Synthèse architecturale flexible

W. Cesario

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

|__/__/__/__/__/__/__/__/__/__/

T H E S E

pour obtenir le grade de

DOCTEUR DE l'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Discipline : Microélectronique

présentée et soutenue publiquement

par

Wander Oliveira CESÁRIO

le 14 octobre 1999

Titre :

Synthèse architecturale flexible

Directeur de thèse :

M. Ahmed Amine JERRAYA

JURY

M.	Bernard COURTOIS	Président
M.	Habib MEHREZ	Rapporteur
M.	Marius STRUM	Rapporteur
Mlle.	Anne MIGNOTTE	Examinateur
M.	Ahmed A. JERRAYA	Examinateur

A mes parents,
à Shirlei, et
à Stéphanie.

Remerciements

Je voudrais tout d'abord remercier mon directeur de thèse Monsieur Ahmed Amine Jerraya pour ses précieux conseils et surtout pour son soutien tout au long de ma recherche. Je tiens également à remercier Monsieur Bernard Courtois pour m'avoir accueilli au Laboratoire TIMA et pour m'avoir fait l'honneur de présider ce jury. Je voudrais exprimer ma gratitude à Monsieur Habib Mehrez et Monsieur Marius Strum pour avoir accepté de rapporter sur cette thèse. Pour la participation au jury, je remercie Mademoiselle Anne Mignotte.

Je remercie le gouvernement brésilien, particulièrement la CAPES (*Coordenação de Aperfeiçoamento de Pessoal de Nível Superior*) pour avoir financé ma recherche. Ainsi que le COFECUB (*Comité Français d'Evaluation de la Coopération Universitaire avec le Brésil*) pour son aide financière pour l'achat des supports techniques.

J'espère que le lecteur appréciera cette thèse, qu'il apprendra beaucoup à sa lecture et qu'il l'utilisera.

Sommaire Condensé

Résumé

Le sujet de cette thèse est le développement d'une nouvelle méthodologie de synthèse basée sur une approche interactive et flexible conçue pour l'exploration de l'espace des solutions. C'est le concepteur qui est au centre du processus de création, il a la possibilité d'adapter les techniques et les méthodes de conception à l'application et il est guidé par des estimations de haute fidélité pour prendre des décisions pendant la phase de synthèse. La flexibilité concerne l'architecture cible et le flot de conception.

La flexibilité de l'architecture cible offre la possibilité de choisir le style des interconnexions selon le facteur de partage des ressources de chaque destination de données. De nouvelles techniques pour l'allocation et l'affectation des interconnexions ont été employées. Elles utilisent des directives d'optimisation visant à réduire le nombre de cellules.

L'existence d'un chevauchement de fonctionnalité entre la synthèse comportementale et la synthèse RTL a permis de nouvelles formes d'intégration entre les deux étapes. Un flot de synthèse qui profite de ce chevauchement est présenté. Les concepteurs peuvent ainsi éviter des étapes de synthèse inutiles ou non-optimales en fonction de l'application et les critères d'optimisation.

Une nouvelle méthodologie pour l'évaluation de performance à partir d'une description de niveau système est aussi présentée. Cette méthodologie est basée sur un modèle de performance exécutable décrit dans un langage de spécification au niveau système. Le modèle tient compte du partitionnement logiciel/matériel, des affectations des multiprocesseurs et du choix des protocoles de communication. Les résultats préliminaires montrent que ce modèle peut atteindre une bonne précision et une bonne fidélité. Toutes ces méthodologies et techniques de conception ont été développés dans le cadre de l'environnement de conception conjointe appelé MUSIC.

Mots-clés : Synthèse de haut niveau et synthèse au niveau système, estimation de performance, exploration de l'espace des solutions, outils et modèles de conception, modèles exécutables pour la spécification de systèmes.

Abstract

This thesis develops a new synthesis methodology based on an interactive and flexible approach to design space exploration. The foundations of this methodology are: the creative process is done by the designer, design techniques and models can be adapted to the application under synthesis and high fidelity estimations guide the decision process during synthesis. This flexibility concerns the target architecture and the design flow.

Firstly, the flexibility of the target architecture is presented: it is the possibility of choosing interconnection schemes according to the resource-sharing factor of each data destination. New design techniques for interconnect allocation/binding with cell minimizing optimization directives were used.

New forms of integrating behavioral and RTL synthesis were made possible by the presence of a functionality overlap in some synthesis environments. A synthesis flow that profits from this overlap is presented. Designers can avoid useless or non-optimal synthesis steps according to the application and the optimization priorities.

A new methodology for performance estimation at the system level of abstraction is introduced. This methodology is based on an executable performance model described in a system-level specification language. The model takes into account hardware/software partitioning, multi-processor bindings and the selection of communication protocols. Preliminary results show that this model can achieve good precision and fidelity. All these design methodologies and techniques were implemented in the co-design environment called MUSIC.

Keywords: High-level and system-level synthesis, performance estimation, design space exploration, design models and tools, executable system-specification models.

Sommaire

Liste des Figures

Liste des Tableaux

Liste des Equations

Chapitre 1

Introduction

It always bothers me that, according to the laws as we understand them today, it takes a computing machine an infinite amount of logical operations to figure out what goes on in no matter how tiny a region of space and no matter how tiny a region of time. How can all that be going on in that tiny space? Why should it take an infinite amount of logic to figure out what one tiny piece of time/space is going to do?

R. P. Feynman, The Character of Physical Law

Ce chapitre présente les motivations, les objectifs et les contributions de ce travail de recherche au domaine de la synthèse des circuits numériques embarqués.

La communauté des concepteurs a été toujours réticente à l'introduction de nouvelles méthodologies et d'outils de conception. La synthèse logique a été développée dans les années 70 et adoptée par l'industrie vers les années 80. La synthèse au niveau transferts de registres (RTL, pour *register transfer level*) a été développée dans les années 80 et aujourd'hui elle est largement utilisée. Bien que la recherche sur la synthèse comportementale ait maintenant une vingtaine d'années [Park79], elle n'est toujours pas complètement appliquée au niveau industriel [SIA97]. Il est vrai qui la synthèse comportementale a connu des succès dans des domaines d'application très restreints [LMWV91][RaMan87][WalC91]. Les méthodologies de conception conjointe (*co-design*) et de conception au niveau système commencent seulement à être disponibles sur le marché. Il est important de souligner que les équipes de conception sont tentées de continuer d'utiliser les outils « testés et qui ont fait leurs preuves » et acceptent difficilement les nouveaux outils.

Le sujet de cette thèse est le développement d'une nouvelle méthodologie de conception appelé *synthèse architecturale flexible* [WOC99b]. Elle est basée sur une approche interactive et flexible d'exploration de l'espace des solutions. Cette méthodologie met le concepteur au centre du processus de création, il a la possibilité d'adapter les techniques et les méthodes de conception à l'application et il peut se baser sur des estimations fidèles pour prendre des décisions pendant la phase de synthèse. La flexibilité concerne l'architecture cible et le flot de conception.

L'utilisation de la méthodologie de synthèse architecturale flexible apporte une amélioration considérable aux environnements de conception actuels. D'abord, travailler avec des objets de haut niveau augmente la productivité des concepteurs et réduit le fossé de productivité. En second lieu, de nouveaux algorithmes ont été développés pour un modèle d'architecture cible flexible. Ces algorithmes tiennent compte de nouvelles contraintes physiques. Finalement, le flot comportemental de synthèse flexible et l'interface flexible avec la synthèse RTL facilite l'intégration de nouveaux outils dans les environnements de conception existants.

1.1 Contribution

Nous croyons fermement que le concepteur doit être le centre du processus de création pendant la synthèse de circuits intégrés. Les outils d'aide à la conception doivent être utilisés

pour faciliter la mise en œuvre des idées du concepteur. Dans cette optique, cette thèse présente une méthodologie interactive et flexible de synthèse qui remet aux concepteurs le pouvoir de décision pendant le processus de synthèse. Les outils fournissent au concepteur des évaluations de performance pour l'aider dans le processus de décision. Cette méthodologie a été appliquée à l'environnement de conception de systèmes appelé **MUSIC**. La contribution spécifique de ce travail de thèse peut être résumée en trois points :

1. La définition des différents algorithmes d'allocation et d'affectation des interconnexions pour une architecture cible flexible ;
2. La mise en œuvre d'un flot flexible de synthèse comportementale et d'une interface flexible avec la synthèse RTL ;
3. La création d'une nouvelle méthodologie d'estimation de performance au niveau système pour les architectures multiprocesseurs.

1.2 Plan de la thèse

Ce travail de thèse se compose des sept chapitres suivants :

- **Chapitre 1 – Introduction** : Ce chapitre présente les motivations, les objectifs et les contributions de ce travail de recherche au domaine de la synthèse des circuits numériques embarqués.
- **Chapitre 2 – La Synthèse de Haut Niveau** : Ce chapitre présente les principaux concepts, méthodologies et environnements utilisés pour la synthèse comportementale. Il présente un outil de synthèse au niveau système dans lequel ont été insérés les outils de synthèse comportementale développés et le modèle général qui est utilisé pour représenter les circuits.
- **Chapitre 3 – La Flexibilité du Flot de Conception** : Ce chapitre présente une approche interactive et flexible pour l'exploration de l'espace des solutions. La flexibilité de cette approche se manifeste à deux niveaux : au niveau du flot de conception et au niveau de l'interface entre la synthèse comportementale et la synthèse au niveau transferts de registres.
- **Chapitre 4 – Modèle d'Architecture Cible Flexible pour la Synthèse** : Ce chapitre présente une méthode flexible pour l'allocation et l'affectation des interconnexions de la partie opérative du circuit. Cette méthode est basée sur un

modèle de chemin de données générique qui permet différents styles d'interconnexions. Nous proposons un modèle d'estimation de la surface de haute fidélité pour guider le concepteur dans le choix du schéma d'interconnexions au niveau comportemental.

- **Chapitre 5 – Les Outils pour la Synthèse Flexible** : Ce chapitre présente les outils développés pour établir un flot flexible pour la synthèse architecturale présenté dans le troisième chapitre. Trois outils ont été développés : un outil de ré-ordonnancement ; un outil d'allocation et d'affectation des unités fonctionnelles ; et un outil d'allocation et d'affectation des interconnexions et de génération d'architecture.
- **Chapitre 6 – L'Estimation de Performance au Niveau Système** : Ce chapitre présente une méthode pour l'estimation de performance au niveau système. L'estimation de performance joue un rôle fondamental pour l'exploration de l'espace des solutions. C'est l'estimation de performance qui permet de guider l'optimisation du partitionnement du système en parties matérielles et parties logicielles. En outre, elle permet une évaluation rapide de l'affectation des canaux de communication aux différents protocoles de la bibliothèque de canaux.
- **Chapitre 7 – Conclusion** : Ce chapitre présente le bilan de ce travail et quelques perspectives pour les développements futurs.

Chapitre 2

La Synthèse de Haut Niveau

La grande littérature est simplement du langage chargé de sens au plus haut degré possible.

Pound (Ezra Loomis), How to Read, I, 4.

Ce chapitre présente les principaux concepts, méthodologies et environnements utilisés pour la synthèse comportementale. Il présente un outil de synthèse au niveau système dans lequel ont été insérés les outils de synthèse comportementale développés et le modèle général qui est utilisé pour représenter les circuits.

2.1 Introduction

La *synthèse* des circuits intégrés est le processus qui transforme une description de haut niveau en une description à un niveau plus bas, tout en gardant la même fonctionnalité. Ce processus de synthèse est l'application d'une méthodologie par le biais des techniques de conception. Cette *méthodologie de conception*, ou *flot de synthèse*, est la séquence des étapes qui créent un système. Elle définit l'ordre des étapes ainsi que les informations transmises entre les outils pour ainsi accomplir le processus de synthèse. Les *techniques de conception* sont les algorithmes qui exécutent les étapes d'une méthodologie. Diverses techniques peuvent être employées pour accomplir une étape dans la méthodologie de conception.

En allant du niveau d'abstraction le plus haut au niveau le plus bas, divers modèles de circuit sont employés, comme montre le Tableau 1 [JDKR97]. Au niveau système, nous travaillons avec des processus qui échangent des messages. Au niveau algorithmique, chaque processus peut être représenté par un graphe de flot de contrôle et/ou de données (GFC, GFD, GFCD, pour *control/data-flow graph*). Il peut également être représenté par une machine d'états finis (MEF) avec chemin de données (FSMD, pour *finite state machine with datapath*). Au niveau transferts de registres, le circuit est généralement représenté par une architecture composée d'un contrôleur et un chemin de données (FSMD). Le contrôleur peut être représenté par une machine d'états finis comportementale (BFSM, pour *behavioral FSM*) si les actions contiennent des opérations. La synthèse logique utilise des équations booléennes, fréquemment représentées sous la forme de BDD (pour *Binary Decision Diagram*). La transposition technologique utilise une bibliothèque de cellules pour produire un réseau de portes interconnectées. Finalement, la synthèse physique produira le dessin des masques, la représentation définitive du circuit intégré.

Niveaux d'abstraction	Modèle de description du circuit
Système	Processus communicants
Algorithmique	GFC, GFD, GFCD, FSMD
Transfert de registres	FSMD, BFSM, MEF, BDD, équations booléennes
Physique	Réseau de portes interconnectées et dessin des masques

Tableau 1 – Les modèles du circuit à différents niveaux d'abstraction

Les techniques de conception employées par les outils de synthèse de MUSIC au niveau système et comportamental sont présentées respectivement dans les sections 2.2 et 2.3.

2.2 La synthèse au niveau système avec MUSIC

La méthodologie de synthèse au niveau système utilisée par MUSIC peut être divisée en trois phases principales : la modélisation, le découpage et le prototypage (voir Figure 1) [MDJ97].

