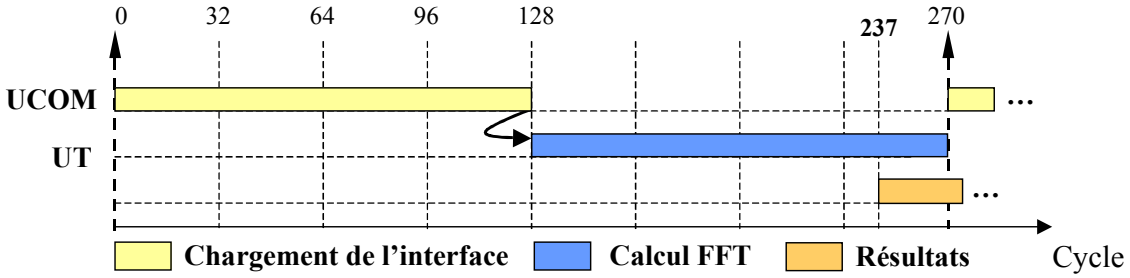


Figure 5-2 : Comportement temporel de la FFT synthétisée sans contrainte

Comme précédemment mentionné dans le chapitre 2 certains outils de synthèse haut niveau ne prennent pas en compte les contraintes d'E/S ou supportent une contrainte unique spécifiée sous la forme d'une latence globale ou d'une cadence d'itération. Ainsi, il est nécessaire de séparer la phase d'acquisition des entrées de la phase de traitement et de production des résultats. Afin de prendre en compte cette caractéristique et de garantir la présence des données sur les ports de *l'UT*, nous avons ôté à la latence totale, 128 cycles correspondant à la phase d'acquisition des entrées. Ainsi, la latence spécifiée pour la synthèse est de : 270-128 = 142 cycles (voir Figure 5-2). Le résultat de la synthèse est présenté dans le Tableau 5-1. Le composant, résultant de cette première synthèse, sera dénommé dans la suite du document : FFT1.

Latence	Soustracteurs	Additionneurs	Multiplieurs	Registres UCOM	Registres UT
4260 ns	12	7	25	256	603

Tableau 5-1 : Résultats de synthèse du composant FFT1

2.2.2. Synthèse sous contraintes d'entrées

Nous avons dans un deuxième temps réalisé la synthèse sous contrainte du même algorithme FFT 128 points en imposant un temps de latence totale identique de 270 cycles (Figure 5-3). Le composant produit par cette deuxième phase de synthèse sera dénommé FFT2 dans le reste du document. La FFT réalise un entrelacement temporel : l'unité de traitement débute les calculs 64 cycles après l'initialisation de la communication (chargement de l'interface). Dans ce contexte, l'unité de communication réalise le brassage des données d'entrée. Comme l'indique la Figure 5-3, la prise en compte des contraintes d'entrées autorise un recouvrement temporel entre l'acquisition des données dans l'unité de communication et la réalisation des calculs dans l'unité de traitement.

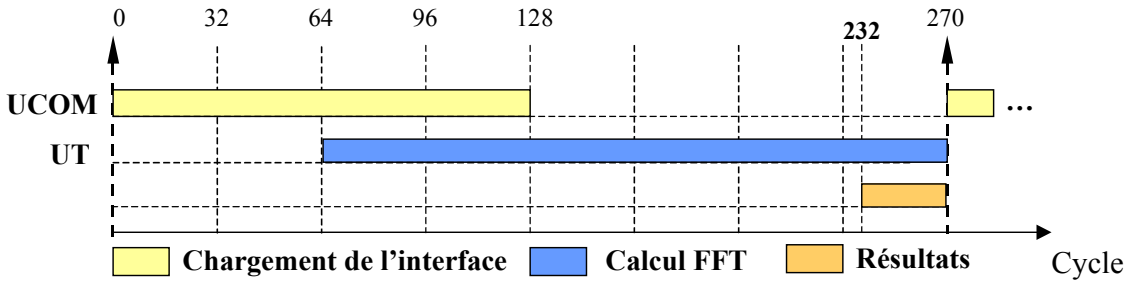


Figure 5-3 : Comportement temporel de la FFT synthétisée sous contraintes d'entrée

Nous avons appliqué l'algorithme de recherche dichotomique du nombre d'opérateurs à allouer que nous avons présenté dans le chapitre 4. Le Tableau 5-2 présente les résultats des deux premières phases de l'algorithme, qui délimitent l'espace de recherche en calculant la borne inférieure par une allocation en moyenne et la borne supérieure par une allocation dynamique.

Latence	Allocation	Soustracteurs	Additionneurs	Multiplieurs	Registres UT
8100 ns	Moyenne	6	4	13	626
	Dynamique	92	63	218	578

Tableau 5-2 : Borne pour la recherche dichotomique d'allocation

Les résultats de synthèse fournis par les phases d'allocation dichotomique sont décrits dans le Tableau 5-3. La synthèse sous contraintes apporte un gain de 28% du nombre de multiplieurs, 14% d'additionneurs et 25% de soustracteurs par rapport à la FFT1. **La conception de l'unité de communication étant réalisée a posteriori de la synthèse de l'unité de traitement, les registres ne sont pas partagés entre ces deux unités fonctionnelles. La prise en compte du coût des registres de l'UCOM et de l'UT durant la synthèse de l'unité de traitement, sera considérée dans des travaux futurs.**

Latence	Soustracteurs	Additionneurs	Multiplieurs	Registres UCOM	Registres UT	Synthèse	N° Tentative
8100 ns	49	33	115	X	478	OK	1
	27	18	64		532	OK	2
	16	11	38		554	OK	3
	11	7	25		566	OK	4
	8	5	19			NOK	5
	9	6	22		571	OK	6
	8	5	20			NOK	7
	9	6	21		569	OK	8
	9	6	20		568	OK	9
	9	6	19		557	OK	10
	9	6	13			NOK	11
	9	6	16			NOK	12
	9	6	17			NOK	13
	9	**6**	**18**	**256**	**557**	**OK**	**14**

Tableau 5-3 : Résultats d'ordonnancement de l'allocation dichotomique

Les synthèses ayant été réalisées sans contrainte de mémorisation, toutes les données sont mémorisées dans l'unité de traitement ce qui explique le nombre élevé de registres.

2.2.3. Synthèse sous contraintes d'entrées/sorties

Enfin, la synthèse de la FFT a été réalisée en ajoutant aux contraintes sur les données d'entrée décrit dans la section 2.2.1, une contrainte de production d'un résultat tous les cycles : le premier résultat doit être fourni à la date 143 ns et le dernier à la date 8100ns. La Figure 5-4 décrit le comportement temporel de la FFT après synthèse. Le Tableau 5-4 présente les résultats de synthèse sans décrire les résultats partiels issus des phases de l'allocation dichotomique.

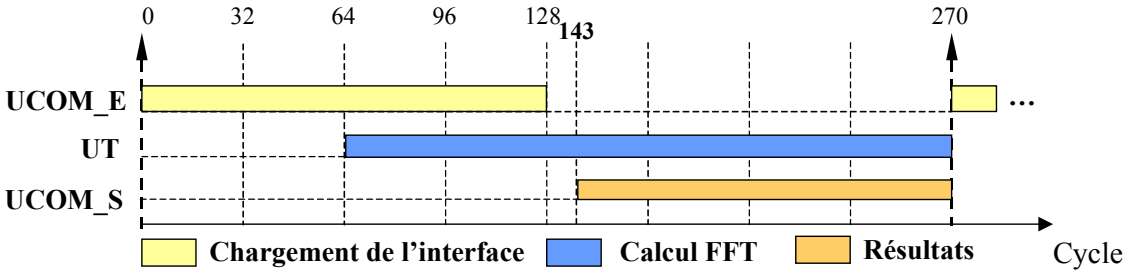


Figure 5-4 : Comportement temporel de la FFT synthétisée sous contraintes d'E/S

Latence	Allocation	Soustracteurs	Additionneurs	Multiplieurs	Registres UCOM	Registres UT
8100 ns	Dynamique	13	11	32	86	442
	Dichotomie	**10**	**8**	**19**	**66**	449

Tableau 5-4 : Résultats de synthèse sous contrainte d'entrée/sortie

Cette dernière expérience montre que pour cet exemple, l'ajout de contraintes sur les sorties aboutit à une synthèse d'une unité de traitement FFT3 qui contient plus d'opérateurs que le composant FFT2 mais moins que FFT1. Le nombre de registres est quant à lui inférieur. Les contraintes étant plus fortes, les durées de vie des données dans l'unité de traitement sont plus courtes, ce qui permet une plus grande réutilisation des registres et donc une meilleure optimisation.

2.3. Conclusion

Cette expérience a montré que la prise en compte des contraintes temporelles imposées par les données d'entrée/sortie durant la synthèse permet d'optimiser le nombre d'opérateurs et de registres à mettre en ouvre dans l'unité de traitement. En effet, lors d'une synthèse sans contrainte, l'unique solution pour prendre en compte le temps de transfert de l'ensemble des données servant à réaliser le calcul est d'en réduire la latence d'autant. A l'inverse, l'utilisation de

la fenêtre de transfert temporelle, durant laquelle les données sont acquises, permet d'obtenir une synthèse adaptée aux contraintes de l'application et aux caractéristiques de communications du système dans lequel doit être intégré le composant virtuel. **La quantité totale de registres de l'IP peut être optimisée en ajoutant au critère de synthèse de l'UT (1) le coût des registres inter opérateurs et (2) le coût des registres de l'UCOM (processus de synthèse itérative). Ce problème sera traité ultérieurement à ces travaux.**

3. Transformée en Cosinus Discrète

Cette deuxième expérience se propose de mettre en avant l'intérêt de l'utilisation de la synthèse haut niveau sous contrainte d'intégration appliquée à des composants virtuels de niveau algorithmique par rapport aux méthodes "classiques" d'intégration utilisant des IP RTL et des adaptateurs additionnels. L'algorithme retenu est une Transformée en Cosinus Discrète dont les principes sont rappelés. Ainsi, nous supposons, dans une première partie d'expérience, que le système, dans lequel le composant virtuel est réutilisé, fournit et acquière les données dans l'ordre de consommation et de production du composant DCT. Seules des perturbations sur les caractéristiques temporelles des transferts ont été introduites et ce tout en conservant un débit moyen constant. Dans une deuxième partie d'expérience, nous ajoutons aux perturbations temporelles une modification de l'ordre des données échangées.

3.1. Présentation

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 fonction 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 deux dimensions (horizontale et verticale) transforme les valeurs de luminance (ou chrominance) discrète 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 :

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

$$\text{avec} \quad \left\{ \begin{array}{l} u = 0,...,7 \\ v = 0,...,7 \\ c(k) = \frac{1}{\sqrt{2}} \quad \text{si } k = 0 \\ c(k) = 1 \quad \text{sin on} \end{array} \right\}$$

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 5-5.

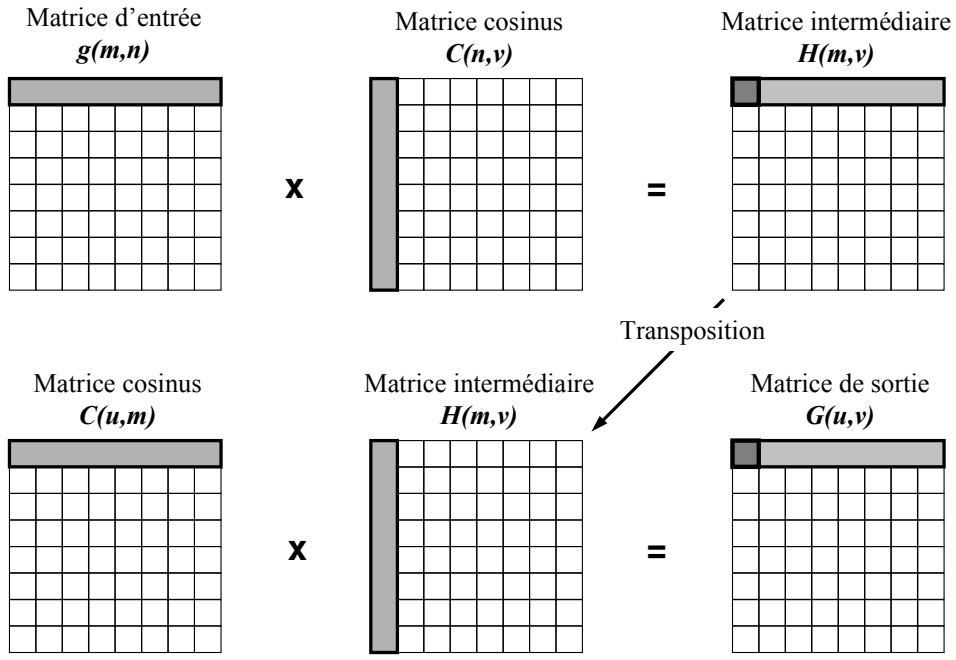


Figure 5-5 : Représentation matricielle d'une DCT-2D

3.2. Intégration d'IP pour différentes classes de débit d'E/S

Dans cette section nous comparons les résultats obtenus par une méthode dite "classique" d'intégration et par la méthode proposée. L'approche traditionnelle utilise un composant DCT au niveau RTL, conçu à l'aide d'outil de synthèse haut niveau ou RTL, et adapte les contraintes d'intégration aux contraintes du composant à l'aide d'un *wrapper* composé d'éléments mémorisants et d'un contrôleur (voir Figure 5-6). La méthode proposée repose quant à elle sur la synthèse architecturale sous contraintes d'intégration d'un composant virtuel algorithmique réalisée à l'aide de l'outil *GAUT*.

3.2.1. Variations temporelles des transferts

A. Utilisation d'un wrapper

Nous supposons dans cette première partie d'expérience que le système, dans lequel le composant virtuel est réutilisé, fournit et acquière les données dans l'ordre de consommation et de production du composant DCT. Nous introduisons des perturbations sur les caractéristiques temporelles des transferts tout en conservant un débit moyen constant.

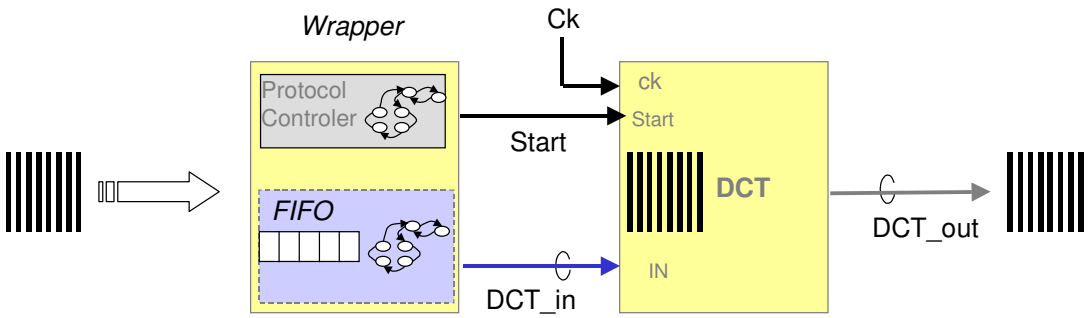


Figure 5-6 : Approche traditionnelle d'intégration

Dans ce contexte, nous avons centré notre étude sur la quantité de registres dédiés à la communication que contient le composant après intégration. Pour cela, nous avons dans un premier temps conçu une DCT au niveau RTL dont les caractéristiques sont les suivantes : la DCT lit ses pixels d'entrée tous les quatre cycles d'horloge dont la période est égale à 30ns dans un schéma d'acquisition colonne à colonne. Les caractéristiques de production des résultats sont identiques en terme de débit et de schéma de transfert (colonne/colonne).

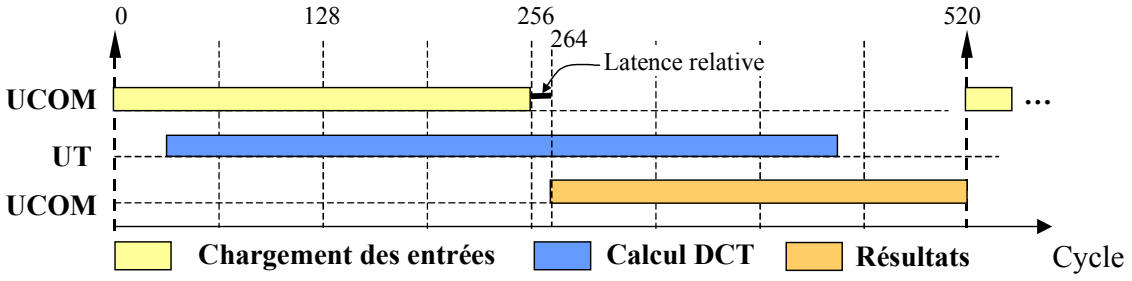


Figure 5-7 : Comportement temporel de la DCT RTL

Le composant contient 5 multiplieurs et 2 additionneurs (voir Tableau 5-5). Son unité de communication contient 7 registres pour mémoriser les entrées et 8 registres pour les sorties. La contrainte imposée sur la production du premier coefficient, une fois le dernier pixel consommé, est de 8 cycles et est nommée *latence relative* (voir Figure 5-7). La latence totale représente le délai entre l'acquisition du premier pixel et la production du dernier coefficient, et est égale à 520 cycles (256 d'acquisition + 8 cycles de latence relative + 256 cycles de production).

DCT RTL	Latence	Additionneur	Multiplieur.	Registres UCOM		
				E	S	Total
	520	2	5	*7*	*8*	15

Tableau 5-5 : Caractéristiques de la DCT de niveau RTL

Nous considérons maintenant que le système (bus partagé) ne peut strictement fournir les données suivant le schéma décrit précédemment. Afin de respecter les nouvelles caractéristiques temporelles des transferts, nous avons dimensionné un *wrapper* optimisé qui réalise la synchronisation et la temporisation des données. Les ordres, d'acquisition et de production, sont identiques à ceux du composant RTL et rendent ainsi possible l'usage d'une FIFO dans l'architecture du wrapper. Plusieurs caractéristiques ont été testées sur les types de transferts. Les paramètres sont la taille d'une rafale *(burst)* et le délai entre chaque transfert. Le délai séparant la transmission de deux données d'une même rafale est de un cycle. Le débit moyen considéré dans le contexte d'intégration est égal au débit initial de un pixel tous les quatre cycles mais comporte des pics de transferts comme le décrit la Figure 5-8.

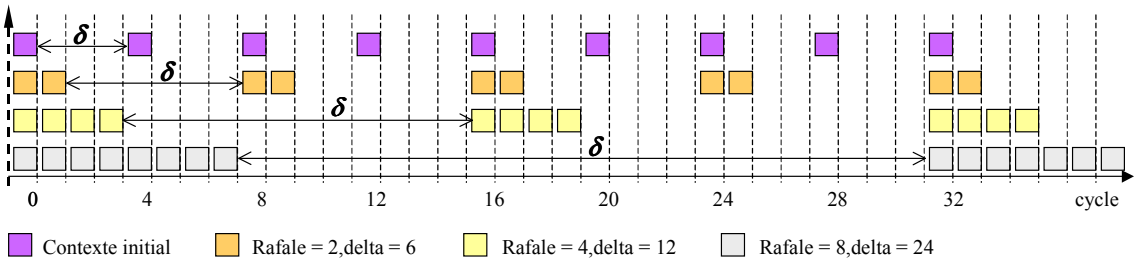


Figure 5-8 : Description des communications pour un débit moyen de 0,25 pixels/cycle

Le Tableau 5-6 décrit les résultats obtenus pour l'intégration de la DCT décrite au niveau RTL réalisée à l'aide d'un *wrapper*: La valeur δ fait référence au délai qui sépare deux rafales lors d'une séquence de transfert. Le wrapper n'introduit aucun *overhead* (latence nécessaire pour effectuer la temporisation et la synchronisation des données). La *latence globale* (temps entre l'acquisition de la dernière entrée et la production du premier résultat) qui est la somme de la latence de la DCT et de l'*overhead* introduit par le *wrapper* est donc constante. La *quantité totale de registres* est égale à la somme des registres dédiés à la communication contenus dans la DCT RTL (voir Tableau 5-5) et dans la FIFO du wrapper (colonne *Registres* du Tableau 5-6).

Taille d'une rafale	Delta	Overhead *(wrapper)*	Latence totale	Taille FIFO *(wrapper)*	Quantité totale de Registres *UCOM DCT + FIFO wrapper*
2	6	0	**520**	2	**17**
4	12			4	**19**
8	24			8	**23**
16	48			16	**31**
32	96			32	**47**

Tableau 5-6 : Résultats d'intégration de la DCT par utilisation d'un wrapper

B. Synthèse sous contrainte

Comme précédemment mentionné, nous avons dans un deuxième temps appliqué la méthodologie d'intégration que nous proposons. Afin de générer un graphe de contraintes globales, nous avons introduit des contraintes temporelles minimum et maximum entre les nœuds d'entrée/sortie du graphe *ACG* dérivé du modèle *IOCG*. Comme pour l'intégration du composant RTL, la latence de calcul est définie par une contrainte de temps maximum entre l'acquisition de la dernière entrée et la production du premier résultat. Cette contrainte a été spécifiée par une connexion arrière allant du premier nœud de sortie au dernier nœud d'entrée du graphe *IOCG* (voir Figure 5-9). La contrainte de production d'un coefficient tous les quatre cycles a été spécifiée par un couple d'arc minimum/maximum (avant et arrière) de poids *w* identiques entre chaques nœuds sorties du *IOCG*. Les données d'entrée appartenant à une même rafale ont été regroupées dans un nœud hiérarchique et reliées par un couple d'arcs modélisant les caractéristiques d'une rafale : une donnée tous les cycles. Ces nœuds hiérarchiques sont eux même reliés par un couple d'arcs représentant le délai *delta* les séparant temporellement.

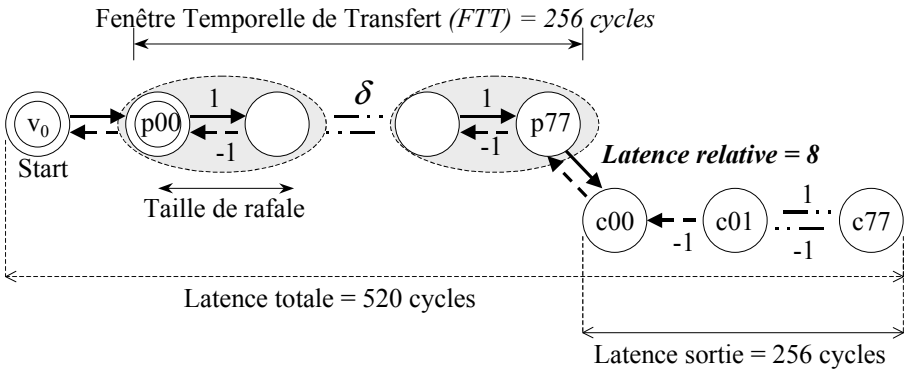


Figure 5-9 : Modèle IOCG pour l'intégration de la DCT

Le Tableau 5-7 présente les résultats obtenus, pour l'intégration d'un composant virtuel algorithmique, en appliquant notre méthode. L'approche proposée permet un gain significatif du nombre de registres dédiés à la communication et ce sur la quasi-totalité des schémas de transfert exprimés. Il faut cependant noter que nous obtenons un nombre de registres supérieurs à celui fourni par la solution à base de wrapper pour une taille de rafale égale à 32 données. Cette perte reste cependant négligeable au regard de la complexité de conception d'un adaptateur.

Taille d'une rafale	δ	Multiplieurs	Additionneurs	Latence	Registres UCOM			
					E	S	Total	Gain
2	6	5	2	520	7	8	15	**11%**
4	12				7	8	15	**21%**
8	24				7	8	15	**34%**
16	48				14	12	26	**16%**
32	96				28	20	48	**–2%**

Tableau 5-7 : Résultats d'intégration de la DCT par la synthèse sous contrainte

3.2.2. Variations des schémas d'acquisition et de production des données d'E/S

A. Utilisation d'un *wrapper*

Dans cette deuxième partie d'expérience, nous ajoutons aux perturbations temporelles une modification de l'ordre des données échangées. Ainsi, le système, dans lequel le composant virtuel est réutilisé, fournit et acquière les données dans un schéma ligne/ligne contrairement au composant RTL qui les consomme et les produits suivant un ordre colonne/colonne. Cette modification de l'ordre des transferts requiert l'utilisation d'un adaptateur d'entrée et d'un adaptateur de sortie (voir Figure 5-10). L'utilisation de FIFO dans les adaptateurs n'est plus possible : les permutations des E/S nécessitent des éléments mémorisants tels que les mémoires ou les bancs de registres. La conversion des séquences d'entrée/sortie nécessite une temporisation des données ce qui introduit une latence appelée *overhread*.