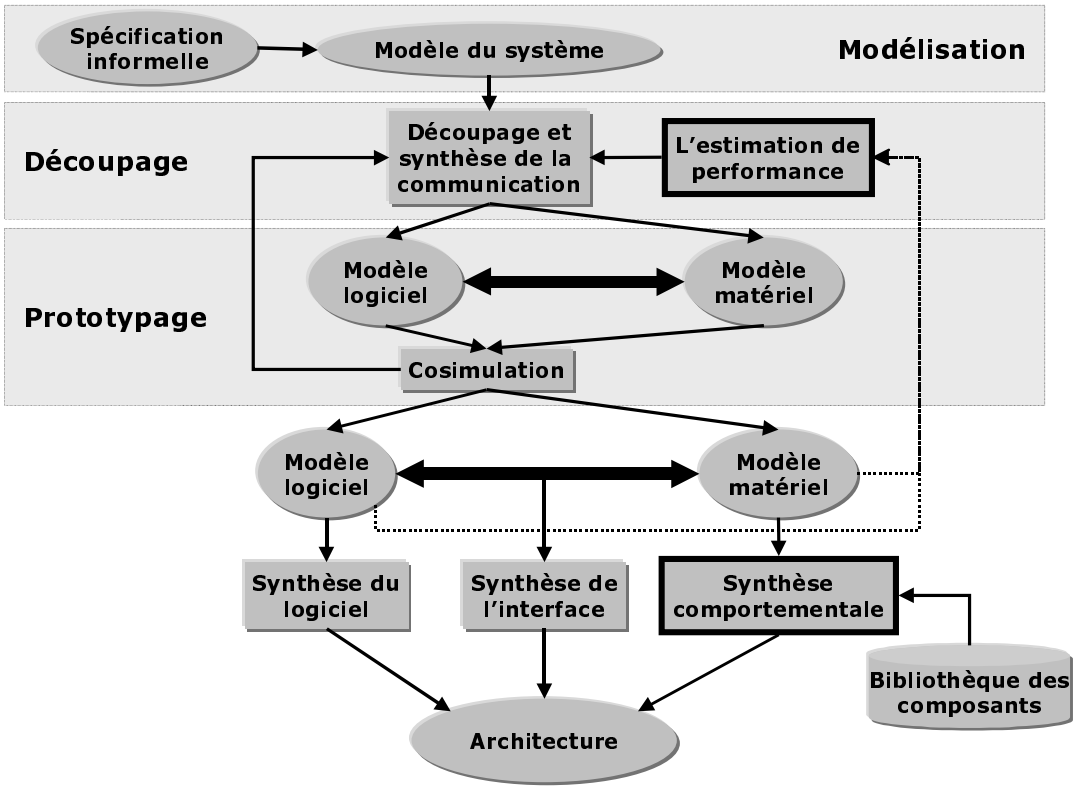


Figure 1 – Les étapes de la conception conjointe

1. La modélisation du système définit la fonctionnalité souhaitée du système et ses restrictions. La spécification peut se baser sur un langage de description au niveau système et souvent elle fournit un modèle exécutable. La validation de cette description utilise des simulateurs ou d'autres techniques de vérification. Le résultat de cette étape est la génération d'une spécification fonctionnelle, libre de tous détails de réalisation ;

2. Le découpage logiciel/matériel et la synthèse de la communication utilisent les informations fournies par l'estimation de performance. Ces informations sont utilisées pour explorer les alternatives de conception et identifier les mieux

adaptées. Cette étape réalise la transposition des fonctions du système sur des composants logiciels et matériels. La synthèse de la communication traduit les primitives de communication à un niveau de détail accepté par les outils de synthèse ;

3. Le prototypage virtuel intègre les techniques de cosimulation et de co-synthèse. La création d'une réalisation pour chaque partie du système est réalisée par les techniques classiques de conception logicielle (compilation, débogage) et matérielle (synthèse comportementale ou RTL). La mise en œuvre de la communication entre les différents modules du système est générée par la synthèse d'interface.

MUSIC applique une méthodologie de conception de systèmes. Les principales caractéristiques de MUSIC sont :

- Le domaine d'application couvre les systèmes distribués et communicants ;
- Le modèle de spécification est traduit dans un format intermédiaire appelé **SOLAR** [JeOB94] qui est utilisé pendant toutes les étapes de raffinement ;
- La flexibilité du processus de synthèse est assurée par l'interaction continue avec l'utilisateur : les transformations sont exécutées automatiquement par les outils et les décisions de conception sont prises par l'utilisateur ;
- MUSIC utilise comme entrée le langage **SDL** [FaeO94] et produit un modèle distribué, en **C** pour les parties logicielles et **VHDL** pour les parties matérielles ;
- Le flot de conception du découpage utilise une approche transformationnelle.

Les contributions de ce travail concernent la phase d'évaluation de performance et dans la phase de synthèse comportementale (voir les boîtes encadrées en gras sur la Figure 1).

2.2.1 Les transformations au niveau système

La Figure 2 présente le flot de conception conjointe. Le processus commence avec la traduction de l'entrée en SDL dans le format SOLAR. Ensuite, l'utilisateur réalise le raffinement du modèle SOLAR. La sortie du système est un prototype virtuel décrit en C et VHDL. MUSIC dispose de trois ensembles de primitives qui modifient le comportement, la structure ou la communication, respectivement : la décomposition fonctionnelle, la réorganisation structurelle et la transformation de la communication [Mar98]. Le flot de

synthèse est flexible, l'utilisateur guide le processus d'interaction en choisissant les transformations adéquates pour obtenir la solution souhaitée.

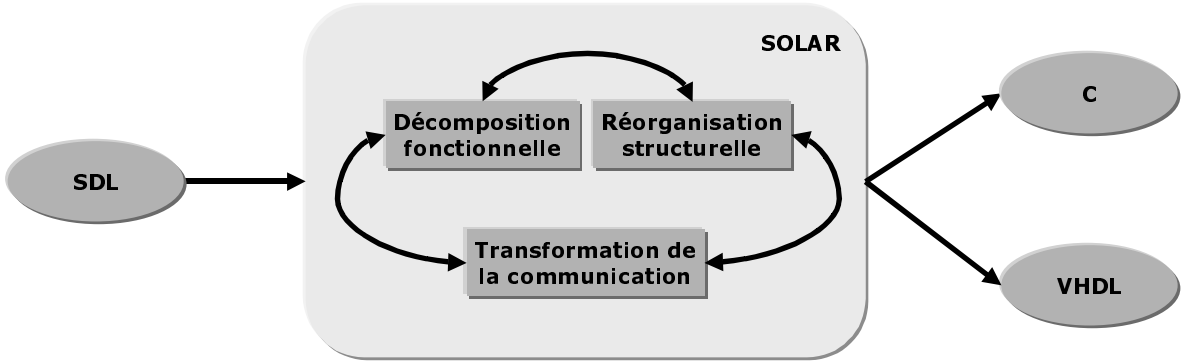


Figure 2 – Le flot de conception au niveau système

La décomposition fonctionnelle permet la décomposition des comportements qui ont besoin d'être exécutés sur des processeurs différents. Cette étape travaille sur des tables d'états et génère éventuellement des éléments de communication additionnels pour que les différentes parties du comportement puissent communiquer (voir Figure 3).

La réorganisation de la structure détermine le nombre et le type de processeurs abstraits et décide de l'affectation des différentes fonctions sur les processeurs alloués. La mise en œuvre des processeurs abstraits peut être réalisée en matériel ou en logiciel. Plusieurs fonctions peuvent être affectées à une simple partition afin de permettre le partage des unités fonctionnelles ou des ressources de communication. Cette étape travaille sur des unités de conception.

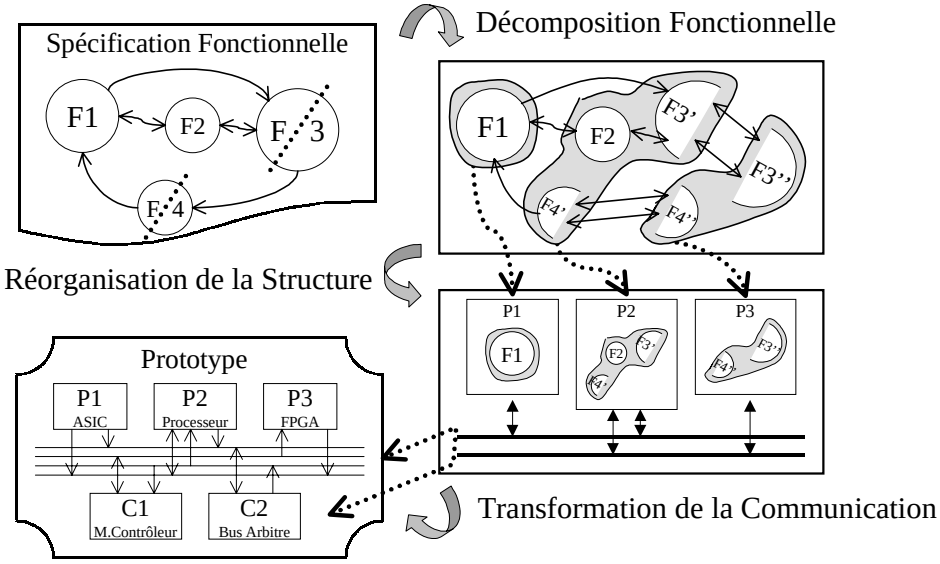


Figure 3 – Etapes du découpage transformationnel

La transformation de la communication permet de raffiner les canaux abstraits utilisés par les processus communicants. Dans ce schéma, les processeurs communiquent à travers des signaux et partagent le contrôle de la communication. Cette étape utilise une bibliothèque de canaux de communication.

L'organisation du découpage en plusieurs étapes augmente la flexibilité du système. L'utilisateur dispose ainsi d'un bon contrôle du processus de conception pour guider l'interaction pendant les transformations incrémentielles. Une méthodologie d'estimation de performance permet d'aider l'utilisateur à prendre des décisions durant la session de synthèse. Cette méthodologie est présentée dans le sixième chapitre.

2.2.2 Le format intermédiaire SOLAR

SOLAR est un modèle de représentation destiné à faire le lien entre les outils de conception de systèmes et les outils de CAO (Conception Assistée par l'Ordinateur) de circuits intégrés. SOLAR est composé d'un modèle de représentation de données et d'un langage textuel. SOLAR permet à la fois la spécification et la synthèse conjointe pour les systèmes contenant du matériel et du logiciel. La Figure 4 représente le contexte d'utilisation de SOLAR.

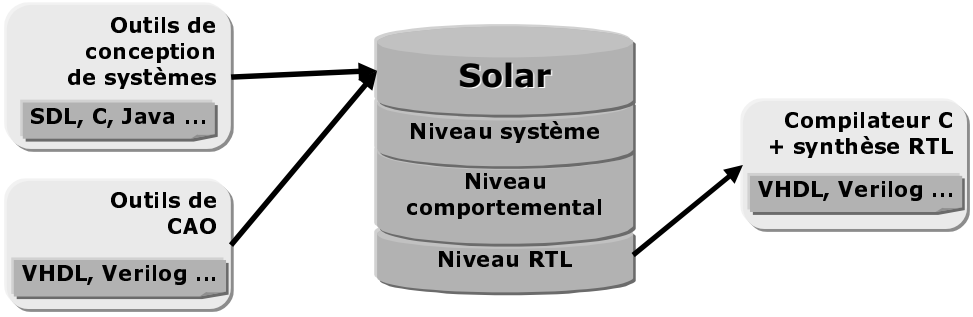


Figure 4 – Le contexte SOLAR

Les caractéristiques principales du modèle SOLAR sont :

- Des structures de données nécessaires aux étapes de co-spécification, de conception conjointe et de co-synthèse ;
- Permettre l'utilisation d'un modèle unique pour les conceptions de systèmes logiciels/matériels complexes et distribués ;

- La représentation peut manipuler à la fois les concepts fonctionnels et les concepts de réalisation facilitant ainsi considérablement la transition entre les différentes étapes de la conception ;
- SOLAR supporte plusieurs niveaux d'abstractions, lui permettant de modéliser des circuits intégrés aussi bien que des systèmes logiciel/matériel distribués.

Le modèle SOLAR combine deux concepts puissants au niveau système :

- Un modèle de machine d'états finis étendue pour décrire le comportement, représenté par des tables d'états ;
- Un mécanisme à base d'appel de procédures à distance pour spécifier la communication de haut niveau, représentée par des canaux.

Comme montre la Figure 5, SOLAR modélise la structure du système par une hiérarchie d'unités de conception. Un système est composé d'un ensemble d'unités qui communiquent. Le comportement des unités « feuilles » est décrit par des tables d'états. La communication entre les unités est modélisée par des appels de procédures à distance. L'unité « canal » est composée d'un contrôleur, d'un ensemble de services et d'un ensemble de signaux d'interconnexion.

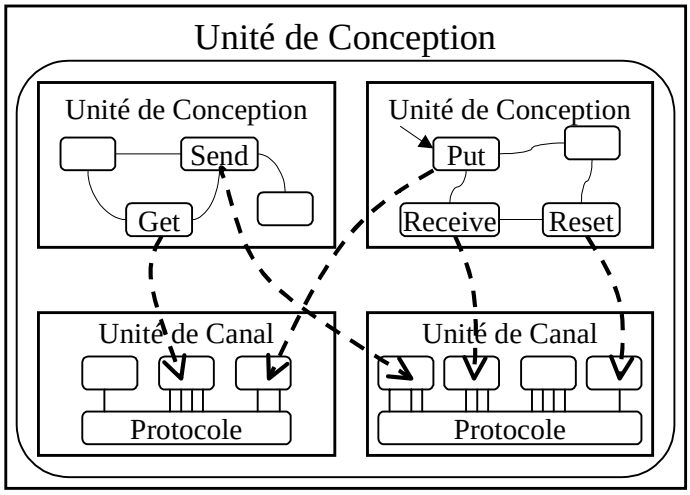


Figure 5 – Les concepts de base de Solar

Les constructeurs SOLAR sont :

- La *Table d'état* est le constructeur de base pour décrire le comportement, représenté par une machine d'états finis étendue;

- Les *Unités de Conception* permettent la structuration de la description du système en un ensemble de sous-systèmes en interaction (processus). Ces sous-systèmes interagissent avec l'environnement en utilisant une frontière bien définie;
- Les *Unités de Canal* réalisent la communication entre les unités de conception. L'unité canal contient la spécification du protocole de communication;
- Les *Unités Fonctionnelles* sont utilisées pour spécifier des opérateurs complexes (DCT, FFT). Ce concept permet la spécification et la réutilisation d'opérateurs partagés entre MEF séquentielles.

2.3 La synthèse comportementale avec MUSIC

La synthèse comportementale de MUSIC est utilisée pour les parties matérielles produites par le flot de synthèse au niveau système (voir Figure 1). Elle prend une description algorithmique du circuit, où il n'y a ni signal d'horloge ni information structurelle, et produit une description composée de transferts de registres (RTL). Les transferts de données sont réalisés à l'intérieur des périodes d'horloge, c.-à-d., la synchronisation est bien définie. La synthèse comportementale emploie une bibliothèque de composants de trois types : des unités fonctionnelles, des unités de stockage et des unités d'interconnexion. La synthèse RTL produit un réseau de portes interconnectées, en utilisant des cellules provenant d'une bibliothèque liée à une technologie de fabrication. Les modules logiciels suivent le flot de synthèse du logiciel (compilation, débogage) et le mécanisme de communication entre les divers modules est généré par la synthèse d'interface (voir Figure 1).