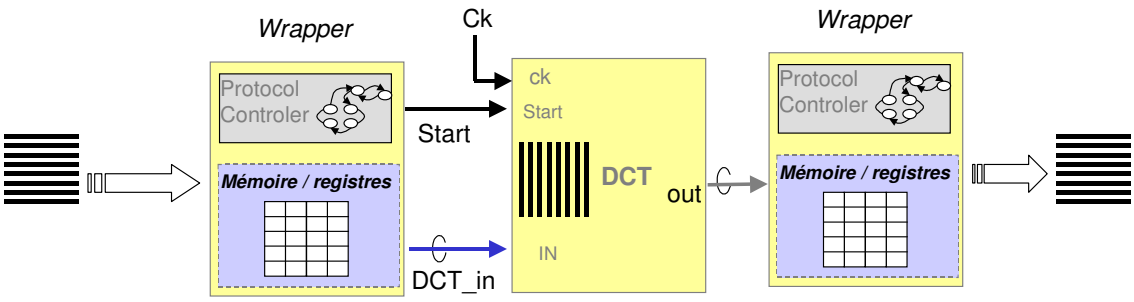


Figure 5-10 : Intégration utilisant des adaptateurs d'E/S

Le Tableau 5-8 présente le nombre optimisé de registres ainsi que la latence introduite par les wrappers d'entrée et de sortie. Le nombre de registres total fait référence à la quantité de registres contenus dans les wrappers à laquelle s'additionne les registres de l'unité de communication de la DCT RTL. La latence totale est quant à elle égale à la somme de l'*overhead* et de la latence de calcul de la DCT. Bien que le débit moyen des E/S considéré dans cette expérience soit constant, la latence introduite par le wrapper varie. Ceci est dû aux différentes possibilités de recouvrements temporels, fonction des valeurs du couple (Taille d'une rafale, delta), entre l'acquisition des dernières données dans le wrapper et la production des premières données vers la DCT.

		Wrapper d'entrée		Wrapper de sortie			
Taille d'une rafale	**Delta**	**Overhead**	**Registres**	**Overhead**	**Registres**	**Registres total**	**Latence totale**
2	6	196	64	106	54	**133**	**822**
4	12	196					**822**
8	24	196					**822**
16	48	172					**798**
32	96	124					**750**

Tableau 5-8 : Résultats d'intégration de la DCT par utilisation de wrappers d'E/S

B. Synthèse sous contrainte

Comme dans la section précédente, nous avons dans un deuxième temps synthétisé l'algorithme de DCT en utilisant les contraintes d'intégration. Le Tableau 5-9 résume les résultats de synthèse. Les solutions fournies par notre approche autorisent des gains importants en terme de latence et nombres de registres.

Taille d'une rafale	**Delta**	**Multiplieurs**	**Additionneurs**	**Latence**		**Registres UCOM**			
					gain	**E**	**S**	**Total**	**Gain**
2	6	5	2	520	**36%**	2	14	16	**87%**
4	12				**36%**	7	13	20	**84%**
8	24				**36%**	15	13	28	**78%**
16	48				**34%**	24	14	38	**71%**
32	96				**30%**	40	15	55	**58%**

Tableau 5-9 : Résultats d'intégration de la DCT par la synthèse sous contrainte

Nous n'avons pas mentionné pour cette expérience, le nombre de registres contenu dans l'unité de traitement puisque à l'heure actuelle leur coût n'est pas pris en compte durant la synthèse. Il faut cependant noter que sans une gestion efficace des registres dans l'UT, la prise en compte des contraintes peut accroître considérablement leur nombre et ainsi remettre en cause l'approche proposée.

3.3. Conclusion

La méthode basée sur la synthèse sous contrainte permet d'utiliser la fenêtre temporelle de transfert pour l'ordonnancement des opérations. Le recouvrement des phases de calcul et des phases de communication aboutit en une quantité de registres de communication inférieure à celle de la DCT de niveau RTL couplé à un adaptateur additionnel. Cette utilisation de la phase de transfert de données entre le système et le composant virtuel permet aussi une réduction

significative de la latence. Il faut cependant remarquer que le recouvrement temporel des phases de calculs et de communication n'est possible que lorsque les dépendances de calculs et les séquences d'entrées coïncident. La phase d'adaptation temporelle des séquences d'entrées réalisée par le wrapper introduit une latence qui additionnée au temps de calcul de la DCT est toujours supérieur ou égale à celle de l'architecture obtenue par l'approche que nous proposons. Cette latence supplémentaire peut, dans certain cas, faire que le composant, une fois intégré, ne respecte pas les contraintes temporelles imposées par l'application et l'architecture SoC dans lequel il doit être intégré. Il est important de remarquer que les composants virtuels de niveau RTL peuvent n'être réutilisés que dans des applications dont le débit est inférieur ou égal au débit pour lequel ils ont été conçus. De plus, les forts volumes de données échangées dans les applications TDSI peuvent aboutir à la conception de wrapper dont la surface est plus importante que celle du composant. Comme nous l'avons vu dans cette série d'expériences, notre approche autorise un gain non négligeable en terme de latence et de quantité de registres. Elle permet de plus une automatisation et donc un gain de temps de conception non négligeable comparée aux méthodes utilisant des adaptateurs qui, selon les types d'adaptations requis, peuvent être extrêmement complexes à concevoir et optimiser.

4. Adéquation Algorithme/Architecture : *MAP / turbo codes*

Cette dernière expérience est réalisée sur un composant virtuel implémentant l'algorithme du Maximum A Posteriori (MAP) (ou Forward-Backward) qui constitue le cœur du décodage convolutif d'une famille de codes correcteurs d'erreurs appelée « Turbo-codes » [BER93]. Elle met en avant l'applicabilité et l'intérêt de notre approche de synthèse sous contraintes pour un algorithme de forte complexité. Les contraintes utilisées pour cette expérience correspondent aux chemins critiques relatifs à chacune des sorties. Elles visent à produire une architecture des plus rapide en terme de latence répondant ainsi au critère temps réel de la norme DVB-RCS.

Nous rappelons dans un premier temps les fondements théoriques sur lesquels repose l'algorithme MAP. Nous décrivons ensuite, dans son ensemble, la démarche Adéquation Algorithme/Architecture (AAA) aboutissant à la sélection d'un algorithme spécifique puis présentons pour ce dernier les résultats de synthèse que nous avons obtenus.

4.1. **Présentation**

L'algorithme du Maximum A Posteriori (MAP) [CHA66][BAH74] (appelé aussi algorithme Forward-Backward (FB) ou Aller-Retour) permet de décoder un code correcteur d'erreur convolutif en générant la fiabilité des bits décodés. Le principe des codes convolutifs est simple : les bits du message que l'on veut transmettre servent à faire évoluer une machine à états (un simple registre à décalage). Pour chaque bit entrant dans la machine à états, une signature X_k caractérisant la transition est transmise au destinataire. Le récepteur reconstruit, à partir de la suite des signatures de transitions bruitées qu'il reçoit, l'évolution la plus probable de la machine à états de l'émetteur, c'est-à-dire le message transmis le plus probable. Pour

simplifier, on peut considérer comme codeur un simple registre à décalage de longueur ν. Il peut donc prendre 2^{ν} états possibles. L'évolution de la machine à états est représentée par un treillis comme indiqué dans la Figure 5-11. A chaque instant, l'ensemble des états possibles est représenté dans une colonne ; chaque état se caractérise par les valeurs des registres R_1 et R_2 de la machine à états. Les transitions possibles entre deux états consécutifs sont représentées par des branches. Par exemple la transition, partant de l'état s_{01} (R_1= 0, R_2 = 1), correspondant à l'arrivée d'un bit u_1 = 1 à l'instant k=1 et arrivant à l'état s_{10} à l'instant k = 2, est représentée en gras sur la Figure 5-11. La signature de transition est donnée par le vecteur X_k. Ce vecteur est calculé comme une fonction linéaire à partir du bit u_k entrant et de l'état courant s du codeur. Les composantes du vecteur X_k servent à moduler un signal qui est transmis au récepteur à travers un canal bruité. Ce dernier reçoit alors le signal représenté par le vecteur $Y_k = X_k + B_k$, avec B_k représentant le vecteur bruit. Dans la suite du document, Y_k sera désigné comme le symbole reçu.

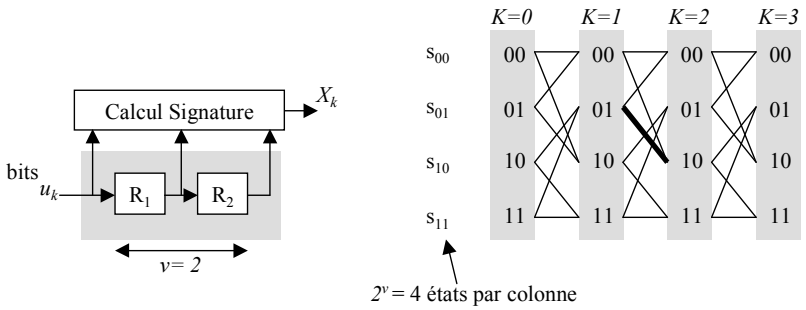


Figure 5-11 : Logigramme d'un codeur convolutif et représentation du treillis associé

Sans plus entrer dans les détails, il est intéressant de noter que l'algorithme MAP permet, par l'exploration exhaustive de tous les chemins possibles dans le treillis, de déterminer la probabilité à posteriori (connaissant l'observation de la séquence bruitée reçue) de chacun des bits entrés dans le codeur. Le principe sous-jacent à ce calcul consiste à déterminer la vraisemblance de chaque chemin, c'est-à-dire, la distance entre la suite des $\{Y_k\}$ reçue et la suite des signatures $\{X_k\}$ associée à chaque chemin. Le lecteur intéressé par l'aspect algorithmique du processus pourra consulter avec profit [CHA66][BAH74]. L'algorithme MAP est décrit, dans le présent document dans le domaine des logarithmes (algorithme Max-Log-MAP) qui est mieux adapté à une réalisation matérielle. Il se déroule en quatre étapes : calcul des métriques de branche, récursion Aller et récursion Retour, calcul des rapports de vraisemblance (LLR pour Log-Likelihood Ratio). Dans ce qui suit, N représente la taille du mot de code.

Calcul des métriques de branches : (BMC pour Branch Metric Computation)

Le calcul de la métrique de branche $G_k(s',s)$ de la branche entre les états s' à l'instant k et s à l'instant k+1, est déterminé comme le produit scalaire entre le symbole $Y_k=(y_k^i)_{0\le i<r}$ reçu à l'instant k et la signature $X_k(s',s)=(x_k^i(s',s))_{0\le i<r}$ associée à la transition entre l'état s' et s.

$$G_k(s',s)=\sum_{i=1}^{r} x_k^i(s',s)\cdot y_k^i \qquad (1)$$

où r est la dimension du vecteur X_k (typiquement 2) et où les $x_k^i(s',s)$ peuvent prendre les valeurs 1 ou –1.

Récursion Aller (ou Forward) :

La récursion Aller correspond au calcul des métriques de nœud A_k associées aux états du treillis. La métrique de nœud $A_k(s)$ associée au nœud s à l'instant k correspond à la vraisemblance, connaissant l'observation des vecteurs Y_i reçus entre les instants $i = 0$ et $i = k$-1, que le codeur se trouve dans l'état s à l'instant k. Les métriques A_k sont calculées récursivement à partir des A_{k-1} et des métriques de branches G_{k-1} de l'instant k-1, comme l'indique le pseudo algorithme suivant : le vecteur A_0 initial indique simplement que le codeur à l'état initial ($k = 0$) se trouve dans l'état $s = 0$ (tous les bits du registre du codeur sont à zéro).

- $A_0(s)$=0, si s=0, $A_0(s)$= -∞ sinon.
- pour k = 0..N-2
  - pour les 2^ν états s du treillis
    - $A_{k+1}(s) = MAX\left[(A_k(s'_0)+G_k(s'_0,s), A_k(s'_1)+G_k(s'_1,s)\right]$

où s'_0 et s'_1 sont les deux nœuds antécédents du nœud s dans le sens direct du treillis. L'opérateur MAX(X,Y) renvoie la plus grande des deux variables. L'opération élémentaire sur chaque nœud est constituée de 2 Additions, d'une Comparaison et d'une Sélection du maximum (ACS).

Récursion Retour (ou Backward) :

La récursion Retour est symétrique de la récursion Aller. La métrique de nœud $B_k(s)$ associée au nœud s à l'instant k correspond à la vraisemblance que, connaissant l'observation des vecteurs Y_i reçus entre les instants $i = k$ et $i = N$ - 1, le codeur se trouve dans l'état s à cet instant, comme l'indique le pseudo algorithme suivant :

- $B_N(s)$=0, si s=0, $B_N(s)$= -∞ sinon.
- pour k = N..2
  - pour les 2^ν états s du treillis
    - $B_{k-1}(s) = MAX\left[(B_k(s'_0)+G_k(s, s'_0), B_k(s'_1)+G_k(s, s'_1) \right]$

où s'_0 et s'_1 sont les deux nœuds successeurs du nœud s dans le sens direct du treillis.

Calcul de la sortie (EXT) :

Le calcul de la sortie tient compte de l'ensemble des informations apportées par tous les symboles de la trame : les informations en provenance du passé (A_k, calculé à partir des instants $k' < k$), celles venant du futur (B_{k+1}, calculé à partir des instants $k'' \geq k+1$) et du présent (G_k).

$$H_k(s',s)=A_k(s')+G_k(s',s)+B_{k+1}(s) \quad (2)$$

$$L_k^0=\underset{(s'_0,s_0)\in \ell_0}{MAX}\left(H_k(s'_0,s_0)\right) \text{ et } L_k^1=\underset{(s'_1,s_1)\in \ell_1}{MAX}\left(H_k(s'_1,s_1)\right) \quad (3)$$

$$LLR(u_k)=L_k^1-L_k^0 \quad (4)$$

où ℓ_1 est l'ensemble des couples associés à une transition du treillis correspondant à la réception d'un bit $u_k = 1$ dans le registre à décalage et où ℓ_0 est l'ensemble des couples d'états correspondant à la réception d'un 0.

4.2. Application de l'approche AAA

Cet algorithme pose un problème de latence de décodage, puisqu'il faut théoriquement attendre les N données du message avant de commencer la récursion Retour (N peut être de l'ordre de plusieurs milliers). Pour contourner ce problème, une solution sous-optimale, proposée dans [DIN99] est utilisée. Cette solution utilise, pour tous les L symboles, un estimateur du vecteur états B_{kL} afin d'anticiper le démarrage de la récursion Retour. Utilisons la même représentation graphique que dans [BOU93] pour expliquer l'ordonnancement des calculs. Considérons le cas de la Figure 5-12 pour lequel $N = 4L$. Dans cette figure, l'axe horizontal représente le temps, avec le cycle symbole comme unité. L'axe vertical représente les indices des symboles reçus. Ainsi, la droite ($x = y$) indique qu'à l'instant k, le symbole Y_k est reçu et peut donc être traité.

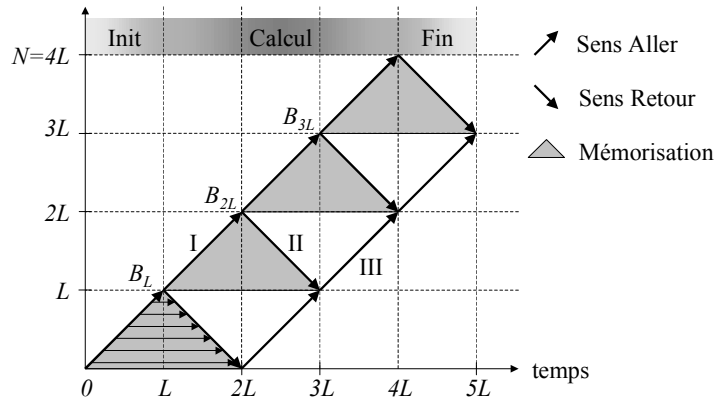


Figure 5-12 : Algorithme Aller - Retour découpé en 3 phases (initialisation, calcul et fin).

Dans cette figure, en dehors des effets de bord (phase d'initialisation et phase d'achèvement), il apparaît un motif de calcul, le triangle, qui est répété avec une périodicité L en fonction du temps et des indices des données. Le triangle indique le séquencement des calculs pour décoder un segment de L données. Considérons les données d'indices compris entre L et $2L$. Entre les instants L et $2L$, la récursion Aller est réalisée au fur et à mesure que les symboles arrivent (flèche I du schéma). A chaque cycle, l'ensemble des métriques de nœud A_k est mémorisé (zone grise du schéma). Entre les instants $2L$ et $3L$, une récursion Retour est réalisée sur les symboles d'indice $2L$ jusqu'à L (flèche II de pente égale à –1 d'après le schéma). Cette récursion commence à partir d'un état B_{2L} préalablement estimé. A chaque cycle, outre le calcul de la récursion Retour, le vecteur A_k correspondant est lu dans la mémoire afin d'utiliser directement le vecteur B_{k+1} généré pour calculer le LLR du bit décodé. Enfin, les bits sont ré-ordonnés (flèche III).

4.2.1. Adéquation Algorithme Architecture

Comme le montre la Figure 5-13, le motif de calcul (triangle élémentaire décrit dans la Figure 5-12) permet de réaliser le traitement des N données par un ensemble de p composants *SubMAP* utilisant L données**.** Ce motif élémentaire est constitué d'une *"récursion Aller"* et d'une *"récursion Retour"* et les composants *SubMAP* sont chaînés comme suit: le $i^{ème}$ bloc utilise les poteaux avants *(fwd_pot)* fournis par le bloc $(i\text{-}1)^{ème}$ bloc et les poteaux arrières *(bwd_pot)* fournis par le $(i+1)^{ème}$ bloc. Les informations, *a priori* et *décodeur*, distribuées sur chacun des blocs, proviennent de mémoires extérieures.

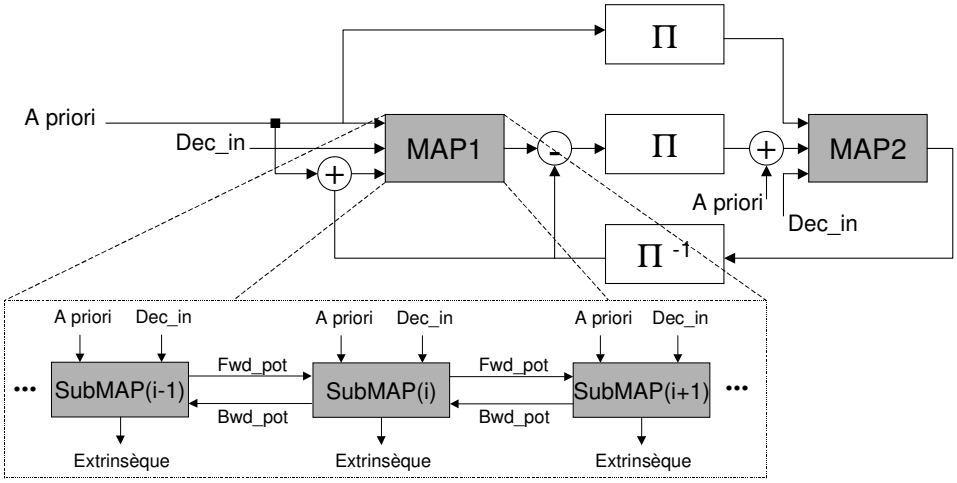


Figure 5-13 : Architecture séquentiel du décodeur

La solution la plus naturelle, qui consiste à utiliser les données αL à $(\alpha+1)L$ pour la récursion avant et la récursion arrière dans un même *SubMAP*, permet un chaînage des traitements comme décrit dans la Figure 5-12. Cependant, ce type de chaînage nécessite l'utilisation de deux composants *SubMAP* s'exécutant en parallèle, et donc la mémorisation de *2L* métriques de nœuds. Une réduction de la quantité de mémoire peut être obtenue en utilisant un unique composant *SubMAP* réalisant les traitements de façon séquentielle *:* les triangles successifs ne se chevauchent plus comme le montre la Figure 5-14.

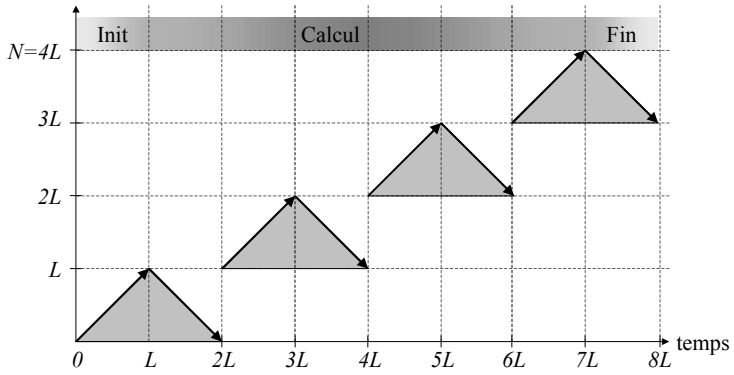


Figure 5-14 : Chaînage de *SubMAP* utilisant les données αL à $(\alpha+1)L$

La Figure 5-15 décrit le comportement temporel de cet algorithme pour lequel la production de l'information extrinsèque est requise au plus tôt, c'est à dire après une durée égale à la somme des temps d'exécution des opérations contenues dans le chemin critique du *SFG*. L'unité du temps de cycle est un ACS (Addition, Comparaison Sélection) soit 4 cycles d'horloge de période 8ns. Les premiers résultats sont fournis après une latence théorique égale à :

Latence = $(L+2)*T_{ACS}$ + latence relative

Latence relative = $T_{MB} + T'_{EXT} + \Delta_{ECR} + \Delta_{LECR}$

Où T_{MB} représente la durée minimum requise pour calculer une métrique de branche, T_{EXT} représente la durée minimum requise pour calculer une donnée extrinsèque à partir de la métrique de branche, et Δ_{ECR} et Δ_{LECR} représentent les temps d'accès en écriture et en lecture à l'unité de communication

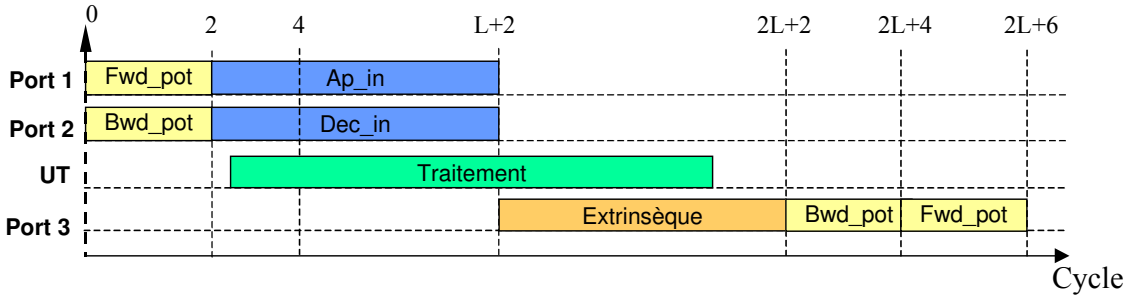


Figure 5-15 : Comportement temporel d'un composant *SubMAP*

La Figure 5-16 représente le graphe de contraintes *IOCG* modélisant les contraintes de cadence sur les entrées, la latence relative et la cadence de production des sorties. Ce graphe ne contient qu'un unique nœud de synchronisation : le nœud v_0 qui symbolise ici le temps absolu t = 0, pour l'expression des contraintes sur les dates de transferts des E/S.

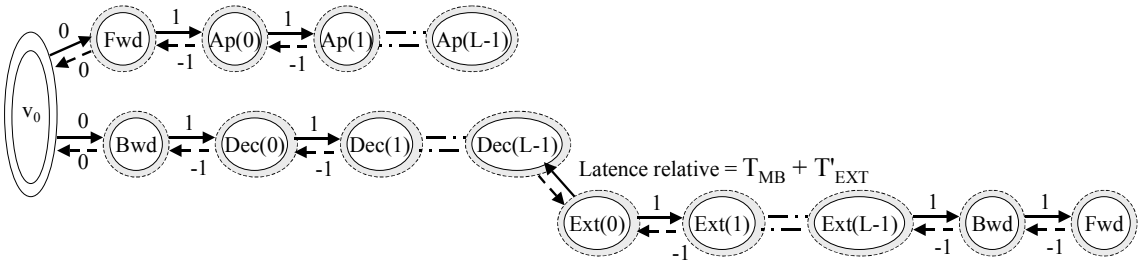


Figure 5-16 : IOCG Modélisant les contraintes d'E/S

Afin de contourner les problèmes de duplication de la mémoire et/ou de non chevauchement temporel des calculs, l'utilisation d'un algorithme effectuant la récursion avant sur les données αL à $(\alpha+1)L$ et la récursion arrière sur les données $(\alpha-1)L$ à αL s'avère nécessaire (voir Figure 5-17) [GNA03].