En ce qui concerne la synthèse comportementale, la littérature mentionne trois tâches fortement interdépendantes : l'ordonnancement, l'allocation et l'affectation des ressources [Gaj92][McFA90].

1. L'ordonnancement définit l'ordre relatif de l'exécution des calculs et transferts de données définis dans la description algorithmique d'entrée. Les techniques d'ordonnancement ne sont pas les mêmes pour les applications orientées vers le flot de contrôle que pour celles orientées vers le flot de données. Pour les applications orientées vers le flot de contrôle, il y a : l'ordonnancement basé sur chemins [Cam91] et à boucles dynamiques (DLS, pour *Dynamic Loop Scheduling*) [ORJ93] ; pour les applications orientées vers le flot de données, les techniques plus utilisées sont : ASAP (pour *as soon as possible*), ALAP (pour *as late as*

possible), LIST (l'ordonnancement à base de liste) [Hu61] et FDS (pour *force-directed scheduling*) [PaKn89].

2. L'allocation choisit un ensemble de ressources dans une bibliothèque de composants. Elle doit allouer des ressources suffisantes pour exécuter tous les calculs, stocker toutes les variables et exécuter tous les transferts de données définis dans la description d'entrée.
3. L'affectation fait l'assignation des calculs aux unités fonctionnelles, l'assignation des variables aux éléments de mémoire et l'assignation des transferts de données aux éléments d'interconnexion.

Si l'ordonnancement est fait en premier, les calculs qui doivent être réalisés en parallèle seront définis. Ainsi, pendant la phase d'allocation, il faudra allouer des ressources suffisantes pour obtenir ce degré prédéfini de parallélisme. D'autre part, si l'allocation est faite d'abord, le nombre maximum de calculs qui pourraient être exécutés en parallèle sera limité par les ressources disponibles. Par conséquent, la phase d'ordonnancement sera contrainte à respecter de telles limitations. Le même genre d'analyse peut être fait au sujet de l'affectation. Par exemple, si toutes les multiplications sont liées à une seule instance de multiplicateur, l'ordonnancement ne pourra pas définir deux multiplications en parallèle. Dans ce cas, l'affectation d'interconnexion doit garantir que tous les arguments des multiplications peuvent être conduits au même multiplicateur. Cette argumentation montre qu'il y a une interdépendance très forte entre l'ordonnancement, l'allocation et l'affectation des ressources.

La solution optimale pourra être obtenue si l'ordonnancement, l'allocation et l'affectation des ressources sont traités comme un seul problème d'optimisation. Une des solutions est d'utiliser les techniques de la programmation linéaire en nombres entiers (ILP, pour *integer linear programming*) [LHLin89]. Cependant, seulement de petits problèmes peuvent être traités de cette façon, parce qu'il s'agit d'un problème NP-complet [Gaj92]. En conséquence, des heuristiques sont requises pour résoudre ces problèmes séparément. La plupart des outils de synthèse comportementale emploient un ordre fixe d'ordonnancement, d'allocation et d'affectation. Il est très difficile d'obtenir les solutions optimales avec cette approche, c'est pourquoi quelques outils emploient un flot flexible de synthèse.

La méthodologie de synthèse comportementale utilisée par MUSIC est basée sur un flot flexible de synthèse. Les caractéristiques principales de cette méthodologie sont :

- Le domaine d'application est principalement celui des contrôleurs embarqués ;
- Le style d'application est celui dominé par le flot de contrôle, mais l'enchaînement des opérateurs et le *pipeline* (avec quelques restrictions) sont aussi traités en cas de descriptions avec calculs intensifs ;
- La méthodologie de synthèse est flexible, le concepteur définit le flot de synthèse et le style d'interconnexion qui sera employé dans l'architecture cible ;
- Tous les outils utilisent le format intermédiaire SOLAR ;
- Les modèles d'entrée et de sortie du circuit sont décrits dans un sous-ensemble du langage VHDL ;

2.3.1 Les techniques de conception pour la synthèse comportementale avec MUSIC

La plupart des systèmes de synthèse comportementale utilisent un flot de conception fixe, c.-à-d., ils exécutent les trois tâches (l'ordonnancement, l'allocation et l'affectation des ressources) de synthèse dans un ordre prédéfini. Dans MUSIC, plusieurs flots sont possibles et le flot de conception peut être défini par le concepteur selon les caractéristiques de la application (voir Figure 6).

Le troisième chapitre montre que le flot de synthèse idéal n'existe pas. Différents flots sont requis pour différents styles d'application et différentes priorités d'optimisation par rapport à la surface, le délai et la consommation d'énergie. Le quatrième chapitre, montre que différents modèles architecturaux sont nécessaires si le domaine d'application n'est pas restreint. Particulièrement, l'impact des différents schémas d'interconnexion sur la surface du circuit est montré.

Les techniques de synthèse comportementale utilisées par MUSIC peuvent être divisées en deux groupes : les outils d'ordonnancement (ordonnancement et ré-ordonnancement, voir la Figure 6) et les outils d'allocation et d'affectation (pour les unités fonctionnelles et pour les unités d'interconnexion). Dans MUSIC, la génération d'architecture fait appel aux algorithmes d'allocation et d'affectation d'interconnexion. Le cinquième chapitre décrit en détail les algorithmes développés pour le ré-ordonnancement, l'allocation et l'affectation des unités fonctionnelles et des interconnexions.

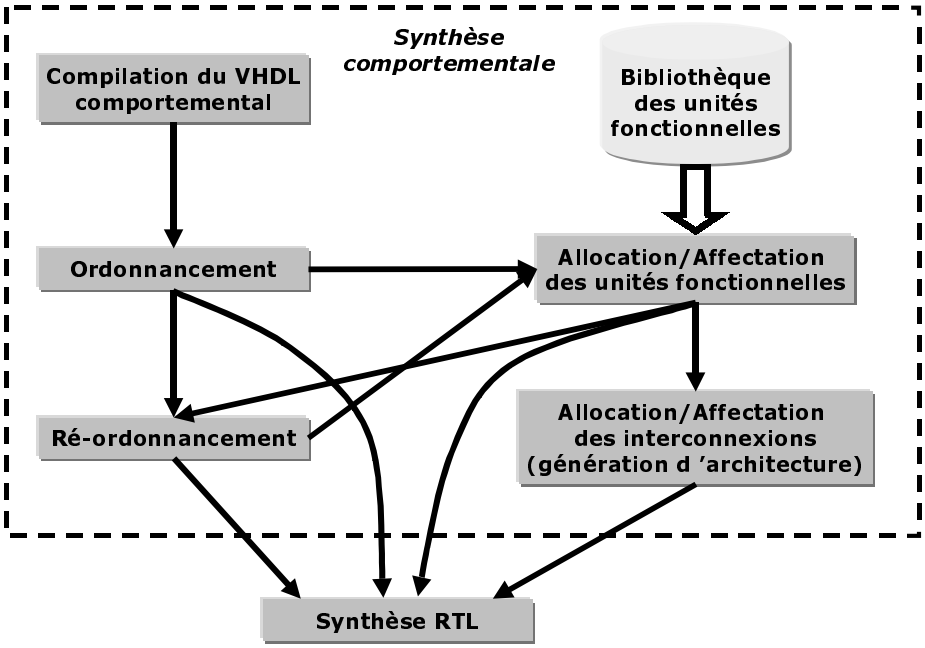


Figure 6 – Le flot de synthèse comportemental de MUSIC

2.3.1.1 La compilation de la description VHDL comportementale

L'entrée principale de la synthèse comportementale réalisée par MUSIC est un algorithme décrit en VHDL (pour *Very high speed integrated circuit Hardware Description Language*). VHDL est un langage standard (IEEE 1076 et 1164) pour la conception et la description des systèmes électroniques. La syntaxe du langage est standard, tandis que la sémantique pour la synthèse est un problème encore non résolu. Quelques travaux ont essayé de résoudre le problème de la variance syntaxique vis-à-vis de la synthèse comportementale [LiGup96][CGR93]. Au niveau RTL, un comité de normalisation a été créé (IEEE 1076.6) pour définir une syntaxe et une sémantique communes à tous les outils de synthèse RTL conformes aux normes [VHDLT98]. Le but est de garantir des résultats uniformes, semblables aux outils de simulation qui utilisent des modèles VHDL au niveau RT. Malheureusement, les outils courants au niveau comportemental définissent leur propre sous-ensemble syntaxique et ont leur propre interprétation sémantique pour la synthèse.

L'étape de compilation traduit la description comportementale d'entrée dans un graphe de flot de contrôle (voir Figure 7). L'étape de compilation effectue la vérification syntaxique de la description VHDL et génère un arbre syntaxique correspondant. Un traitement supplémentaire permet d'en extraire les informations pertinentes sous la forme d'un graphe de flot de contrôle. Le graphe de contrôle (ou graphe de flot de contrôle et de données) ainsi obtenu est composé :

- De nœuds pouvant correspondre soit à des opérations, soit à des instructions d'attente ;
- D'arcs rattachés à une condition et décrivant la séquence possible des opérations ;
- Et d'entrées et sorties pour les connexions du circuit avec le monde externe.

Figure 7 – La compilation VHDL

Dans le graphe de contrôle, les opérations sont séquencées suivant l'ordre d'écriture utilisé, sans notion explicite de temps. Cette description sera utilisée par l'ordonnancement et transformée en une machine d'états finis comportementale.

2.3.1.1.1 L'interprétation syntaxique

Une description VHDL est composée d'entités et d'architectures. Une entité définit l'interface entre une conception et son environnement par une liste de ports d'entrée et de sortie. L'architecture décrit comment l'entité se comporte ou sa composition. Dans MUSIC, seulement une paire « entité/architecture » est considérée et seulement un processus peut être synthétisé. Ce processus peut décrire le comportement du circuit comme un algorithme. La plupart des constructions du langage VHDL sont acceptées par la synthèse comportementale ([JDKR97] discute en détail l'interprétation de chaque construction pour la synthèse) :

- Les instructions conditionnelles et le *case* ;
- Les boucles (même avec l'arrêt dépendant de l'exécution) ;
- Les appels de procédures et de fonction ;
- Les expressions avec des variables et des signaux ;
- Les instructions d'attente (*wait*).

Une fonction ou une procédure peut être synthétisée de deux manières : elle peut être expansée (pour *expanded inline*) ou interprétée comme une opération complexe qui doit être exécutée sur une unité fonctionnelle. Cette dernière interprétation donne accès à un nombre

d'opérations infiniment complexes. La description comportementale admise par MUSIC peut inclure plusieurs instructions d'attente à l'intérieur d'un même processus. Il est possible de combiner des instructions d'attente sur différents signaux et des conditions complexes avec des boucles et des instructions de contrôle. L'utilisation de plusieurs instructions d'attente à l'intérieur d'un même processus donne un grand avantage : elle permet une description facile des applications avec des protocoles complexes d'échange de données avec le monde externe.

2.3.1.1.2 L'interprétation sémantique

L'interprétation sémantique des descriptions VHDL est étroitement liée à la façon dont les flots d'exécution (pour *threads*) et les opérations d'entrée-sortie de données sont ordonnancées. La synthèse comportementale emploie quelques transformations complexes qui peuvent changer l'ordre d'entrée-sortie des données. Afin de comprendre le résultat de la synthèse, il est fondamental de connaître l'interprétation sémantique exacte de la description d'entrée.

Un flot d'exécution est défini comme un code exécuté entre deux instructions d'attente successives pendant la simulation. Une description comportementale en VHDL peut être vue en tant qu'une machine d'états finis à un niveau très élevé ; dans laquelle les états correspondent aux instructions d'attente et les transitions aux différents flots d'exécution. Un flot d'exécution peut inclure des calculs complexes et des boucles dépendant de données. Les fonctions principales de la synthèse comportementale sont : l'extraction des flots d'exécution à partir de la description d'entrée et leur ordonnancement, c.-à-d., leur répartition aux cycles d'horloge. Il y a notamment trois classes d'ordonnancement des flots d'exécution [Kna96][JDKR97]:

1. **L'ordonnancement à cycle fixe :** chaque flot d'exécution est réalisé dans une étape unique de contrôle qui peut être exécutée durant un seul cycle d'horloge. La période d'horloge doit être assez longue pour permettre l'exécution de toutes les opérations du flot d'exécution le plus complexe. Les flots d'exécution ne doivent pas inclure ni boucles dépendantes de données, ni boucles infinies, ni opérations multicycle ;
2. **L'ordonnancement sur des super états :** chaque flot d'exécution peut être réalisé par une séquence d'étapes de contrôle (un super état). L'ordonnancement définit le nombre d'étapes de contrôle de chaque super état. Les flots d'exécution ne doivent pas inclure ni boucles dépendantes de données, ni boucles infinies ;

3. **L'ordonnancement sur des états comportementaux :** chaque flot d'exécution est réalisé par une MEF. Chaque transition de cette MEF est exécutée durant un seul cycle d'horloge. Il n'y a aucune restriction quant au contenu des flots d'exécution.

Pendant la synthèse comportementale, l'ordre relatif d'exécution des opérations peut changer. Ces transformations peuvent être utiles pour le partage des ressources, mais peuvent également affecter les opérations d'entrée-sortie de données. Il y a deux stratégies d'ordonnancement pour ces opérations :

1. **L'ordonnancement avec des cycles d'entrée-sortie fixes :** l'ordre relatif des opérations d'entrée-sortie de données est préservé ;
2. **L'ordonnancement avec fluctuation libre des entrées-sorties :** Les opérations d'entrée-sortie sont traitées comme les autres instructions VHDL.

MUSIC emploie l'ordonnancement sur des états comportementaux et avec des cycles d'entrées-sorties fixes. Cette approche est un bon compromis entre la liberté d'écriture des descriptions comportementales et l'exploration de l'espace des solutions.