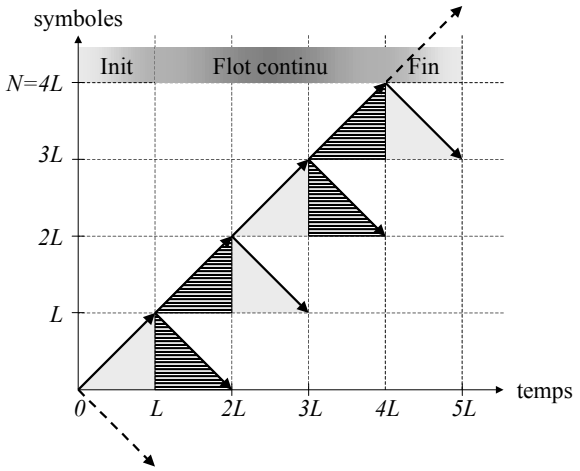


Figure 5-17 : Chaînage de *SubMAP* modifiés

Dans ce cas, la récursion arrière s'effectue sur un ensemble de données stockées dans la mémoire interne du composant et produites par la récursion avant. Ainsi, l'utilisation de cet algorithme modifié permet le chaînage des traitements. Cependant, comme le montre la Figure 5-18, l'expression de la latence de calcul des informations extrinsèques, requiert une spécification différente de celle utilisée pour l'algorithme précédemment décrit. En effet le calcul des informations extrinsèques utilise uniquement des données provenant de la mémoire interne du composant et peut donc commencer à s'exécuter à t = 0. Ainsi, le chemin critique pour produire la première valeur extrinsèque a une valeur égale à $T_{EXT} + \Delta_{ECR}$. Les informations extrinsèques peuvent cependant être produites à une cadence d'une donnée tous les 4 cycles ACS.

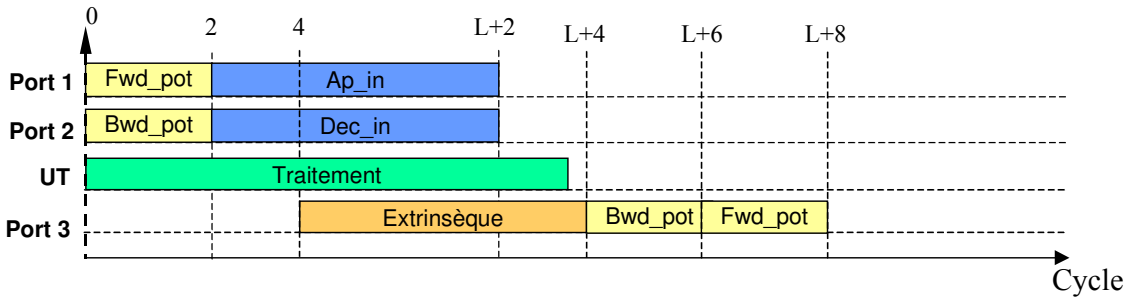


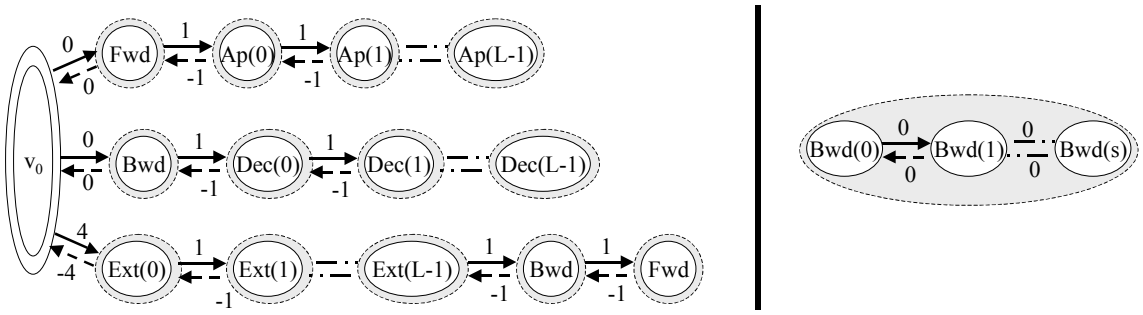
Figure 5-18 : Comportement temporel d'un composant SubMAP modifié

4.2.2. Synthèse sous contraintes d'un composant sous MAP

Nous avons décrit dans la section précédente comment dans une démarche AAA un concepteur système choisissait un algorithme spécifique. Nous présentons maintenant les résultats de synthèse sous contraintes que nous avons obtenu à l'aide du logiciel *GAUT*.

La Figure 5-19(a) représente le graphe de contraintes *IOCG* utilisé pour effectuer la synthèse de l'algorithme *subMAP*. Les nœuds hiérarchiques décrits dans la Figure 5-19(b) sont utilisés pour exprimer l'arrivée simultanée d'ensembles de données d'entrée. Ainsi, les 8 données constituant l'ensemble des poteaux arrière, sont fournies par paquets de 4 données, chaque paquet étant transmis simultanément. Il en est de même pour l'ensemble des données d'entrées sorties considéré dans cette expérience.

Il est important de remarquer que la date de production des sorties est exprimée, contrairement à la Figure 5-16, relativement au début de l'exécution symbolisé par le nœud v_0.



K. **(b)**

Figure 5-19 : Modélisation des contraintes d'E/S (a) IOCG, (b) nœud hiérarchique

Le SFG contient 95210 arcs et 44048 nœuds, répartis en 23168 nœuds opérations et 20880 nœuds opérandes. Le fichier source décrivant l'algorithme utilise 128 lignes de code VHDL comportemental pour spécifier l'ensemble des traitements. Le Tableau 5-10 présente la répartition par type de fonctions des nœuds opérations contenus dans le SFG.

	Inverseurs	Additionneurs	Soustracteurs	Min4	Min2	variables
Opérations	8448	11520	2304	768	128	20880

Tableau 5-10 : Composition du *SFG*

Le fichier VHDL RTL décrivant l'architecture du composant SubMAP_L_32 contient 2469272 lignes (soit 20 mille fois plus que le code source) générées en 40 minutes sur une machine Ultra-Sparc 64 bits, 900 MHz, Solaris8. Cette durée se décompose en 5 minutes pour la génération de la description intermédiaire GC, 4 minutes pour la transformation du GC en SFG, 2 minutes pour l'ordonnancement et 29 minutes pour la génération de l'unité de contrôle, de l'unité de traitement et du fichier VHDL les décrivant. La description RTL ainsi que le comportement de cette architecture ont été validés par simulation à l'aide du logiciel ModelSim de Modeltech version 5.5c. Les résultats fonctionnels fournis par la description RTL sont identiques à ceux du modèle spécifié en langage C.

L'architecture finale contient 64 inverseurs, 96 additionneurs, 32 soustracteurs, 52 comparateurs 2->1. La FSM pilotant l'unité de traitement est constituée de 144 états générant 8 bits de contrôle (voir Tableau 5-11). Les registres sont au nombre de 996, ce qui correspond à une moyenne de 3 registres par opérateur (le modèle architectural en imposant au minimum 2 pour 1).

FSM (nb états)	Inverseurs	Additionneurs	Soustracteurs	Comparateurs	Registres UT
144	64	96	32	52	996

Tableau 5-11 : Résultats de synthèse sous contraintes d'un *subMAP*

D'après la description algorithmique utilisée, le nombre attendu d'opérateurs est le suivant :

Calcul	Inverseurs	Additionneurs	Soustracteurs	Comparateurs	
Métrique de branche	4	32	16	0	
Aller		32		24	
Retour		32		24	
Extrinsèque	8	32	16	28	
	12	**96**	**32**	**72**	**Total**

Tableau 5-12 : Nombre attendu d'opérateurs par type de calcul

Le nombre d'opérateurs contenus dans l'architecture générée correspond à l'estimation que nous pouvions faire. Seule la quantité d'inverseurs alloués par *GAUT* est bien plus importante car ce dernier n'effectue pas de réduction d'expressions communes actuellement**. Des travaux sont en cours de réalisation sur cette fonctionnalité**. La différence du nombre de comparateurs provient de la réutilisation des opérateurs dans les calculs. **Notons qu'a terme, la réduction d'expressions communes permettra un gain plus important sur le nombre d'opérateurs et le nombre de registres.**

4.3. Conclusion

Cette expérience a décrit dans son ensemble une démarche Adéquation Algorithme/Architecture (AAA) aboutissant à la sélection d'un algorithme spécifique de Maximum a Posteriori. Les résultats de synthèse obtenus sur une fonction de décodage convolutif basée sur le calcul du Maximum A Posteriori *(MAP)* ont montré la validité des modifications apportées à l'outil de synthèse *GAUT*. Cette expérience a de plus montré l'intérêt de la modélisation des contraintes de communication que nous proposons: La synthèse du composant virtuel réalisée sous contraintes temporelles à permis d'obtenir un composant virtuel adapté au contexte d'intégration. Enfin, cette expérience a illustré la capacité que présente notre méthode et outil à traiter des problèmes industriels de grande complexité.

5. Conclusion

Les trois expériences que nous avons présentées dans ce chapitre ont mis en avant l'intérêt de l'approche que nous proposons. Ainsi, nous avons montré que l'utilisation d'une description de composant virtuel de niveau algorithmique couplé à la synthèse sous contraintes permet de générer des architectures adaptées à l'environnement dans lesquelles elles doivent être intégrées. Comparée aux méthodes d'intégration d'IP RTL utilisant des wrappers, notre approche autorise une réduction significative du coût matériel. Enfin, nous avons illustré la capacité que présente notre méthode et notre outil à traiter des problèmes industriels de grande complexité.

Conclusions et Perspectives

Nous avons proposé dans ce mémoire une approche de réutilisation des Ips dans les applications orientées traitement du signal, de l'image et des télécommunications. Pour cela, nous basons notre approche sur la notion de composants virtuels de niveau algorithmique, définie dans le cadre du projet *RNRT MILPAT (Méthodologie et Développement pour les Intellectual Properties pour Applications Telecom)*. Le flot de conception proposé s'inscrit dans la démarche Adéquation Algorithme Architecture du projet *RNRT ALITPA (Définition et Application d'une méthodologie de développement pour les (IP) intellectual property de niveau comportemental dans les applications de télécommunication)* et est basé sur l'utilisation de techniques de synthèse haut niveau sous contraintes d'intégration. Les unités fonctionnelles constituant l'architecture cible du composant sont (re)conçues en fonction des caractéristiques de l'architecture de communication du système et de la spécificité de l'application.

Dans ce contexte, la spécification de l'IP est modélisée par un Graphe Flot de Signaux *(SFG)* qui, couplé aux temps de propagations des opérateurs et à la cadence d'itération, permet la génération d'un graphe de contrainte algorithmique *ACG*. Nous avons développé une analyse formelle des contraintes, qui repose sur les calculs de cycles, et permet de vérifier la cohérence entre la cadence, les dépendances de données de l'algorithme et les contraintes technologiques.

Les contraintes d'intégration, spécifiées pour chacun des bus (ports) connectants l'IP aux autres composants du système, sont modélisées par un graphe de contraintes d'Entrée/Sortie *IOCG (IO Constraint Graph)*. Ce modèle supporte, entre autre, la modélisation (1) du type de transferts, (2) de l'indéterminisme dans les dates d'arrivées des données dû aux phases d'arbitrage pour l'accès au bus partagé, (3) du séquencement des données échangées (4) des mécanismes liés aux protocoles (synchrone / asynchrone)... Les contraintes d'intégration et les contraintes algorithmiques de l'IP sont fusionnées pour fournir un graphe détaillé des contraintes *GCG (Global Constraint Graph)* exhibant les points de synchronisation entre l'environnement et le composant. Des optimisations pour l'implémentation sont proposées à partir de transformations formelles du graphe.

La synthèse de l'unité de traitement *(UT)* est réalisée à l'aide de l'outil *GAUT* (Générateur Automatique d'Unité de Traitement) dont l'ordonnancement est contraint par les paramètres

temporels de l'intégrateur du composant virtuel. La synthèse de l'*UT* génère un ensemble de contraintes aux E/S, modélisé sous la forme d'un *IPERM (IP Execution Requirement Model)*. Ce dernier spécifie (1) les dates de production et de consommation des données dans l'UT et (2) les bus sur lesquels elles transitent entre l'unité de communication et l'unité de traitement. Ces modèles *IPERM* et *IOCG* seront à terme utilisés pour synthétiser l'unité de communication.

Nous avons présenté un ensemble de résultats obtenu en appliquant notre méthode à des algorithmes des domaines du Traitement du Signal et de l'Image (TDSI) et des Télécommunications. Nous avons montré l'intérêt et la validité de la mise en œuvre et de l'utilisation dans l'outil GAUT du formalisme de spécification des contraintes d'intégration que nous proposons. Ainsi, la première expérience a été réalisée sur un exemple de Transformée de Fourrier Rapide (*FFT)*. Pour les conditions d'expérimentation retenues, l'optimisation du nombre d'opérateurs est en moyenne de 20% et celle des registres de 7%, par rapport à une approche classique. La deuxième expérience a utilisé une Transformée en Cosinus Discrète *(DCT)* pour comparer les résultats, obtenus en appliquant l'approche d'intégration proposée dans ce manuscrit, avec les résultats des méthodes utilisant des adaptateurs. Pour l'exemple considéré, le gain sur les registres de communications varie de -2% à 88% à débit d'E/S constant. La dernière expérience, qui a été réalisée, démontre l'applicabilité de notre méthodologie sur un composant virtuel comportemental complexe (Maximum A Posteriori MAP) dans le contexte d'une application de Turbo décodage temps réel.

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 d'automatisation doit être mené sur la méthode d'analyse et de vérification des contraintes que nous avons présentées. De plus, bien qu'elle puisse être dérivée automatiquement d'un outil de Codesign, la spécification des contraintes requiert, afin d'être réalisée par l'intégrateur, le développement d'une interface graphique intégrée à l'outil *GAUT*. Notons qu'une thèse est actuellement en cours sur l'extraction des contraintes temporelles d'intégration à partir du raffinement des communications dans l'environnement Cofluent/MCSE.

Un deuxième ensemble de travaux visant la synthèse de l'unité de traitement doit être réalisé. Un premier point concerne le module d'allocation. En effet, la technique d'allocation par recherche dichotomique, restant à coder dans l'outil, a permis dans un premier temps, de prendre en compte les contraintes d'E/S dans l'outil *GAUT*. Toutefois, une approche par estimation du nombre d'opérateurs à allouer avant synthèse peut aussi être envisagée pour résoudre le problème de l'allocation avant celui de l'ordonnancement ([DIG96]).

Un deuxième point concerne la synthèse par parties de l'algorithme. Les travaux de recherche sur cet aspect devront définir une méthodologie de partitionnement et un ensemble de transformations formelles pour la sérialisation du *GCG*. Sur un critère de coût, cette

méthodologie choisira, parmi l'ensemble des solutions architecturales possibles, la meilleure solution d'implémentation. Dans ce contexte, des modifications concernant les modules d'allocation, d'ordonnancement et d'assignation du logiciel *GAUT* devront être apportées.

Un troisième ensemble de travaux concerne l'automatisation de la génération des modèles *IPERM* et *IOCG* pour la cosimlulation et la synthèse, et du modèle temporel IP Delay Model pour l'analyse de performances et l'exploration architecturale dans des outils tel que VCC.

Enfin, le développement d'un outil de synthèse de l'unité de communication intégré à *GAUT* devra être réalisé. Ce logiciel concrétisera l'approche de synthèse de l'*UCOM* proposée par A. Baganne [BAG97] a laquelle sera ajoutée la gestion des protocoles associés aux ports d'E/S et plus particulièrement du protocole standard VCI.

Bibliographie Personnelle

CONFÉRENCES INTERNATIONALES

[COU04] P. COUSSY, D. GNAEDIG, A. NAFKHA, A. BAGANNE, E. BOUTILLON, E. MARTIN *"A Methodology for IP Integration into DSP SoC: A Case Study of a MAP Algorithm for Turbo Decoder"*, Dans Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Montreal, Canada, mai 2004

[COU03A] P. COUSSY, A. BAGANNE, E. MARTIN, *"Communication and Timing Constraints Analysis for IP Design and Integration"*, Dans Proc. International Conference on Very Large Scale Integration (IFIP, VLSI-SoC), Darmstadt, Allemagne, décembre 2003

[COU02A] P. COUSSY, A. BAGANNE, E. MARTIN *" Virtual Component IP Re-use in Telecommunication Systems Design:A Case Study of MPEG-2 / JPEG2000 Encoder ",* Dans Proc. IEEE International Conference on Electronics, Circuits, and Systems (ICECS), Dubrovnik, Croatia, septembre 2002

[COU02B] P. COUSSY, A. BAGANNE, E. MARTIN *"IP Cores Integration in DSP System-On-Chip Designs"* Dans Proc. European Signal Processing Conference (EUSIPCO), Toulouse, France, septembre2002

[COU02C] P. COUSSY, A. BAGANNE, E. MARTIN *"Platform-Based Design For Digital Signal Processing Systems : A Case Study of MPEG-2 / JPEG2000 Encoder"*, Dans Proc. IEEE International Conference on Communication Circuits and Systems (ICCCAS), Bejung, China, juin 2002

[COU02D] P. COUSSY, A. BAGANNE, E. MARTIN *"A Design Methodology for IPIntegration",* Dans Proc. IEEE International Symposium on Circuits and Systems (ISCAS), Scottsdale, USA, mai 2002.

[COU02E] P. COUSSY, A. BAGANNE, E. MARTIN *"A Design Methodology for Integrating IP into SOC Systems"* Dans Proc. IEEE International Custom Integrated Circuits Conference (CICC), Orlando, USA, mai 2002

[COU01A] G. SAVATON, P. COUSSY, E. CASSEAU, E. MARTIN *"A Methodology for Behavioral Virtual Component Specification Targeting SoC Design with High-Level Synthesis Tools",* Dans Proc. Forum on Design Languages (FDL), Lyon, France, septembre 2001

[COU01B] S. PILLEMENT, O. SENTIEYS, D. CHILLET, E. CASSEAU, P. COUSSY, E.MARTIN, G. SAVATON, S. ROUX *"Design and synthesis of behavioral level virtual components"*, Dans Proc. International Conference on Very Large Scale Integration (IFIP, VLSI-SoC), Montpellier, France, décembre 2001

[COU04] P. COUSSY, PhD Forum IEEE International Conference on Design Automation and Test in Europe (DATE), Paris, février 2004

CONFÉRENCES NATIONALES

[COU03B] P. COUSSY, A. BAGANNE, E. MARTIN, E. CASSEAU, *"Intégration Optimisée de Composants Virtuels orientés TDSI par la Synthèse d'Architecture"*, Dans Actes colloque GRETSI sur le traitement du signal et des images, Paris, septembre 2003

[COU02F] P. COUSSY, A. BAGANNE, E. MARTIN *"Réutilisation de Composant Virtuel dans les Systèmes de Télécommunication: Une Etude de Cas MPEG-2 / JPEG2000*", Dans Actes Journées Francophones sur l'Adéquation Algorithme/Architecture (JFAAA), Monastir, Tunisie, décembre 2002

[COU02G] P. COUSSY, A. BAGANNE, E. MARTIN *"Analyse Fonctionnelle des Moyens de communication Proposés dans les Systèmes sur Silicium",* Dans Actes Journées Francophones sur l'Adéquation Algorithme/Architecture (JFAAA), Monastir, Tunisie, décembre 2002

Bibliographie

[ADE03] Adelante Technologies. A|RT Designer. http://www.adelantetech.com/

[ALE00] P. ALEXANDER, D. BARTON, R. KARNATH, *"System Specification in Rosetta"*, In Proceedings of IEEE Engineering of Computer Based Systems Symposium (ECBS), 2000

[ARM99] AMBA : *"Specification Revision 2.0"*, ARM Ltd., 1999

[BAG97] A. BAGANNE, *"Méthodologie de synthèse des unités de communication matérielles dans une approche de conception mixte logiciel/matériel (co-design)"*, Thèse de l'université de Rennes I, décembre 1997

[BAG98] A. BAGANNE, J.L PHILIPPE, E. MARTIN, *"A Formal Technique for Hardware Interface design"*, *In. IEEE Trans. On Circuits And Systems*, Vol.45, N5, pp.584-591, 1998.

[BAH74] L. BAHL, J. COCKE, F. JELINEK, J. RAVIV, *"Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate"*, *IEEE Transactions on Information Theory*, pp.284-287. 1974

[BAL97] F. BALARIN, M. CHIODO, 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.

[BAL98] F. BALARIN, E. SENTOVICH, M. CHIODO, P. GIUSTO, H. HSIEH, B. TABBARA, A. JURECSKA, L. LAVAGNO, C. PASSERONE, K. SUZUKI, A. SANGIOVANNI-VINCENTELLI. *"Hardware-Software Co-Design of Embedded Systems: The POLIS Approach", Norwell,* Kluwer Academic Publishers, 1998

[BEN94] T. BEN ISMAIL, M. ABID, A.A. JERRAYA, *"COSMOS: A CoDesign Approach for Communicating Systems",* Proceedings of IEEE International Workshop on Hardware/Software Co-Design (CODES), pp.17-24, 1994

[BEN96] A. BENDER, *"Design of Optimal Loosely Coupled Heterogeneous Multiprocessor System"*, Proc. IEEE International Conference on Design Automation and Test in Europe (DATE), pp 275-281, 1996.

[BEN96] T. BEN ISMAIL, *"Synthèse au niveau système et conception de systèmes mixtes Logiciels/Matériels",* Thèse de l'INPG, janvier 1996

[BER00] G. BERRY, *"The Fundations of Esterel"*, Rapport technique, Ecoles des Mines Paris/INRIA, 2000

[BER92] R. BERGAMASCHI, A KUEHLMANN, S-M. WU, V. VENKATARAMAN, D. REISCHAUER, D NEUMANN, *"A Methodology for Production Use of High Level Synthesis"*, In Proceedings of International Workshop on High Level Synthesis, 1992

[BER93] C. BERROU, A. GLAVIEUX, P. THITIMAJSHIMA, *"Near Shannon Limit Error-Coding and Decoding: Turbo-codes",* Proceedings IEEE International Conference on Communications (ICC), pp. 1064-1070, 1993

[BIL96] G. BILSEN, M. ENGELS, R. LAUWEREINS, J.A PEPERSTRAETE, *"Cyclo-Static Dataflow"*, IEEE Transactions On Signal Processing, Vol 44, No. 2, 1996

[BJA93] E. BJARNASON, E. HEANSLER, M. RUPP, *"Acoustic Echo Control: Advances in Algorithm Techniques",* In Proc. of Workshop of Acoustic Echo Control, 1993.

[BOM03] P. BOMEL, E. MARTIN, E. CASSSEAU, *"Spécification du protocole et de la structure de l'interface des circuits synthétisés par GAUT"*, Rapport technique du LESTER, version 2.0, juin 2003

[BOO99] GRADY BOOCH, IVAR JACOBSON, JAMES RUMBAUGH, JIM RUMBAUGH, *"The Unified Modeling Language User Guide"*, Addison Wesley Professional, 1999

[BOR87] G. BORIELLO, R.H. KATZ, *"Synthesis and Optimization of Interface Transducer Logic"*. Proc. IEEE International Conference on Computer Aided Design (ICCAD), pp. 274-277, 1987.

[BOU93] E. BOUTILLON, N. DEMASSIEUX, *"A generalized precompiling scheme for surviving path memory management in Viterbi decoders",* Proceedings Proc. IEEE International Symposium on Circuits and Systems (ISCAS),vol. 3, pp. 1579-82, 1993.

[BRU00] J-Y. BRUNEL, W. KRUIJTZER, H. KENTER, F. PETROT, L. PASQUIER, E.A. DE KOCK, W. SMITS, *"COSY Communication IP's"*, Proceedings IEEE International Design Automation Conférence (DAC), pp. 406-410, 2000

[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), pp. 142-146, 1999

[CAD03A] CADENCE, Signal Processing Worksystem SPW
http://www.cadence.com/products/incisive_spw.html

[CAD03B] CADENCE VIRTUAL COMPONENT CO-DESIGN VCC, 2003
http://www.cadence.com/datasheets/vcc.html

[CAI03] L. CAI, D. GAJSKI, *"Transaction Level Modeling in System Level Design"*, CECS Technical Report 03-10, 2003

[CAM86] R. CAMPOSANO, A. KUNZMANN, *"Considering Timing Constraints in Synthesis From Behavioral Description"*, Proceedings IEEE International Conference on Computer Design (ICCD), pp 6-9, 1986

[CAM89] R. CAMPOSANO, W. ROSENSTIEL, *"Synthesizing circuits from behavioral descriptions"*, IEEE Transactions on Computer Aided Design (CAD/ICAS), vol. Vol. 8, no. No. 2, pp. 171–180, 1989.

[CAT97] A. CATALDO, *"VSI abandons plans for system-chip bus"*, EETimes, 1997

[CES00] W. CESARIO, L. GAUTHIER, D. LYONNARD, G. NICOLESCU, A.A. JERRAYA, *"An XML-based Meta-model for the Design of Multiprocessor Embedded Systems"*, VHDL International User's Forum (VIUF) Fall Workshop, Orlando, FL, October 2000

[CES02] W. CESARIO, A. BAGHDADI , L. GAUTHIER , D. LYONNARD , G. NICOLESCU , Y. PAVIOT , S. YOO , A. JERRAYA, ET AL. *"Component-based design approach for multicore SoCs",* Proceedings IEEE International Design Automation Conférence (DAC), 2002

[CHA66] R. CHANG, J. HANCOCK, *"On Receiver Structures for Channels Having Memory"*, *IEEE Transactions on Information Theory*, Vol. IT-12, No. 4. Pp. 463-468, 1966.