2.3.1.2 L'ordonnancement

L'ordonnancement fait l'attribution des calculs présents dans la description d'entrée aux étapes de contrôle (voir Figure 8). Il produit une machine d'états finis comportementale (BFSM, pour *behavioral finite state machine*) dans laquelle chaque transition correspond à une étape de contrôle. Dans cette étape, il est possible de faire des compromis entre le nombre d'étapes de contrôle et les ressources requises pour exécuter l'algorithme. Chaque transition peut, en réalité, contenir un graphe de flot de données qui pourrait prendre plusieurs cycles d'horloge pour s'exécuter. La phase de ré-ordonnancement traite ces graphes de flot de données et définit la période du cycle d'horloge.

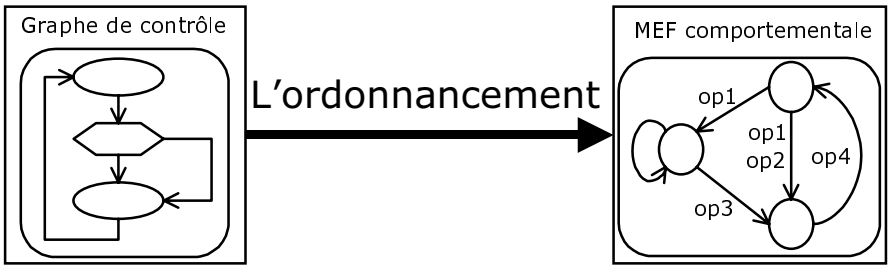


Figure 8 – L'ordonnancement

L'algorithme employé par MUSIC est l'ordonnancement à boucles dynamiques (DLS, pour *dynamic loop scheduling*) [ROA94]. DLS traite les boucles en optimisant les effets de bord. Il permettra aux calculs d'itérations successives d'être exécutés en parallèle si la dépendance de données peut être respectée. La BFSM créée contient le parallélisme maximal qui a pu être extrait à partir de la description initiale. Les calculs sont ordonnancés pour s'exécuter aussitôt que possible (ASAP, pour *As Soon As Possible*).

2.3.1.3 L'allocation et l'affectation des unités fonctionnelles

L'allocation des unités fonctionnelles définit les types et le nombre d'unités fonctionnelles qui seront employés dans le circuit (voir Figure 9). L'affectation choisit les unités fonctionnelles pour l'exécution de chaque calcul présent dans la BFSM d'entrée. Les calculs appartenant à une même transition de la BFSM ne peuvent pas être exécutés par la même unité fonctionnelle. Dans MUSIC, l'allocation et l'affectation des unités fonctionnelles peuvent être faites après l'ordonnancement ou le ré-ordonnancement. L'algorithme employé est basé sur l'algorithme du graphe bipartite pondéré (BWGM, pour *bipartite weighted graph matching*), algorithme présenté par [Hsu90].

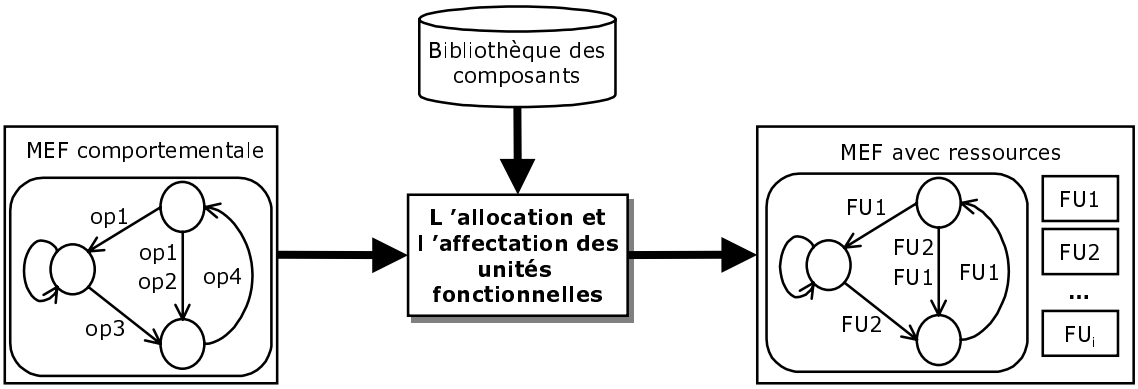


Figure 9 – L'allocation et l'affectation des unités fonctionnelles

La bibliothèque des composants est un fichier défini par l'utilisateur. Il contient des informations concernant l'interface des unités fonctionnelles disponibles pour la phase d'allocation et d'affectation. Il n'y a aucune description de la fonctionnalité interne des unités fonctionnelles dans cette bibliothèque, c.-à-d., elles sont décrites en tant que boîtes noires. Les unités fonctionnelles peuvent être des unités arithmétiques et/ou logiques simples ou des co-processeurs complexes produits par des sessions de synthèse précédentes. Deux genres d'information doivent être fournis : des informations structurelles sur les ports et des

informations comportementales sur le protocole d'accès. Il est permis aux protocoles d'accès de prendre plus d'un cycle d'horloge pour leurs exécutions.

2.3.1.4 Le ré-ordonnancement

Le ré-ordonnancement réalise deux fonctions principales : insérer de nouveaux états dans la BFSM et changer l'enchaînement des opérations (voir Figure 10). Quelques états sont introduits pour respecter les protocoles d'accès des unités fonctionnelles multicycles. En cassant l'enchaînement des opérations, le délai du chemin critique peut être réduit (c'est un type de *re-timing* en haut niveau). Par exemple, sur la Figure 10 le nœud en grisé dans la transition allant de l'état numéro 1 à l'état numéro 3 représente une unité fonctionnelle qui s'exécute en deux cycles ; après le ré-ordonnancement, l'état numéro 4 a été ajouté pour respecter cette contrainte. En outre, la chaîne de calcul représentée par des nœuds en grisé dans la transition allant de l'état numéro 3 à l'état numéro 2 peut prendre plus qu'un cycle d'horloge ; le ré-ordonnancement a inséré l'état numéro 5 pour casser la chaîne de calcul.

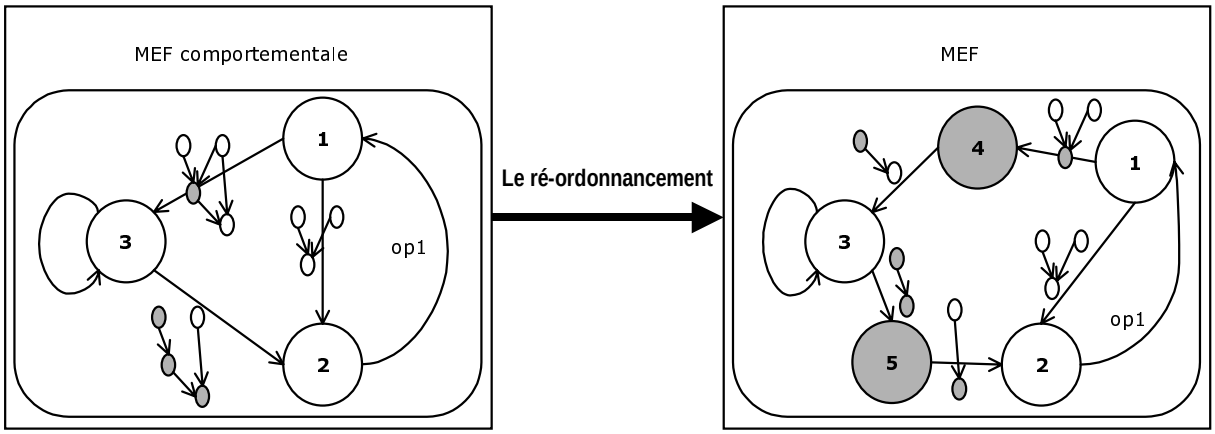


Figure 10 – Le ré-ordonnancement

Dans MUSIC, l'algorithme de ré-ordonnancement est basé sur l'algorithme classique pour des graphes de flot de données connu sous le nom de LIST *scheduling* [McFar86]. Dans ce type d'ordonnancement les ressources sont des contraintes et le but est de trouver la solution avec le moins d'étapes de contrôle possibles. Il y a une autre classe d'algorithmes d'ordonnancement dans lesquels le nombre d'étapes de contrôle est le contraint, par exemple, le *force-directed scheduling* [PaKn89]. Le ré-ordonnancement peut être fait après l'ordonnancement ou après l'allocation et l'affectation des unités fonctionnelles.

2.3.1.5 L'allocation et l'affectation des interconnexions

L'allocation et l'affectation des interconnexions définissent la façon de transporter les données des éléments de mémoire aux unités fonctionnelles. Le parallélisme entre les transferts de données spécifiés dans l'algorithme et ceux impliqués par les arguments de calculs doit être respecté. Après cette étape, des bus, des cellules à trois états ou des multiplexeurs sont créés selon l'architecture cible choisie (voir Figure 11).

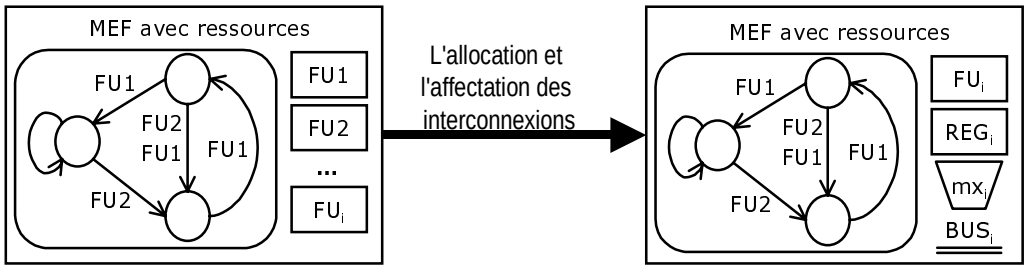


Figure 11 – L'allocation et l'affectation des interconnexions

Dans MUSIC, l'allocation et l'affectation des mémoires sont faites implicitement par l'étape d'ordonnancement. Les éléments de mémoire spéciaux (RAM, ROM, FIFO, banc de registres, etc.) peuvent être employés si décrits comme unités fonctionnelles liées avec des appels de procédure et/ou fonction.

De nouveaux algorithmes ont été développés pour résoudre le problème de l'allocation et de l'affectation des interconnexions. Le but de ces algorithmes est de réduire le nombre de cellules d'interconnexion, comme sera décrit dans le cinquième chapitre. Cette étape doit être faite après l'allocation et l'affectation des unités fonctionnelles.

2.3.1.6 La génération d'architecture

La génération d'architecture produit un modèle VHDL de l'architecture cible composé d'un contrôleur et d'un chemin de données (voir Figure 12). Le chemin de données est un modèle structurel contenant un réseau de composants interconnectés : des unités fonctionnelles, des registres, des multiplexeurs, des bus, des cellules à trois états, etc. Le contrôleur envoie les signaux pour commander l'écriture des registres, la sélection des chemins d'interconnexion et la sélection du mode d'opération des unités fonctionnelles. Et il reçoit les comptes-rendus du chemin de données.

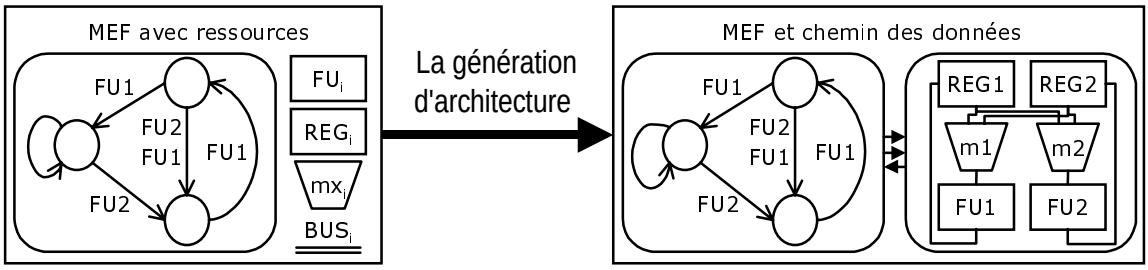


Figure 12 – La génération d'architecture

Le contrôleur est décrit par son comportement ; c'est une machine d'états finis du type Mealy. Les entrées sont les signaux de contrôle externes et les comptes-rendus du chemin de données, et les sorties sont les signaux de contrôle pour le chemin de données. La description VHDL du chemin de données est composée de plusieurs modules instanciés et interconnectés. Le contrôleur est représenté par deux processus concurrents : un qui définit les signaux de contrôle actifs à chaque transition et un autre qui synchronise la transition d'un état à l'autre sur les fronts du signal d'horloge. La génération d'architecture doit être faite après l'allocation et l'affectation des interconnexions.

2.4 Conclusion

Ce chapitre est une introduction générale aux concepts liés à la synthèse de haut niveau. Nous avons présenté les différents modèles de descriptions du circuit utilisés pour les outils de synthèse à chaque niveau d'abstraction. Ce travail se situe dans le flot de conception de systèmes utilisé par l'outil de synthèse appelé MUSIC. Cet outil est basé sur un processus de synthèse transformationnel. L'outil d'estimation de performance qui sera présenté dans le sixième chapitre aide à effectuer les choix architecturaux durant ce processus de synthèse. Pour la partie matérielle, MUSIC utilise un flot de conception flexible qui part d'une description comportementale du circuit. L'ordre d'exécution des trois tâches de synthèse comportementale (l'ordonnancement, l'allocation et l'affection des ressources) peut être modifié selon le style de l'application. Cette approche permet l'obtention de meilleurs résultats. Le flot de conception flexible sera détaillé dans le troisième chapitre.

Chapitre 3

La Flexibilité du Flot de Conception

[...] Certes, la galère domine
Le flot qui se ploie et s'incline,
Cependant, le flot seul est maître.
Petofi (Sándor), Un océan s'est soulevé.

Ce chapitre présente une approche interactive et flexible pour l'exploration de l'espace des solutions. La flexibilité de cette approche se manifeste à deux niveaux : au niveau du flot de conception et au niveau de l'interface entre la synthèse comportementale et la synthèse au niveau transferts de registres.

3.1 Introduction

Ce chapitre décrit deux dimensions de la flexibilité de notre système de synthèse comportementale : la flexibilité du flot de synthèse et la flexibilité de l'interface avec les outils de synthèse RTL. Le quatrième chapitre discute la flexibilité de l'architecture cible. Dans ce chapitre, nous démontrerons l'importance de la flexibilité du flot de synthèse.

Comme il a été expliqué dans la section 2.3, la synthèse comportementale peut être divisée en trois tâches interdépendantes : l'ordonnancement, l'allocation et l'affectation. Pour obtenir la solution optimale, il faut aborder ces trois problèmes en même temps. Malheureusement, la complexité des applications réelles rend cette approche inutilisable. Ainsi, la plupart des outils emploient des heuristiques pour résoudre les trois tâches dans un ordre donné. Il est clair qu'il n'y a aucun ordre qui garantit que la solution optimale peut être atteinte dans tous les cas. Une solution est de permettre aux trois tâches d'être exécutées dans n'importe quel ordre. Un flot flexible de synthèse permet au concepteur d'adapter la méthodologie de conception selon la nature de l'application et la qualité des outils de synthèse. La section 3.2 présente le flot de synthèse employé par quelques outils de synthèse comportementale choisis dans la littérature. Le flot flexible de synthèse utilisé par MUSIC (flexibilité du flot interne) est présenté dans la section 3.4.

La qualité du résultat de la synthèse dépend non seulement de la qualité des outils utilisés mais également de l'application elle-même. Il est impératif de donner au concepteur la possibilité d'explorer tous les flots possibles de synthèse. Ce chapitre propose un flot de synthèse avec des frontières flexibles entre la synthèse comportementale et la synthèse RTL. Cette possibilité, donne au concepteur beaucoup plus de contrôle sur le processus global de la synthèse. L'interface flexible avec la synthèse RTL (flexibilité du flot externe) sera présentée dans la section 3.3.

Le fossé sémantique entre une description algorithmique et une description RTL rend difficile la compréhension du résultat de la synthèse comportementale. Il y a un autre aspect lié à ce problème-là : la question du débogage. Le concepteur simule la description RTL générée, mais les modifications doivent être faites sur la description algorithmique d'entrée [Kroli98]. Néanmoins, la corrélation entre les deux descriptions n'est pas toujours évidente à cause du fossé sémantique. Afin de faire face à ces problèmes, l'environnement de synthèse doit faciliter la compréhension des résultats intermédiaires. Par exemple, en fournissant un

mécanisme de corrélation entre la description RTL synthétisée et la description algorithmique d'entrée. La section 3.5 présente les mécanismes fournis par MUSIC pour faciliter la compréhension des résultats intermédiaires du processus de synthèse. En conclusion, la section 3.6 montre l'importance d'avoir de multiples chemins de synthèse à l'aide de quelques exemples.

3.2 Les flots de synthèse comportementale

Le flot de synthèse comportementale traditionnel exécute les tâches de synthèse dans un ordre fixe. Il produit une architecture composée d'un contrôleur et d'un chemin de données pour les outils de synthèse RTL. Ce flot est habituellement une séquence d'ordonnancement, d'allocation et d'affectation de ressources (indépendamment de l'ordre). Ces tâches sont interdépendantes. Or, la plupart des outils de conception courants les exécutent en tant que tâches indépendantes. Quelques systèmes [LMD94][LHLin89] traitent les trois tâches comme un seul problème d'optimisation ; pourtant la complexité empêche l'application de cette approche dans la pratique. Il n'y a aucun ordre prédéfini pour faire l'ordonnancement et l'allocation de ressource. Cependant, c'est seulement après avoir fait l'ordonnancement qu'il est possible d'obtenir un modèle du système précis au niveau du cycle d'horloge. Pour cette raison, l'approche utilisée par MUSIC requiert que l'ordonnancement soit exécuté en premier.

Le flot de synthèse utilisé par MEBS [TsHsu92] peut produire des descriptions VHDL après l'exécution de chaque tâche de synthèse. Cependant, ces descriptions ne peuvent pas être synthétisées mais seulement être simulées. L'environnement interactif de synthèse (ISE, *Interactive Synthesis Environment*) [GICH96] et *Matisse* [KCGPT98] ont un flot flexible de conception puisqu'ils laissent accomplir les tâches de synthèse dans n'importe quel ordre. Néanmoins, ce sont également des systèmes intégrés, qui font la synthèse RTL et le planning d'emplacement en parallèle à la synthèse comportementale. En outre, l'interface avec d'autres outils de synthèse est seulement possible à un bas niveau d'abstraction. Les optimisations utilisées par HYPER peuvent être commandées par un outil qui guide la conception (*design guidance system*) [GPRab98] mais il ne peut pas être appliqué à l'ordre des tâches de synthèse. Le système ADAM [KnPar91] peut engendrer un ordre adéquat des tâches de conception pour une application donnée. Toutefois, il n'a pas une interface flexible avec la synthèse RTL.

La suite de ce chapitre propose un flot de synthèse comportementale flexible et une interface flexible avec la synthèse RTL. L'interaction avec le concepteur n'est pas limitée à l'indication des contraintes de synthèse. Il est possible de définir le flot de conception et de choisir la séquence d'outils pour exécuter les diverses tâches de synthèse. Par exemple, en estimant les paramètres de qualité du circuit, il est possible de choisir entre les estimateurs de niveau comportemental qui sont rapides [KuDu97] et les estimateurs de niveau RT qui sont précis mais lents. Par conséquent, une exploration efficace de l'espace des solutions peut être réalisée.

3.3 L'interface flexible avec la synthèse RTL

Le flot de synthèse classique commence avec une description algorithmique pour produire une architecture. Ensuite la synthèse RTL utilise cette architecture pour produire un réseau de portes interconnectées dans une technologie donnée (voir Figure 13a). Les outils de synthèse comportementale exécutent quatre étapes principales : l'ordonnancement, l'allocation, l'affectation et la génération d'architecture. Dans la plupart des cas, les outils de synthèse RTL exécutent seulement la synthèse de MEF et la transposition technologique. Pourtant, la nouvelle génération d'outils de synthèse RTL peut également exécuter l'allocation et l'affectation de ressources (voir Figure 13b). Donc, il peut y avoir un chevauchement de fonctionnalité entre les outils de synthèse comportementale et les outils avancés de synthèse RTL (comme montre la boîte avec double bordure dans la Figure 13b). Ce chevauchement rend possible d'autres flots de synthèse : nous pourrions passer de la synthèse comportementale à la synthèse RTL juste après l'ordonnancement ou après une allocation/affectation partielle (comme est représenté par les grandes flèches grises sur la Figure 13b). Dans cette section, nous présenterons un modèle de circuit qui permet la mise en œuvre d'une interface flexible entre l'outil de synthèse comportementale et RTL en profitant de ce chevauchement de fonctionnalité.

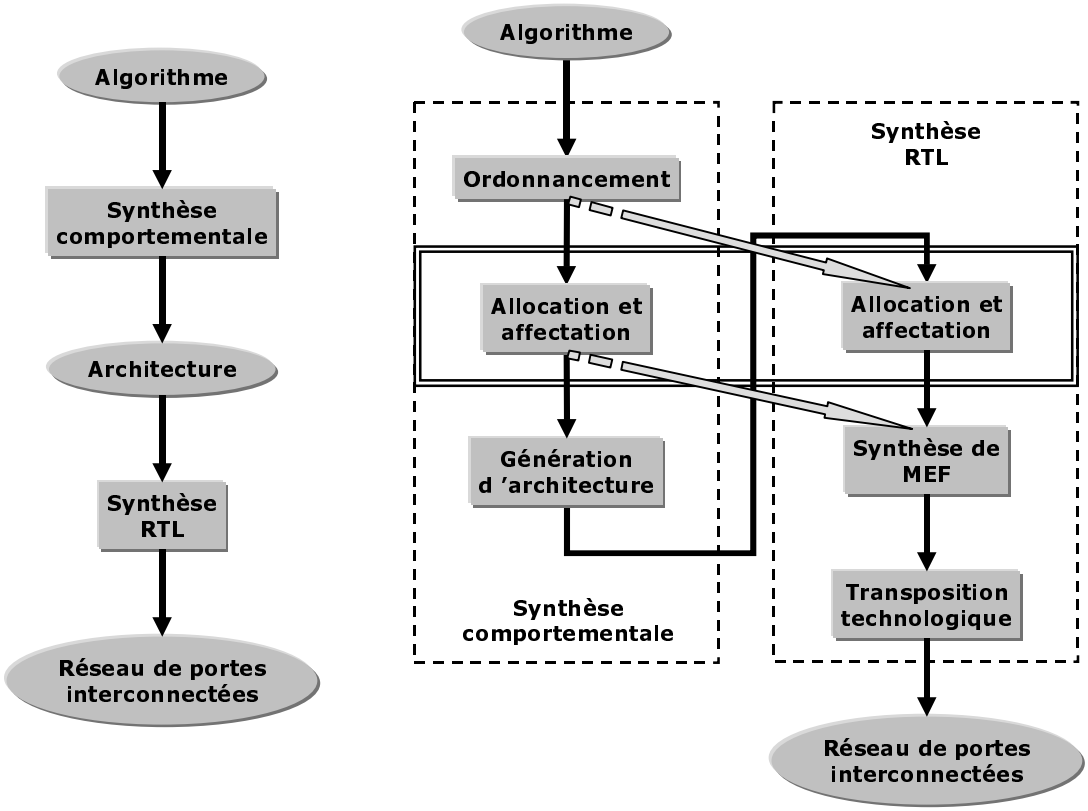


Figure 13 – Le chevauchement de fonctionnalité entre la synthèse comportementale et RTL

3.3.1 Les modèles de circuit et le séquencement

Les outils de synthèse manipulent le circuit à différents niveaux d'abstraction, du plus haut niveau (niveau système) au plus bas niveau (niveau physique). Les modèles de circuit et les points de synchronisation sont différents pour chaque niveau d'abstraction. Dans cette section, nous étudierons les modèles de circuit utilisés pendant le flot de synthèse partant d'un niveau de système vers le bas niveau RT.

La synthèse comportementale et la synthèse RTL opèrent à différents niveaux d'abstraction du circuit. En ce qui concerne les systèmes numériques synchrones, les différences essentielles entre ces niveaux d'abstraction sont les points de synchronisation utilisés [KuMic92]. Au niveau d'abstraction comportementale, les points de synchronisation définissent le séquencement de l'exécution de l'algorithme. A ce niveau d'abstraction, les points de synchronisation sont les interactions du système avec l'environnement externe. En VHDL, ceux-ci correspondent aux instructions d'attente (*Wait statements*), dans StateChart, à des états définis explicitement par le langage. L'ordonnancement transforme le modèle initial en une machine d'états finis. D'une façon générale, une transition de la MEF prendra

seulement un cycle d'horloge pour s'exécuter. Ce modèle est appelé machine d'états finis avec chemin de données (FSMD) [Gaj92]. Les étapes d'allocation et d'affectation des unités fonctionnelles lient les opérations (par exemple, des opérations arithmétiques) aux composants d'une bibliothèque. L'allocation d'unités d'interconnexion détermine les multiplexeurs ou les bus nécessaires pour la formation du chemin de données. Le résultat final de la synthèse comportementale est généralement le modèle d'architecture composée d'un contrôleur et d'un chemin de données. Au niveau d'abstraction RTL, la synchronisation est faite explicitement par un (ou plusieurs) signal (signaux) d'horloge. Le code exécuté pendant une transition est composé de transferts simples entre registres. Par exemple, il ne peut pas inclure des boucles dont les tailles dépendent des données. Par conséquent, un modèle précis au niveau du cycle d'horloge est fondamental pour la synthèse RTL.

Pendant la synthèse, nous allons du niveau d'abstraction le plus haut au niveau le plus bas, divers points de synchronisation et modèles de circuit sont employés (Tableau 2). Au niveau système, nous utilisons des processus communicants qui se synchronisent par l'échange des messages. Après le découpage [GuMi90], chaque processus pourrait être représenté au niveau algorithmique par un graphe de flot de contrôle/donnée. Ce graphe est synchronisé par des événements d'entrée-sortie ou des opérations d'attente. Le graphe peut également être représenté utilisant une FSMD comportementale. La synthèse comportementale utilise ce graphe pour produire un modèle RTL. Au niveau transfert de registres, les transferts de données sont synchronisés aux fronts du signal d'horloge. La synthèse RTL suivie par la synthèse logique transposeront les composants du contrôleur et du chemin de données dans une bibliothèque de cellules afin de produire un réseau de portes interconnectées (*netlist*). En fin de compte, la synthèse physique produira le schéma final du dessin des masques. Au niveau physique, les changements de valeur des fils définissent les données valides.

Niveaux d'abstraction	Points de synchronisation	Modèle de description du circuit
Système	Messages inter-processus	Processus communicants
Algorithmique	Evénements d'entrée-sortie	GFC, GFD, GFCD, FSMD
Transfert de registres	Signal d'horloge	MEF, FSMD, BDD, Equations Booléennes
Physique	Changement de valeur des fils	Réseau de cellules interconnectées et dessin de masques

Tableau 2 – Les modèles de circuit et les points de synchronisation

3.3.2 Le champ d'application des synthèses comportementale et RTL

Afin de profiter du chevauchement de fonctionnalité entre la synthèse comportementale et la synthèse RTL, les flots de synthèse doivent être modifiés. Il devrait être possible de passer à la synthèse RTL à n'importe quelle étape du flot de synthèse comportementale. Ce qui introduirait un degré supplémentaire de liberté au processus de synthèse, puisqu'un modèle architectural strict n'est pas imposé comme la seule interface possible.

Le Tableau 3 détaille les différentes tâches de synthèse exécutées par les outils de synthèse comportementale et RTL. Les différents modèles du circuit sont également détaillés pour chaque tâche. Ce tableau montre clairement qu'il peut y avoir un chevauchement de fonctionnalité (sur les lignes en gris) entre les outils de synthèse comportementale et RTL. La précision du modèle de circuit produit par la synthèse comportementale dépend des tâches accomplies. Chaque étape de synthèse raffine le modèle initial. L'exécution de toutes les tâches produira une architecture composée de contrôleur et chemin de données. Les modèles d'entrée pour la synthèse RTL vont des FSMDs précises au niveau du cycle d'horloge, aux architectures complètement spécifiées. Tous ces modèles ont quelque chose en commun : ils synchronisent aux fronts du signal d'horloge, c.-à-d., sont précis au niveau du cycle d'horloge près. Ce niveau de précision est seulement atteint après la tâche d'ordonnancement de la synthèse comportementale. Le modèle FSMD précis au niveau du cycle d'horloge est la clef pour l'intégration des synthèses comportementale et RTL. Le Tableau 3 montre que ce modèle peut être employé pour faire l'allocation et l'affectation des ressources dans l'intervalle de chevauchement de fonctionnalité.