[CHA99] H. CHANG, L. COOKE, ET AL., *"Surviving the SOC revolution, A guide to Platform-Based Design"*, Kluwer academic publishers, 1999

[CHI93] M. CHIODO, P. GIUSPO, A. JURECSKA *"Synthesis of mixed software-hardware implementations from CFSM specifications"*. Proceedings IEEE International Workshop on Hardware/Software Co-Design (CODES), 1993.

[CHI93] M. CHIODO, P. GIUSTO, H. HSHIEH, A. JURESKA, L. LAVAGNO, A. SANGIOVANNI-VINCENTELLI, *"A formal specification model for hardware / software codesign"*, Proceedings IEEE international workshop on hardware / software codesign (CODES), 1993

[CHI97A] D. CHILLET,*"Méthodologie de conception architecturale des mémoires pour circuits dédiés au traitement du signal temps réel",* Thèse de l'université de Rennes I, janvier 1997

[CHI97B] D. CHILLET, J.P. DIGUET, J.L. PHILIPPE, O. SENTIEYS, *"Méthodologie de conception des unités mémoires appliquée au traitement du signal temps réel"*, Revue Technique et Science Informatique TSI, vol. 16, 1997

[CHO95] P. CHOU, R. ORTEGA, G. BORRIELLO, *"Interface Co-Synthesis Techniques for Embedded Systems"*. Proceedings IEEE International Conference on Computer Aided Design (ICCAD), pp. 280-287, 1995

[CHO99] P. CHOU, R. ORTEGA, K. HINES, K. PARTRIDGE, G. BORRIELLO, *"IPChinook: an integrated IP-based design framework for distributed embedded systems",* Proceedings IEEE International Design Automation Conférence (DAC), 1999.

[COF03] COFLUENTSTUDIO, http://www.cofluentdesign.com/

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

[COR00] J. CORDATELLA, A. KONDRATYEV, L. LAVAGNO, AL., *"Task generation and compile-time scheduling for mixed data-control embedded software"*, Proceedings IEEE International Design Automation Conférence (DAC), 2000

[COR03A] G. CORRE, N. JULIEN, E. SENN, ERIC MARTIN, *"Contraintes mémoire et solution architecturale pour applications TDSI"*, Actes du colloque GRETSI, 2003.

[COR03B] G. CORRE, N. JULIEN, E. SENN, E. MARTIN, *"Ordonnancement sous contraintes de mémorisation : une optimisation efficace des ressources lors de la synthèse d'architecture",* Actes des Journées Francophones d'études Faible Tension Faible Consommation (FTFC), 2003.

[COR99] B. CORDAN, *"An efficient Bus architecture for System-on-Chip Design"*, Proceedings IEEE International Custom Integrated Circuits Conference (CICC)*, 1999.*

[COW03] Coware, Napkin-to-Chip N2C, http://www.coware.com

[CUE01] F. CUESTA, *" Synthèse des ressources de communication pour la conception de systèmes embarqués temps réel flots de données ",* Thèse de l'Université de Nice, octobre 2001

[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

[DAB97] ETSI ETS 300 401, "RADIO BROADCASTING SYSTEMS; DIGITAL AUDIO BROADCASTING (DAB) TO MOBILE, PORTABLE AND FIXED RECEIVERS", MAY 1997

[DAV01] JM. DAVEAU, G. FERNANDES MARCHIORO, T. BEN-ISMAIL, A. JERRAYA, *"Protocol Selection and Interface generation"*, Proceedings IEEE international workshop on hardware / software codesign (CODES), 2001

[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 modeling for signal processing systems"*, Proceedings IEEE International Design Automation Conférence (DAC), pp. 402–405, 2000

[DeM88] G. DE MICHELI, D. KU, *"Hercules, a System for High-Level Synthesis"*, Proceedings IEEE International Design Automation Conférence (DAC), pp. 483-488, 1988

[DeM90] G. DE MICHELI, D. KU, F. MAILHOT, T. TRUONG, *"The Olympus Synthesis System for Digital Design"*, IEEE Design and Test Magazine, pp. 37-53, 1990.

[DEW85] P DEWILDE, E. DEPRETTERE, R. NOUTA, *"Parallel and Pipelined VLSI Implementation of Signal Processing Algorithms"*, In S. Kung, H. Whitehouse and T. Kailath (ed), VLSI and Modern Signal Processing, pp 257-264, Prentice-Hall, 1985

[DEY97] S. DEY, S. BOMMU, "Performance Analysis of a System of Communicating Processes", Proceedings IEEE International Conference on Computer Aided Design (ICCAD), 1997

[DIG96] J.P. DIGUET, *"Estimation de Complexité et Transformations d'Algorithmes de Traitement du Signal pour la Conception de Circuits VLSI",* Thèse de l'Université de Rennes I, octobre 1996

[DIN99] A. DINGNINOU, F. RAFAOUI, C. BERROU, *"Organisation de la mémoire dans un turbo décodeur utilisant l'algorithme SUB-MAP",* Actes du colloque GRETSI, pages 71-74, 1999.

[DVB00] DVB-RCS STANDARD, "*Interaction channel for satellite distribution systems*", ETSI EN 301 790, V1.2.2, PP. 21-24, DEC 2000

[DVB96] ETSI ETS 300 744, "*Digital Video Broadcasting (DVB), Framing structure, channel coding and modulation for digital terrestrial television (DVB-T)"*, NOV. 1996.

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

[ELL00] J. P. ELLIOTT, *"Understanding Behavioral Synthesis. A Practical Guide to High-Level Design",* Kluwer Academic Publishers, 2000.

[FER99] A. FERRARI, A. SANGIOVANNI-VINCENTELLI, *"System Design : Traditional Concepts and New Paradigms"*, Proceedings IEEE International Conference on Computer Design (ICCD), 1999

[FIL93] D. FILO, D. C. KU, C. N. COELHO, G. DE MICHELI, *"Interface optimization for concurent systems under timing constraints"*. IEEE Trans on VLSI systems, pp. 268-281, 1993.

[FRE96] L. FREUND, M. ISRAËL, F. ROUSSEAU, J.M. BERGE, M. AUGUIN, C. BELLEUDY, G. GOGNIAT, *"A codesign experiment in acoustic echo cancellation: Gmdf α"*. Proceedings IEEE International Symposium on System Synthesis (ISSS), 1996.

[G2C03] Get2Chip. Architectural Compiler (G2C-AC). http://www.get2chip.com/index/products/g2c-ac_ds.asp.

[GAI03] R. GAIECH, A BAGANNE, E. MARTIN, M. ABID, *"A Multi-level co-simulation Approach for virtuel component based design"*, IEEE International Conference on Signals Systems Decision and Information Technology (SSD'03), 2003.

[GAJ00] D. GAJSKI, AL., *" SpecC : Specification Language and Methodology"*, Kluwer Academic Publishers, 2000

[GAJ91] D. GAJSKI et al. *"High-Level Synthesis : Introduction to Chip and System Design",* Kluwer Academic Publishers, 1991

[GAJ98] D. GAJSKI, R. DÖMER, J. ZHU, *"IP-centric Methodology and design with the SpecC Language",* NATO-ASI Workshop on System Level Synthesis for Electronic Design, 1998.

[GAR03] GARTNER DATAQUEST, Site Internet, http://www4.gartner.com/Init

[GNA03] D. GNAEDIG, E. BOUTILLON, E. MARTIN, ET AL, *"Synthèse d'architecture pour la réalisation comportementale de l'algorithme MAP pour Turbo Décodeur", Anales des Télécommunications, 2003*

[GOG97] G. GOGNIAT, *"Architecture générique et synthèse des communications pour la conception conjointe de systèmes embarqués logiciel/matériel"*. Thèse de l'Université de Nice, novembre 1997.

[GON96] J. GONG, D. GAJSKI, S. BAKSHI, *"Model refinement for hardware/software codesign",* Proceedings IEEE International Conference on Design Automation and Test in Europe (DATE), 1996.

[GRA91] W. GRASS, *"A Branch-and-Bound Method for Otpimal Transformation of Data Flow Graphs for Observing Hardware Constraints",* Proceedings IEEE International European Design Automation Conference (EDAC), pp. 73-77, 1991

[GUE00] P. GUERRIER, A. GREINER, *"A Generic Architecture for On-Chip-Packet-Switched Interconnections"*, Proceedings IEEE International Conference on Design Automation and Test in Europe (DATE), pp. 250-256, 2000

[GUP91] R. GUPTA, G. DE MICHELI, *"VULCAN - A System for High-Level Partitioning of Synchronous Digital Circuits",* Technical Report CSL-TR-91-471, April 1991

[GUP95] R. GUPTA, *"Co-Synthesis of Hardware and Software for Digital Embedded Systems",* Kluwer academic publishers, 1995

[HAL91] N. HALBWACHS, P. CASPI, P. RAYMOND, D. PILAUD, *"The Synchronous dataflow programming language LUSTRE"*, In Proceedings of IEEE, vol. 79, N°9, pp. 1305-1320, 1991

[HAR87] D. HAREL *"Statecharts : A visual formalism for complex systems"*, Science of Computer Programming 8, pp. 231-274, 1987.

[HEN94] J. HENKEL, R. ERNST, U. HOLTMAN, T. BENNER. *"Adaptation of partitioning and high level synthesis in hardware/software co-synthesis"*. Proceedings IEEE International Conference on Computer Aided Design (ICCAD), pp. 96-100, 1994

[HOA85] C. HOARE, *"Communicating Sequential Processes"*, Prentice Hall, 1985

[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), pp. 48-53, 2001

[IBM03] IBM, http://www-3.ibm.com/chips/products/coreconnect/

[ICL99] INTERNATIONAL COMPUTERS LTD, *"VHDL+ Language Reference Manual Extensions to VHDL for system specification"*, version 5.0, 1999

[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

[ITR01] ITRS, International Technology Roadmap for Semiconductors, Rapport Technique, 2001

[ITU99] ITU-T Recommendation Z-100*, " Specification and description language (SDL)",* ITU-T Z-Series Recommendations, *1999*

[JER97] A. A. JERRAYA, H. DING, P. KISSION, M. RAHMOUNI, *"Behavioral Synthesis and Component Reuse with VHDL",* Kluwer Academic Publishers. 1997

[KAH74] G. KAHN, *"The semantics of a simple language for parallel programming",* In *Information Processing*, pp. 471–475, 1974.

[KAR02] F. KARIM, A. NGUYEN, S. DEY, *"An Interconnect Architecture for Networking Systems on Chips"*, IEEE Micro, pp. 36-45, 2002

[KEA99] M. KEATING, P. BRICAUD, *"Reuse Methodology Manual for System-on-a-Chip Design",* Kluwer Academic Publishers, 1999

[KEN99] H. KENTER, C. PASSERONE, W.J.M. SMITS, Y. WATANABE, A. SANGIOVANNI-VINCENTELLI, *" Designing Digital Video Systems: Modeling and Scheduling ",* Proceedings IEEE International Workshop on Hardware/Software Co-Design (CODES), pp. 64-68, 1999

[KEU00] K. KEUTZER, S. MALIK, A. NEWTON, J. RABAEY AND A. SANGIOVANNI-VINCENTELLI *"System-Level Design: Orthogonalization of Concerns and Platform-Based Design", IEEE Transaction of Integrated Circuits and Systems, Vol.19, N°12, December 2000*

[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, pp.338-349, 1997.

[KNA95] D. KNAPP, T. LYAND and AL. *"Behavioral Synthesis Methodology for HDL-Based Specification and Validation"*, Proceedings IEEE International Design Automation Conférence (DAC), 1995

[KNA96] D. KNAPP, *"Behavioral Synthesis. Digital System Design Using the Synopsys Behavioral Compiler"*, Prentice Hall, 1996.