Tâches de synthèse	Points de synchronisation	Modèle de description du circuit
Ordonnancement	Evénements d'entrée-sortie	CFG
Allocation des ressources	Signal d'horloge	FSMD
Affectation des ressources	Signal d'horloge	FSMD avec ressources
Synthèse logique	Signal d'horloge	MEF et chemin de données
Synthèse physique	Changement de valeur des fils	Réseau de cellules interconnectées

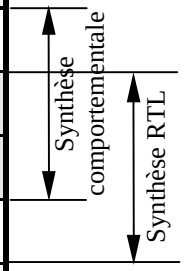


Tableau 3 – Le champ d'application de la synthèse comportementale et RTL

L'allocation et l'affectation des ressources produisent un modèle FSMD avec ressources. L'allocation et l'affectation d'unités de stockage ou de mémorisation assignent les registres/mémoires aux variables, aux constantes et aux tableaux. L'allocation d'unités fonctionnelles affecte des unités fonctionnelles aux opérations. L'allocation d'unités d'interconnexion définit les chemins entre les unités de stockage et les cellules de calcul. En fin de compte, une architecture composée de contrôleur et de chemin de données est créée. Les informations sur les ressources peuvent être inachevées, puisque nous pourrions être intéressés à faire l'allocation et l'affectation des ressources en deux étapes. Par exemple, des opérations complexes peuvent être traitées par la synthèse comportementale tandis que les opérations simples sont transférées à la synthèse RTL. Dans ce cas-ci, nous devons avoir la possibilité de traduire un modèle FSMD avec des informations partielles sur les ressources dans un format acceptable par la synthèse RTL.

Les outils de synthèse comportementale et RTL peuvent exécuter l'allocation et l'affectation des ressources et la génération de l'architecture. Le Tableau 4 compare l'exécution de ces tâches par différents types d'outils de synthèse. Dans MUSIC, l'allocation de mémoire est implicite, c.-à-d., il y a un registre pour chaque variable trouvée dans la description d'entrée. L'allocation et l'affectation des unités fonctionnelles au niveau comportemental est capable de traiter le partage des fonctions complexes (par exemple, des co-processeurs). Le partage d'unités fonctionnelles réalisé au niveau RTL se limite aux opérations simples (comme l'addition, la soustraction et la multiplication). L'allocation et l'affectation des interconnexions au niveau comportemental permettent l'utilisation des multiplexeurs ou des bus avec cellules à trois états, alors que la synthèse RTL utilise seulement les multiplexeurs. Enfin, l'architecture produite par les outils au niveau

comportemental est hiérarchique et le chemin de données peut contenir des co-processeurs avec des contrôleurs esclaves. La synthèse RTL produit une architecture mise à plat sous la forme d'une MEF complexe.

L'étape de conception	Synthèse comportementale	Synthèse RTL
L'allocation/affectation des mémoires	Implicite	Implicite
L'allocation/affectation des unités fonctionnelles	Co-processeurs, unités complexes	Opérateurs simples (+,-,*,…)
L'allocation/affectation des interconnexions	Multiplexeurs ou bus	Multiplexeurs
La génération d'architecture	Hiérarchique ; contrôleur + chemin de données	Sans hiérarchie ; MEF complexe

Tableau 4 – L'exécution des tâches par différents types d'outils de synthèse

Le modèle FSMD précis au niveau du cycle d'horloge ne permet pas des opérations complexes. Par exemple, les opérations qui ont un temps d'exécution dépendant des données. Il y a deux façons de contourner ce problème. La première solution est de considérer les opérations complexes comme des appels de procédure et de les associer aux unités fonctionnelles externes. Les appels de procédure seront employés pour mettre en marche ces unités fonctionnelles externes et pour obtenir les résultats [Camp87]. Chaque appel de procédure doit prendre un cycle d'horloge seulement. Dans ce cas-ci, l'ordonnancement doit être capable de manipuler des appels de procédure. La deuxième solution est de décrire des opérations complexes par des procédures qui emploient des opérations simples seulement. Alors, les appels de procédures peuvent être expansés (ou mises à plat) [Vah95] et être ordonnés avec le reste de la description [LKMM95].

3.4 Le flot flexible de synthèse utilisé par MUSIC

Cette section traite du système de synthèse comportementale MUSIC, un environnement flexible de synthèse (voir Figure 14). La première tâche dans le flot de synthèse de MUSIC est la compilation de la spécification en VHDL (tâche A, voir la Figure 14). Elle produit un GFC qui sera employé par l'ordonnancement (tâche B). L'ordonnancement produit un modèle du système précis au niveau du cycle d'horloge.

Comme a été expliqué dans la section 2.3, MUSIC utilise deux tâches d'ordonnancement (à savoir l'ordonnancement et le ré-ordonnancement) entrelacées avec deux tâches d'allocation et d'affectation des ressources (pour les unités fonctionnelles et les interconnexions). Dans le précédent outil de synthèse **AMICAL** [Park92], ces tâches étaient exécutées dans un ordre fixe : l'ordonnancement, l'allocation et l'affectation des unités fonctionnelles, le ré-ordonnancement et l'allocation et l'affectation des interconnexions. Nous avons rendu ce flot flexible.

D'une façon générale, les outils de synthèse comportementale sont optimisés pour un certain style d'application : dominé par le flot de contrôle ou par le flot de données. Chaque style demande un outil d'ordonnancement approprié, basé sur un graphe de contrôle ou sur un graphe de données (respectivement). MUSIC accepte des applications de styles mixtes, premièrement l'ordonnancement est appliqué au GFC entier produisant une FSMD. Dans cette FSMD, chaque transition peut contenir des calculs complexes organisés dans un graphe de flot de données (GFD). Si nécessaire, le ré-ordonnancement (la tâche D) est employé sur chaque transition. L'allocation et l'affectation des ressources traitent les unités fonctionnelles (tâche C), et les interconnexions (tâche E). Les travaux de cette thèse ont aussi concerné le développement des tâches C, D et E qui seront présentées dans le cinquième chapitre.

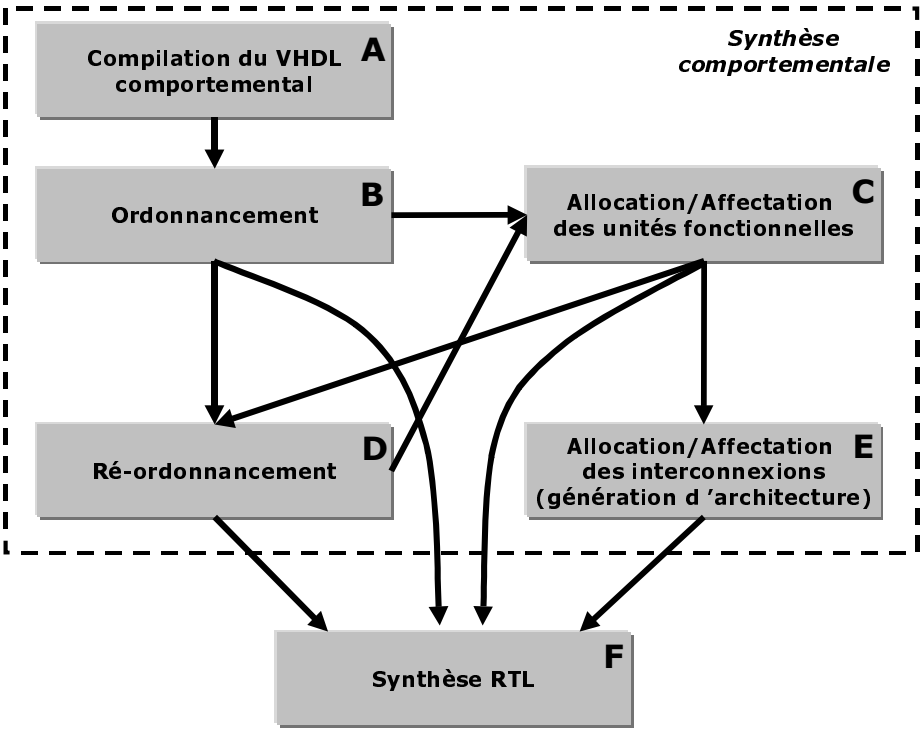


Figure 14 – Le flot flexible de conception utilisé par MUSIC

Tous les arcs de la Figure 14 représentent des séquencements possibles des tâches de la synthèse. Ainsi plusieurs flots peuvent être définis. Les divers flots de conception ou chemins de synthèse présentés dans la Figure 14 sont réalisables avec MUSIC. Chaque chemin allant du nœud A au nœud F constitue un flot de synthèse possible. S'il n'y a aucune opération multicycle ou complexe à effectuer, l'allocation d'unités fonctionnelles peut être réalisée par la synthèse RTL. Pour certains systèmes orientés à flot de contrôle, la synthèse RTL peut se faire juste après l'ordonnancement. Le ré-ordonnancement peut être fait avant ou après l'allocation d'unités fonctionnelles. Enfin, pour quelques systèmes orientés à flot de contrôle, le ré-ordonnancement peut ne pas être nécessaire.

Dans la section 3.6, nous essayerons les différents flots de synthèse réalisables avec le système MUSIC sur un ensemble d'exemples d'application.

3.5 L'interprétation des résultats intermédiaires

Comme indiqué dans l'introduction de ce chapitre, les concepteurs doivent faire face au fossé sémantique qui sépare la description comportementale de la description RTL générée. Pour corriger l'architecture, des modifications doivent être faites dans la description algorithmique (le problème du débogage) [Kroli98]. Dans MUSIC, la génération de résultats intermédiaires synthétisables peut aider à mieux comprendre le processus de synthèse, puisque le concepteur a la possibilité d'examiner les résultats partiels du flot de synthèse.

3.5.1 Les modèles VHDL synthétisables

Dans MUSIC, le modèle employé par les outils de synthèse comportementale est le modèle FSMD précis au niveau du cycle d'horloge. En VHDL, ce modèle peut être décrit en utilisant la structure traditionnelle pour décrire la partie MEF et un réseau des composants interconnectés pour décrire le chemin de données. Chaque transition de la MEF prendra un cycle d'horloge pour s'exécuter, parce que les outils RTL ne font pas d'ordonnancement. Les descriptions du circuit produites par MUSIC emploient trois types de modèles FSMD : la MEF comportementale, la MEF avec ressources et la MEF avec chemin de données. Tous ces modèles peuvent être écrits en VHDL sous une forme synthétisable.

3.5.1.1 Le modèle VHDL pour représenter la MEF comportementale

La MEF comportementale (BFSM) est le modèle du circuit obtenu juste après la compilation et l'ordonnancement (comme expliqué dans la section 2.3.1). La BFSM est définie en tant que table de transitions, les actions peuvent être des calculs de complexité illimitée et prendre plusieurs cycles d'horloge. Un modèle RTL synthétisable peut être produit à partir de la BFSM si toutes les transitions prennent seulement un cycle d'horloge. La chaîne d'opérations la plus longue d'une transition doit s'exécuter dans une seule période d'horloge.

La description comportementale peut être considérée comme une MEF de très haut niveau, où chaque instruction d'attente correspond à un état. La Figure 15 fait la correspondance entre une description comportementale en VHDL (Figure 15a) et la BFSM produite par MUSIC (Figure 15b). Dans ce cas, les boucles peuvent aussi produire des états dans la BFSM (par exemple, l'état ST_P1_2 dans la Figure 15b). Les actions (expressions, attributions, calculs, etc.) entre deux instructions d'attente consécutives de l'algorithme, sont attribuées à la transition entre les deux états correspondants de la BFSM. En outre, les actions peuvent également inclure des appels de fonction et de procédure. Dans ce cas, le traitement dépendra des attributs spéciaux ajoutés aux déclarations de fonction/procédure en VHDL. Deux choix peuvent être faits : mettre à plat l'appel de fonction/procédure ou le considérer comme une utilisation d'une unité fonctionnelle complexe. Dans ce dernier cas, les paramètres de l'appel de fonction/procédure seront attribués aux ports de l'unité fonctionnelle spécifiés dans la bibliothèque d'unités fonctionnelles.

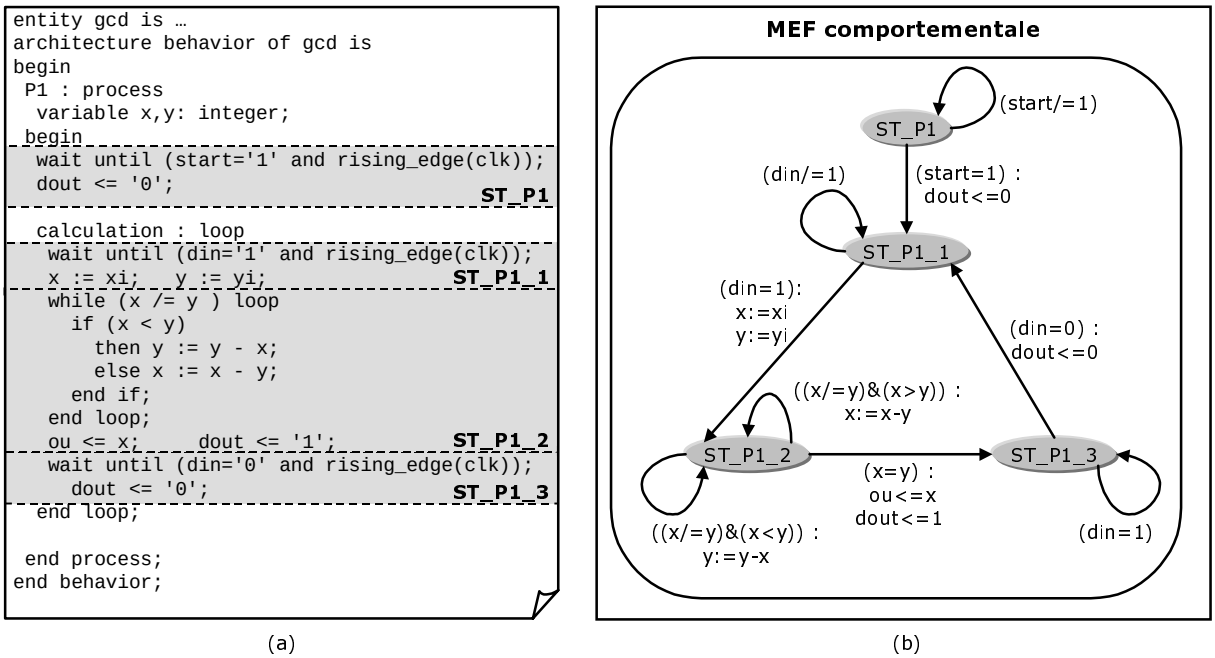


Figure 15 – La description comportementale et le modèle BFSM

Si l'algorithme contient des calculs intensifs entre deux instructions d'attente consécutives, un graphe de flot de données (GFD) sera établi à partir des dépendances de données entre les calculs qui forment le chemin d'exécution entre ces deux points de synchronisation. Ce graphe sera associé à la transition correspondante de la BFSM. Chaque calcul dans ce GFD doit s'exécuter seulement une fois quand la transition associée est activée, puisque le GFD est à l'intérieur d'un graphe de flot de contrôle. Ceci signifie que la sémantique de ces GFDs diffère des sémantiques traditionnelles, qui supposent que tous les calculs sont répétés indéfiniment.

Le modèle VHDL produit pour la BFSM se compose de deux processus concurrents : le processus de transition d'état et le processus de synchronisation (voir Figure 16). Le premier définit les actions à exécuter et l'état suivant pour chaque transition possible. Le second processus tient compte de la synchronisation, en permettant la transition d'un état à l'autre si et seulement si un front montant d'horloge se présente. Il s'occupe également du signal asynchrone de « remise à zéro » (*reset*). Ce signal remet la BFSM au premier état et replace toutes les variables à zéro. Le signal d'horloge et celui de remise à zéro peuvent être identifiés en tant que tels, en mettant des attributs spéciaux dans les ports de l'entité.

```
library IEEE;
 use IEEE.STD_LOGIC_1164.all;
 use IEEE.STD_LOGIC_ARITH.all;
 use IEEE.STD_LOGIC_UNSIGNED.all;
entity GCD is
  port (clk : in std_logic;
        reset : in std_logic;                          Déclaration d'entité
        ou : out std_logic_vector (31 downto 0) );
end GCD;
architecture RTL of GCD is
  type STATE_TYPE is (ST_P1,ST_P1_1,ST_P1_2,ST_P1_3);
  signal CURRENT_STATE, NEXT_STATE : STATE_TYPE;
  signal x, next_x : std_logic_vector (31 downto 0) ;
begin -- of architecture

  P1 : process (gcdstart, din, xi, yi, x, y,CURRENT_STATE)
  begin -- of process
    next_x <= x ;    next_y <= y ;    dout <= '0';
    NEXT_STATE <= ST_P1;
    case CURRENT_STATE is
      when ST_P1 => ...
      when ST_P1_1 => ...
      when ST_P1_2 =>
        if ((x /= y) and (x < y)) then
          next_y <= ( y - x );
          NEXT_STATE <= ST_P1_2 ;                      Processus de
        elsif ((x /= y) and (x >= y)) then             transition d'états
          next_x <= ( x - y );
          NEXT_STATE <= ST_P1_2 ;
        elsif (x = y) then
          ou <= x;
          dout <=  '1';
          NEXT_STATE <= ST_P1_3 ;
        end if;
      when others => null;
    end case;
  end process g;

  SYNCH : process ( clk, reset )
  begin   -- of SYNCH process
    if (reset = '0') then
        CURRENT_STATE <= ST_P1;
-- Resetting variables
    x <= (others => '0') ; ...                         Processus de
    elsif (clk'event and clk = '1') then               synchronisation
        CURRENT_STATE <= NEXT_STATE;
-- Updating variables
    x <= next_x ;    y <= next_y ;
    end if;
  end process SYNCH;
end RTL ; -- of architecture
-- synopsys_synthesis_off
configuration cfg_GCD of GCD is for RTL end for;
end cfg_GCD;
-- synopsys_synthesis_on
```

Figure 16 – Le modèle VHDL pour la MEF comportementale

3.5.1.2 Le modèle VHDL pour la MEF avec ressources

Après l'allocation et l'affectation des unités fonctionnelles, l'endroit où tous les calculs (ou certains d'entre eux) sont exécutés est connu, c.-à-d., le nombre et les types d'unités fonctionnelles sont définis. Les calculs dans les transitions sont remplacés par des transferts de données entre les variables et les ports d'instances des unités fonctionnelles. Par

conséquent, le modèle BFSM est transformé en un modèle appelé MEF avec ressources. Si l'allocation et l'affectation des unités fonctionnelles peuvent être faites sur un ensemble d'opérations choisi, les transitions seront un mélange de calculs et de transferts de données (du type mentionné précédemment). Cette possibilité n'est pas supportée actuellement par MUSIC, pourtant le modèle synthétisable pour la MEF avec ressources présenté ici est toujours applicable. Il est assumé que toutes les transitions s'exécutent dans un seul cycle d'horloge.

Dans la version actuelle de MUSIC, la génération automatique du modèle VHDL pour la MEF avec ressources n'a pas encore été élaborée. Dans cette section, nous discutons d'un modèle VHDL fait manuellement, en supposant que la première soustraction dans la Figure 15a a été remplacée par une division qui s'exécutera dans une unité fonctionnelle appelée FU_DIV. Le modèle VHDL continue à être organisé comme un ensemble de deux processus concurrents : le processus de transition d'état et le processus de synchronisation (voir Figure 17). Néanmoins, trois nouvelles sections sont introduites :

- La déclaration des signaux qui représentent les ports d'unités fonctionnelles : cette section déclare les signaux qui seront reliés aux ports des instances d'unités fonctionnelles ;
- La déclaration des composants : la définition de l'interface des unités fonctionnelles doit être déclarée dans cette section ;
- Et une section avec les instances des composants : les caractéristiques de chaque instance sont déclarées dans cette section et les signaux sont reliés aux ports correspondants.

Comme nous le montre la Figure 17 (dans les lignes en gris), l'opération de division (y <= y/x) a été remplacée par des attributions de signaux. Les variables impliquées dans le calcul lisent ou écrivent les signaux reliés aux ports de FU_DIV. Si l'unité fonctionnelle exige des signaux de contrôle, ceux-là doivent également être insérés dans les transitions appropriées de la MEF. Dans le cas où l'unité fonctionnelle exécuterait des opérations multicycle, cette section va correspondre à un protocole qui comprend plusieurs états.

```
library IEEE…
entity GCD is …
architecture RTL of GCD is
-- Instance FU_00 of FU_DIV.
  signal FU_00_in1 : std_logic_vector(31 downto 0);
  signal FU_00_in2 : std_logic_vector(31 downto 0);
  signal FU_00_out1 : std_logic_vector(31 downto 0);
  component FU_DIV
    generic (Width_in1, Width_in2, Width_out1 : positive := 8);
    port    (in1 :  IN STD_LOGIC_VECTOR (Width_in1-1 downto 0);
             in2 :  IN STD_LOGIC_VECTOR (Width_in2-1 downto 0);
             out1 :  OUT STD_LOGIC_VECTOR (Width_out1-1 downto 0));
  end component;
  type STATE_TYPE is (ST_P1,ST_P1_1,ST_P1_2,ST_P1_3);
  signal CURRENT_STATE, NEXT_STATE : STATE_TYPE;
  signal x, next_x : std_logic_vector (31 downto 0) ;
begin -- of architecture
  I_FU_00  :  FU_DIV
    generic map (Width_in1 =>  32, Width_in2 =>  32, Width_out1 =>  32)
    port map (in1 => FU_00_in1, in2 => FU_00_in2, out1 => FU_00_out1);
  P1 : process (gcdstart, din, xi, yi, x, y,CURRENT_STATE)
  begin -- of process
    next_x <= x ;    next_y <= y ;
    FU_00_in1 <= "00000000000000000000000000000000";
    FU_00_in2 <= "00000000000000000000000000000000";
    NEXT_STATE <= ST_P1;
    case CURRENT_STATE is
      ...
      when ST_P1_2 =>
        if ((x /= y) and (x < y)) then
   -- do : next_y <= ( y / x );
          FU_00_in1 <= y;
          FU_00_in2 <= x;
          next_y <= FU_00_out1;
          NEXT_STATE <= ST_P1_2 ;
        elsif ((x /= y) and (x >= y)) then
          next_x <= ( x - y );
          NEXT_STATE <= ST_P1_2 ;
        elsif (x = y) then
          ou <= x;
          dout <=  '1';
          NEXT_STATE <= ST_P1_3 ;
        end if;
      when others => null;
    end case;
  end process g;
  SYNCH : process ( clk, reset )  ...
  end process SYNCH;
end RTL ; -- of architecture
```

Déclaration des signaux pour les ports de l'instance

Déclaration du composant

Instance

Utilisation de l'unité fonctionnelle FU_DIV

Processus de transition d'états

Processus de synchronisation

Figure 17 – Le modèle VHDL pour la MEF avec ressources

3.5.1.3 Le modèle VHDL pour la MEF avec chemin de données

Le flot de synthèse classique produit une architecture composée d'un contrôleur et un chemin de données. Notre modèle VHDL pour cette architecture est hiérarchique : il y a la description du circuit qui appelle les instances du contrôleur et du chemin de données, décrits dans deux fichiers séparés. La description du circuit (voir Figure 18) et du chemin de données sont des modèles VHDL structuraux composés principalement de quatre sections :

- La déclaration des signaux qui représentent les ports des instances et les fils : cette section déclare les signaux qui seront reliés aux ports des instances et les signaux qui représentent les fils d'interconnexion ;
- La déclaration des composants : les définitions d'interface pour tous les composants référencés dans la description doivent être déclarées dans cette section ;
- Les instances des composants : les caractéristiques de chaque instance sont déclarées dans cette section et leurs ports reliés aux signaux correspondants ;
- Le réseau d'interconnexion des composants : il est décrit explicitement dans cette section.

```
library IEEE;
  use IEEE.STD_LOGIC_1164.all;
entity GCD_circuit is                                          Déclaration d'entité
  port (...
        clk : in std_logic;
        ou : out std_logic_vector(31 downto 0));
end GCD_circuit;
architecture BEHAVIOUR of GCD_circuit is
-- Instance ControlPart of GCD_controller.                     Déclaration des
  signal ControlPart_clk : std_logic; ...                      signaux pour les
-- Instance DataPath of GCD_datapath.                          ports des instances
  signal DataPath_xi : std_logic_vector(31 downto 0);...       et fils de connexion
-- net declarations
  signal Net_net001 : std_logic;...

  component  GCD_controller
    port (...
          yi_SEL :  out  std_logic;
          Mux05_SEL :  out  std_logic);                        Déclaration du contrôleur
  end component ;                                              et du chemin des données
  component  GCD_datapath
    port (...
          dout_SEL :  in  std_logic;
          Mux05_SEL :  in  std_logic);
  end component ;

begin -- of architecture
  I_ControlPart  :  GCD_controller
    port map (...
              ou_SEL => ControlPart_ou_SEL,                    Les instances du
              Mux05_SEL => ControlPart_Mux05_SEL);             contrôleur et du
  I_DataPath  :  GCD_datapath                                  chemin des données
    port map (...
              dout_SEL => DataPath_dout_SEL,
              Mux05_SEL => DataPath_Mux05_SEL);

-- net connections
  DataPath_clk <= Net_net001;
  ControlPart_clk <= Net_net001;
  Net_net001 <= clk;
  DataPath_reset <= Net_net002;                                Le réseau d'interconnexion
  ControlPart_reset <= Net_net002;
  Net_net002 <= reset;
  DataPath_Mux05_SEL <= Net_net022;
  Net_net022 <= ControlPart_Mux05_SEL; ...
end BEHAVIOUR; -- of architecture
```

Figure 18 – Le modèle VHDL du circuit

Le contrôleur est décrit par son comportement, plus précisément, comme une table de transition mise en œuvre par deux processus concurrents. Cependant, la MEF contient des détails explicites concernant l'activation des signaux qui commandent les éléments du chemin de données (voir Figure 19). Les signaux commandent l'écriture de registres, définissent les bits de sélection des multiplexeurs et les bits de contrôle d'unités fonctionnelles.

```
library …
entity GCD_controller is …
architecture RTL of GCD_controller is …
begin -- of architecture
  Controller : process (gcdstart, din, FLAG_x, FLAG_y,…
    case CURRENT_STATE_Controller is
      when ST_P1_Controller =>…
      …
      when ST_P1_2_Controller =>
        if ((FLAG_x /= FLAG_y) and (FLAG_x < FLAG_y)) then
          Mux02_SEL <= '1';
          Mux03_SEL <= '0';
          Cw_y <=  '1';
          Mux05_SEL <= '0';
          sel_FU_00 <= "10";
          NEXT_STATE_Controller <= ST_P1_2_Controller ;
        elsif ((FLAG_x /= FLAG_y) and (FLAG_x >= FLAG_y)) then
          Mux02_SEL <= '0';
          Mux03_SEL <= '1';
          Cw_x <=  '1';
          Mux04_SEL <= '0';
          sel_FU_00 <= "10";
          NEXT_STATE_Controller <= ST_P1_2_Controller ;
        elsif (FLAG_x = FLAG_y) then
          ou_SEL <=  '1';
          dout_SEL <=  '1';
          Mux01_SEL <= '1';
          NEXT_STATE_Controller <= ST_P1_3_Controller ;
        end if;
    end case;
  end process Controller;
  SYNCH : process ( clk, reset )…
end RTL ; -- of architecture
```

Processus de transition d'états

Figure 19 – Le modèle VHDL du contrôleur

3.5.2 Le mécanisme de corrélation

Dans MUSIC, les étapes de synthèse comportementale sont capables de garder la trace des lignes de la description d'entrée qui produisent les nouveaux objets. Par exemple, chaque état de la BFSM garde le numéro de la ligne qui l'a produit (une instruction d'attente ou une condition de boucle). Cette information nous permet de faire la corrélation entre les résultats intermédiaires de synthèse et la description d'entrée. Ce mécanisme s'avère très utile quand un problème apparaît dans un modèle VHDL généré. Les concepteurs peuvent alors facilement tracer le problème dans les lignes correspondantes de la description d'entrée.

La Figure 20 montre une copie d'écran de MUSIC, elle illustre le lien entre le modèle comportemental et le modèle RTL. A l'arrière plan, nous voyons une partie de la fenêtre principale de MUSIC. Du côté gauche, nous pouvons voir la description comportementale d'entré et du côté droit, la description RTL produite juste après l'ordonnancement. Le concepteur peut demander la correspondance entre la description comportementale et RTL. En cliquant sur une ligne comportementale, toutes les lignes RTL correspondantes changent de couleur (lignes indiquées à l'aide des boites et des flèches dans la Figure 20). L'inverse est également possible, c.-à-d., si une ligne RTL est choisie les lignes comportementaux correspondants changent de couleur.