[KNU99] P.V KNUDSEN, J. MADSEN, *"Integrating Communication Protocol Selection with Hardware/Software Codesign"*, IEEE Transactions on Computer Aided Design / Integrated circuits and systems (CAD/ICAS), vol. 18, N°8, 1999

[KU90] D. KU AND G. DE MICHELI, *"HardwareC - A Language for Hardware Design*

(version 2.0)" CSL Technical Report CSL-TR-90-419, Stanford University, 1990

[KU91A] D. KU, G. DE MICHELI, *"Optimal Synthesis of Control Logic from Behavioral Specifications"*, Integration: the VLSI Journal, Vol. 10, No. 3, pp. 271-298, 1991

[KU91B] D. KU, G. DE MICHELI, *"Constrained Resource Sharing and Conflict Resolution in Hebe"*, Integration: the VLSI Journal. Vol. 12, No. 2, pp. 131-166, 1991

[KU91C] D. KU, D. FILO, G. DE MICHELI, *"Optimizing Control by Resychronization of Operations"*, Proceedings IEEE International Design Automation Conférence (DAC), pp. 366-371, 1991

[KU92] D. KU AND G. DE MICHELI, *"Relative Scheduling Under Timing Constraints: Algorithms For High-Level Synthesis of Digital Circuits"*, IEEE Transactions on Computer Aided Design, Vol.11, N°6, 1992

[LAH01] K. LAHIRI, A. RAGHUNATHAN, S. DEY, "System-Level Analisys for Designing On-Chip Communication Architectures", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (CAD/ICAS), Vol. 20, N°. 6, June 2001

[LAV99] L. LAVAGNO, E. SENTOVICH, "ECL : A Spefication Environment For System-Level Design", Proceedings IEEE International Design Automation Conférence (DAC), 1999.

[LEE87] E.A LEE, D.G MESSERSCHMITT, *"Synchronous Dataflow"*, Proceedings of IEEE, 1987

[LEG91] P. LE GUERNIC, T. GAUTIER, M. LE BORGNE, C. LE MAIRE, *"Programming Real-Time Applications with Signal"*, In Proceedings of the IEEE, vol. 79, N° 9, pp.1321-1336, 1991.

[LEH99] H. LEHR, D. GAJSKI, *"Modeling Custom Hardware in VHDL",* Technical Report ICS-99-29, 1999

[LEN00] C. K. LENNARD, P. SCHAUMONT, G. DE JONG, A. HAVERINEN, P. HARDEE, *"Standards for system-level design: practical reality or solution in search of a question?"*, Proceedings IEEE International Conference on Design Automation and Test in Europe (DATE), pp. 576–585, 2000.

[LES03] Http://www.lester.univ-ubs.fr:8080

[LIA83] Y; LIAO, C. WONG, *"An algorithm to compact VLSI symbolic layout with mixed constraints",* IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (CAD/ICAS), vol. 2, pp62-69, 1983

[LIN94] B. LIN, S. VERCAUTEREN, *"Synthesis of concurrent system interface modules with automatic protocol conversion generation".* Proceedings IEEE International Conference on Computer Aided Design (ICCAD), 1994.

[LY95] T. Ly, D. Knapp, R. Miller, D. MacMillen, *"Scheduling using Behavioral Templates",* Proceedings IEEE International Design Automation Conférence (DAC)*, 1995*

[LYO01] D. Lyonnard, S. Yoo, A. Baghdadi, A. Jarraya, "Automatic Generation of Application-Specific Architectures for Heterogeneous Multiprocesor System-on-Chip", Proceedings IEEE International Design Automation Conférence (DAC), 2001

[LYS00] R. L. LYSESCKY, F. VAHID, T. D. GIVARGIS, *"Techniques for reducing Read*

Latency of Core Bus Wrapper ", Proceedings IEEE International Conference on Design Automation and Test in Europe (DATE), 2000

[MAD95] J. MADSEN, B. HALD, *"An approach to interface synthesis"*, Proceedings IEEE International Symposium on System Synthesis (ISSS), pp16-21, 1995.

[MAR01] E. MARTIN, C. NOUET, J.M. TOUREILLES, *"Conception optimisée d'architectures en précision finie pour les applications de traitement du signal"*, Revue Traitement du Signal, Volume 18 numéro 1, 2001

[MAR92] E. MARTIN, J.L. PHILIPPE, *"Noyau de l'outil de synthèse GAUT",* Rapport technique B, LASTI / ENSSAT, Novembre 1992

[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

[MAT03] MATHWORKS 2000, Matlab-Simulink-StateFlow, http://www.mathworks.com

[MCS03] Modélisation et Conception des Systèmes Electroniques, http://mcse.ireste.fr/

[MEL99] A. MELIKIAN, D. ALTAS, G. FAYAD, *"A High Level Synthesis Approach to Soft IP Reuse",* In Proceedings of Synopsys Users Group Forum (SNUG), 1999

[MOT99] MOTOROLA, *"IP Interface Standart document"*, version 2.0, 1999

[NAR94] S. Narayan, D. Gajski, *"Synthesis of System-level Bus Interface"*, Proceedings IEEE International European Design Automation Conference (EDAC), pp 395-399, 1994.

[NAR95] S. NARAYAN, D. GAJSKI, *"Interfacing incompatible protocols using interface process generation"*,. In Proceeding of DAC, pp 468-473, 1995.

[NES86] J. NESTOR, D. THOMAS, *"Behavioral Synthesis with Interfaces"*, Proceedings IEEE International Design Automation Conférence (DAC), pp112-115, 1986.

[NES93] J. NESTOR, G. KRISHNAMOORTHY, *"SALSA: A New Approach to Scheduling with timing constraints",* IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (CAD/ICAS), vol.12, no 8, pp 1107-22, 1993

[NIC02] G. NICOLESCU, S. YOO, A. BOUCHHIMA, A.A. JERRAYA, *"Validation in a Component-Based Design Flow for Multicore SoCs"*, Proceedings IEEE International Symposium on System Synthesis (ISSS), 2002.

[OCB00] ON CHIP BUS DEVELOPMENT WORKING GROUP, VSI ALLIANCE, *"VIRTUEL COMPONENT INTERFACE STANDARD (OCB 2 2.0)"*, 2000

[OMI96] OPEN MICROPROCESSOR SYSTEMS INITIATIVE, *"OMI 324 : Pi-Bus"*, April-May 1996

[OSC01] OPEN SYSTEMC INITIATIVE (OSCI),*"SystemC Version 2.0 User's Guide", Technical Report, 2001*

[PAL01] PALMCHIP CORP., "CoreFrame view white paper: Overview of the CoreFrame Architecture", 2001

[PAR93] I. PARK, K. O'BRIEN, A. JERRAYA, *"AMICAL: Architectural Synthesis Based on VHDL"*, In "Synthesis for control dominated circuits", Ed. G. Saucier, Publ. ELSEVIER, 1993.

[PAS98] R. PASSERONE, J. A. ROWSON, *"Automatic Synthesis of Interfaces between Incompatible Protocols"*, Proceedings IEEE International Design Automation Conference (DAC), 1998

[PAU89] P. G. PAULIN, J. P. KNIGHT, *"Algorithms for High-Level Synthesis"*, IEEE Design

and Test Computers, pp.18-31, 1989

[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), pp. 51-56, 2003

[PHI95] J.L PHILIPPE, O.SENTIEYS, J.P DIGUET, E. MARTIN, *"From Digital Signal Processing Specification to Layout"*, In Proceedings of Logic and Architecture Synthesis : state-of-the-art and novel approaches, pp. 307-313, 1995

[PIL01] S. PILLEMENT, O. SENTIEYS, D. CHILLET, E. CASSEAU, P. COUSSY, E.MARTIN, G. SAVATON, S. ROUX *"Design and synthesis of behavioral level virtual components"*, Proceedings International Conference on Very Large Scale Integration (IFIP, VLSI-SoC), 2001

[RNR02] RNRT, LESTER, LASTI, ET FRANCE TELECOM R&D. Site Internet du Projet MILPAT. http://lester.univ-ubs.fr/milpat

[RNR03] RNRT, LESTER, TNI-VALIOSYS, THALES, SACET, TURBOCONCEPT, ENST BRETAGNE. Site Internet du Projet ALIPTA. http://lester.univ-ubs.fr/alipta

[ROW97] J. ROWSON AND A. SANGIOVANNI-VINCENTELLI, *"Interface-Based Design",* Proceedings IEEE International Design Automation Conference (DAC), pp.178-183, 1997

[SAN00] A. SANGIOVANNI-VINCENTELLI, M. SGROI, L. LAVAGNO, *"Formal Models for Communication-based Design ",* Proceedings International Conference on Concurrency Theory (CONCUR), 2000

[SAV01] G. SAVATON, P. COUSSY, E. CASSEAU, E. MARTIN *"A Methodology for Behavioral Virtual Component Specification Targeting SoC Design with High-Level Synthesis Tools",* Proceedings Forum on Design Languages (FDL), 2001.

[SAV02] G. SAVATON, *"Méthodologie de Conception de Composants Virtuels Comportementaux pour une Chaîne de Traitement du Signal Embarquée"*, Thèse de l'Université de Bretagne Sud, décembre 2002

[SEN96A] O. SENTIEYS, J.P. DIGUET, J.L. PHILIPPE, E. MARTIN, *"Hardware module selection for real time pipeline architectures using probabilistic cost estimation"*, Proceedings IEEE International ASIC Conference, 1996

[SEN96B] O. SENTIEYS, D. CHILLET, J.P. DIGUET, J.L. PHILIPPE, *"Memory module selection for high level synthesis",* IEEE Press, VLSI Signal Processing, 1996

[SEP92] J. SEPTIEN, D. MOZOS, F. TIRADO, ET AL. *"Heuristic for Branch-and-Bound Global Allocation"*, Proceedings IEEE International European Design Automation Conference (EURO-DAC), pp. 334-340, 1992,

[SGR00] M. SGROI, L. LAVAGNO, A. SANGIOVANNI-VINCENTELLI, *"Formal Models for Embedded System Design"*, IEEE Design & Test of Computers, pp. 2-15, 2000

[SHI02A] D. SHIN, D. GAJSKI, *"Interface Synthesis from Protocol Specification",* Technical Report CECS-02-13, 2002

[SHI02B] D. SHIN, D. GAJSKI, *"Queue Generation Algorithm for Interface Synthesis ",* Technical Report CECS-02-12, 2002

[SIL01] SILICORE CORP , *"Wishbone System-On-Chip Interconnection Architecture for Portable IP Core"*, Rev. B.1, 2001

[SON00A] SONICS INC, *"Sonics µnetworks Technical Overview"*, June 2000

[SON00b] Sonics Inc, *"Open Core Protocol Specification 1.0"*, 2000

[STO92] A. STOLL ,P. DUZY, *"High-Level Synthesis from VHDL with Exact Timing Constraints",* Proceedings IEEE International Design Automation Conference (DAC), 1992

[SUT99] S. SUTHERLAND*, "The Verilog PLI Handbook :, A User's Guide and Comprehensive Reference on the Verilog Programming Language Interface"*, Kluwer Academic Publishers, 1999

[SYN97] SYNOPSYS *"COSSAP Reference Manual"*, Reference Manual, 1997

[VER00] F. VERMEULEN, F. CATTHOOR, D. VERKEST, H. DE MAN, *"Formalized Three-Layer System-Level Model and Reuse Methodology for Embedded Data-Dominated Applications"*, IEEE Trans. on VLSI Systems, Vol.8, No.2, pp.207-216, 2000

[VER02] F. VERMEULEN, *"Reuse of System-Level Design Comonents in Data-Dominated Digital System",* Thèse de l'université Catholique de Louvain, Decembre 2002

[VSI00] VSI ALLIANCE. Deliverables Document _ Version 2.3. Rapport technique, 2000.

[VSI03] VSI ALLIANCE. Site Internet. http://www.vsi.org.

[VSI97] VSI ALLIANCE. Architecture Document _Version 1.0. Rapport technique, 1997.

[WOL92] W. WOLF, S. TAKACH, C.-Y. HUANG, R. MANNO, E. WU, *"The Princeton University Behavioral Synthesis System"*, Proceedings IEEE International Design Automation Conference (DAC), pp. 182-187, 1992

[YEN95] T. YEN, W. WOLF, *"Communication Synthesis for Distributed Embedded Systems"*, Proceedings IEEE International Conference on Computer Aided Design (ICCAD), 1995

[YEN96] T. YEN, W. WOLF, *"Hardware-Software Co-Synthesis of Distributed Embedded Systems"*, Kluwer Academic Publishers. 1996.

[YOO01] S. YOO, G. NICOLESCU, D. LYONNARD, A. BAGHDADI, AND A. A. JERRAYA, *"A generic wrapper architecture for multi-processor soc cosimulation and design"*, Proceedings IEEE International Workshop on Hardware/Software Co-Design (CODES), pp. 195–200, 2001.

[ZHU97] J. ZHU, D. D. GAJSKI, AND R. DOEMER, *"Syntax and semantics of the spec C+ language"*. Proceedings Workshop on Synthesis And Simulation Meeting and International interchange (SASIMI), pp. 75–82, 1997

Annexe