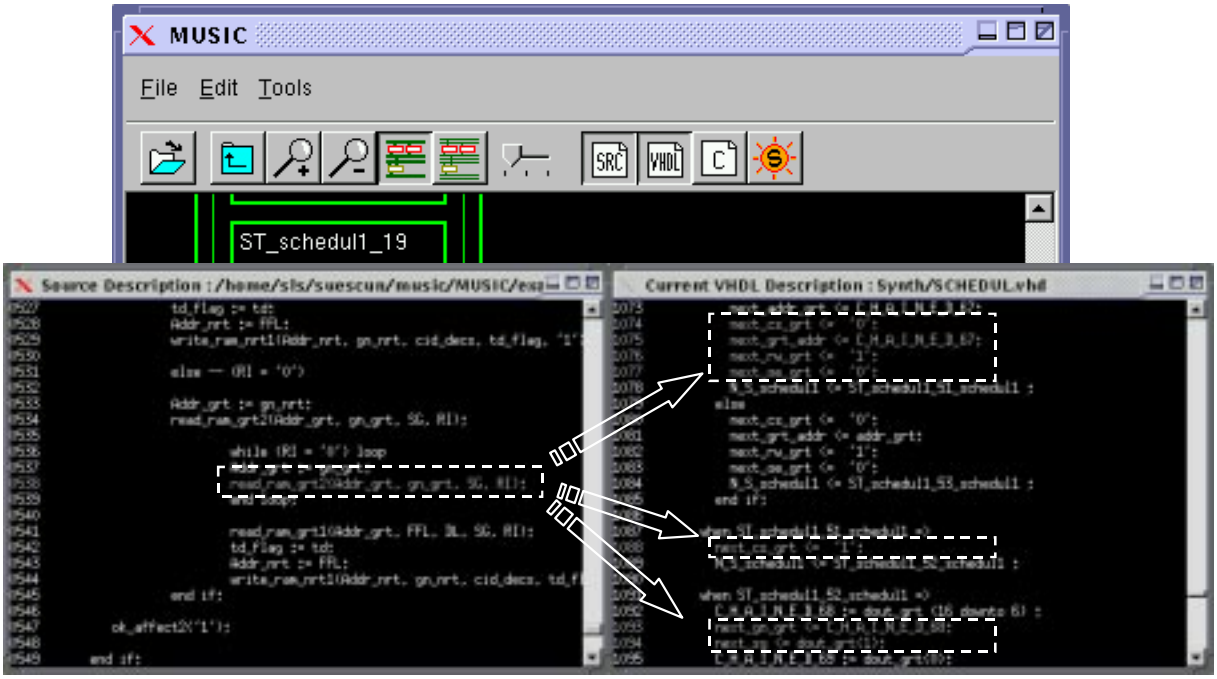


Figure 20 – Le mécanisme de corrélation

3.6 Les résultats

MUSIC peut traiter les applications qui utilisent un style mixte (dominés par le flot de contrôle et de données). Ainsi, l'ensemble de tests qui a été choisi [PaDu95] représente les deux styles :

- Le plus grand diviseur commun (GCD, pour *greatest common divisor*) est un exemple dominé par le flot de contrôle ;

- L'unité à point fixe (FPU, pour *fixed-point unit*) est capable de faire l'addition, la soustraction, la multiplication et la division. Elle peut être classifiée comme une application de type mixte ;
- L'exemple QRS (*Q-rate system*) est une application médicale, il est employé dans un moniteur de fréquence cardiaque et il est dominé par le flot de contrôle ;
- Le filtre elliptique de cinquième ordre (ELLIPF, *fifth order elliptical wave filter*) est un exemple bien connu d'application dominée par le flot de données.

Le Tableau 5 récapitule les résultats obtenus avec la synthèse comportementale de MUSIC pour une technologie CMOS 0,8µm avec deux couches de métal. La synthèse RTL (tâche F) a été réalisée par un outil commercial (*Design Compiler* de **Synopsys**). Les valeurs présentées dans ce tableau proviennent des estimateurs RTL pour plus de précision. Le nombre d'opérations (# op) et le nombre d'états du modèle FSMD donnent une idée de la complexité de l'application. Les valeurs de puissance dynamique ont été calculées pour une fréquence d'horloge de 5 MHz pour le FPU et 10 MHz pour les autres exemples.

Exemple	Flot de conception	# Op	Nombre d'états de la FSMD	Surface (sq.mils)	Chemin critique (ns)	Puissance dynamique (mW)
GCD	SF1	2	4	1319	40,58	0,86
GCD	SF2	2	4	1200	43,11	0,74
FPU	SF1	14	16	10147	103,5	3,48
FPU	SF2	14	16	11473	119,1	2,52
FPU	SF3	14	34	14855	81,21	3,67
QRS	SF1	32	75	7302	49,07	3,48
QRS	SF2	32	75	7166	44,04	2,86
ELLIPF	SF1	34	14	6659	29,00	6,02
ELLIPF	SF2	34	14	7280	31,83	7,51
ELLIPF	SF3	34	17	5865	35,36	5,34

Tableau 5 – Les résultats de synthèse pour différents flots de conception

Pour chaque exemple, nous avons appliqué trois flots de conception (**SF1**, **SF2** et **SF3**). Le premier (SF1) exécute seulement l'ordonnancement (chemin **ABF**, voir la Figure 14) et passe directement à la synthèse RTL. Dans le deuxième flot (SF2), l'allocation et l'affectation des ressources sont faites juste après l'ordonnancement (chemin **ABCEF**). Le troisième flot (SF3) inclut le ré-ordonnancement (chemin **ABDCEF**) après l'ordonnancement.

Pour le GCD et le QRS, le flot SF3 donne les mêmes résultats que le flot SF2, parce qu'ils sont dominés par le flot de contrôle.

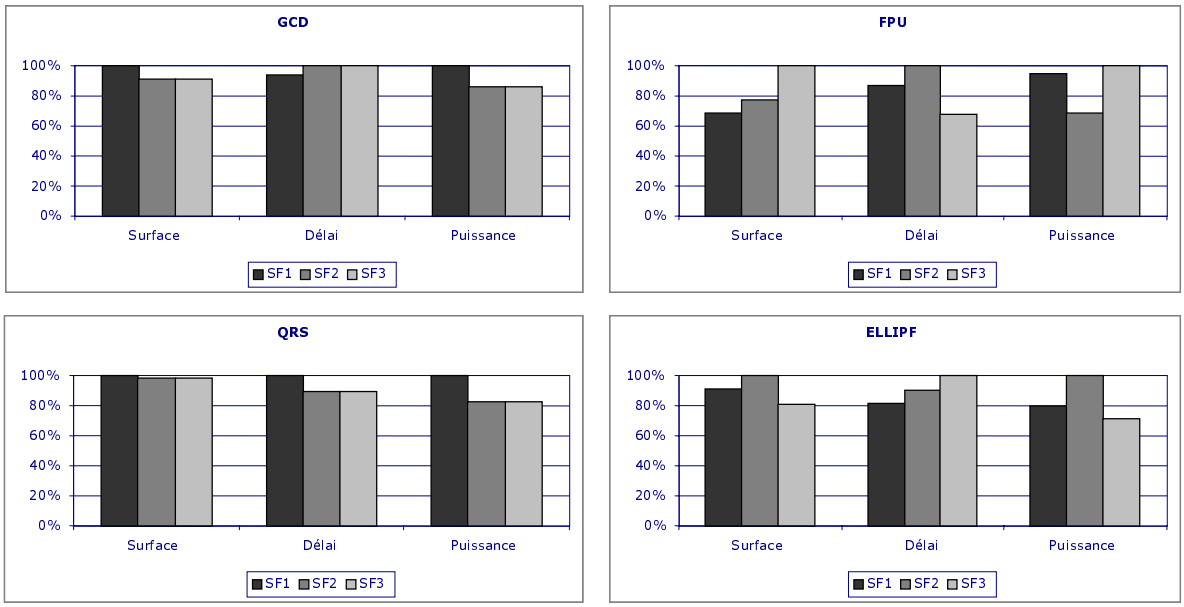


Figure 21 – La comparaison des résultats de synthèse pour différents flots de conception

Dans le cas de l'exemple GCD, le meilleur délai est obtenu avec le flot SF1 tandis que la surface et la puissance dynamique sont plus petites avec le flot SF2 (voir Figure 21). L'allocation et l'affectation réalisées au niveau comportemental mènent à la meilleure solution pour l'exemple QRS (flot SF2).

Le flot de conception SF1 donne la meilleure surface pour l'exemple FPU. En utilisant le flot SF2, nous avons obtenu quelques améliorations quant à la consommation d'énergie. Finalement, le ré-ordonnancement (flot SF3) réduit la longueur d'enchaînement des opérateurs et améliore le délai. Evidement, cette dernière solution emploie plus de surface et le nombre d'états est plus que doublé.

ELLIPF est une application fortement dominée par le flot des données. La solution la plus rapide est obtenue avec le flot SF1. En utilisant le ré-ordonnancement (flot SF3), nous pouvons produire une solution avec un multiplieur et deux additionneurs tandis que les deux autres solutions ont deux multiplieurs et six additionneurs. Cette solution est la meilleure en termes de surface et de puissance.

Il est clair que les résultats de chaque flot dépendent de l'application à synthétiser. Certaines optimisations faites par la synthèse RTL ne sont pas possibles quand elle reçoit des structures préconçues. Dans le cas de nos exemples, aucun des flots utilisés pour nos

expérimentations ne donne le meilleur résultat pour les trois paramètres et pour toutes les applications. Le choix du flot de conception doit être basé sur le style de l'application et la priorité d'optimisation des critères de qualité du circuit.

3.7 Conclusion

Dans ce chapitre, nous avons exploré le chevauchement de la fonctionnalité entre les outils de synthèse comportementale et synthèse RTL. Nous avons démontré qu'il y a plusieurs flots de conception possibles pour combiner la synthèse comportementale et la synthèse RTL. Nos résultats [WOC99] prouvent que le flot de conception qui mène à la meilleure solution n'est pas toujours le même. Il dépend non seulement de la qualité des outils de synthèse utilisés et de la nature de l'application, mais également de la priorité d'optimisation des différents critères de qualité du circuit. La qualité de l'interface entre les outils de synthèse comportementale et RTL influence également le résultat. Il est clair qu'il est difficile d'avoir un flot unique de synthèse, qui soit optimisé pour tous les genres d'applications. Néanmoins, les outils avec un flot flexible de synthèse ont l'avantage d'être facilement adaptables à un plus grand nombre de styles d'applications.

Chapitre 4

Modèle d'Architecture Cible Flexible pour la Synthèse

C'est [...] cette manière d'épauler, de viser, de tirer vite et juste, que je nomme le style.

Cocteau (Jean), Le Secret professionnel (Stock).

Ce chapitre présente une méthode flexible pour l'allocation et l'affectation des interconnexions de la partie opérative du circuit. Cette méthode est basée sur un modèle de chemin de données générique qui permet différents styles d'interconnexions. Nous proposons un modèle d'estimation de la surface de haute fidélité pour guider le concepteur dans le choix du schéma d'interconnexions au niveau comportemental.

4.1 Introduction

Ce chapitre discute des implications d'une architecture cible flexible sur la synthèse comportementale. Les premiers outils de synthèse ont utilisé des composants très simples et un domaine d'application restreint afin de faciliter la résolution du problème. L'utilisation des composants complexes élargit le domaine d'application mais complique la tâche de synthèse. Chacun des trois composants principaux du chemin de données peut avoir un modèle flexible (les unités fonctionnelles, les unités de stockage et les unités de communication).

L'architecture cible du système MUSIC peut contenir de co-processeurs structurés d'une manière hiérarchique. Ces co-processeurs sont des unités fonctionnelles capables d'exécuter des fonctions complexes (ils peuvent avoir leur propre contrôleur interne). [Kiss96] a exploré la flexibilité des unités fonctionnelles comme base pour une méthodologie de conception structurée et hiérarchique. Il y a plusieurs avantages à l'emploi de la conception hiérarchique :

- La méthodologie développée applique le principe de « diviser pour mieux régner » pour traiter les systèmes réels complexes ;
- Elle facilite la réutilisation de blocs existants et la conception orientée à la réutilisation ;
- Elle peut impliquer des gains en productivité et en temps de conception importants (d'un facteur de l'ordre de 3 à 5 selon les applications).

La discussion de ce chapitre se concentrera sur la flexibilité des unités de communication. L'interconnexion est devenue le problème le plus difficile à traiter en ce qui concerne la conception au niveau physique. L'évolution vers les technologies profondément submicroniques (TPS) apporte de nouvelles contraintes relatives aux interconnexions à la synthèse de haut niveau [DaVe97][OtBr98]. Une manière d'adapter facilement l'architecture du circuit à ces nouvelles contraintes, est d'avoir un modèle flexible d'interconnexion. Actuellement, les interconnexions peuvent être réalisées avec des bus et des cellules à trois états ou avec des multiplexeurs. Dans MUSIC, les deux styles d'interconnexion peuvent être employés. La section 4.2 présente l'état de l'art de la synthèse d'interconnexion.

Nous allons étudier l'impact de l'emploi des différentes techniques d'optimisation des interconnexions avec les nouvelles technologies [WOC97] (dans la section 4.3). En outre,

nous discuterons des difficultés pour obtenir un modèle d'estimation précis au niveau comportemental (dans la section 4.4). La section 4.5 présente un modèle de chemin de données générique qui permet différents styles d'interconnexions. Puis, nous présenterons un modèle simple d'estimation de surface au niveau comportemental pour la technologie *standard-cell* (dans la section 4.6). Ce modèle a été validé en comparant les résultats d'estimation au niveau RT (dans la section 4.7). L'utilisation de ce modèle, nous permettra de faire le point des avantages liés à chaque style d'interconnexion disponible dans MUSIC.

4.2 L'état de l'art concernant la synthèse d'interconnexion

La synthèse d'interconnexion est étroitement liée à l'allocation et l'affectation des unités fonctionnelles et des unités de stockage [Gaj92]. La synthèse d'interconnexion vise à transformer les unités abstraites de communication en composants physiques comme les fils, les bus et les multiplexeurs. De nouveaux éléments du chemin de données seront spécialement créés pour réaliser les interconnexions. Même les éléments créés par l'allocation de ressources peuvent être employés avec la fonction de routage s'il n'y a aucun conflit d'utilisation.

La plupart des outils de synthèse comportementale utilisent le modèle d'interconnexion basé sur des multiplexeurs [WalC91]. Ce modèle est facile à tester et il peut bénéficier directement des optimisations réalisées par la synthèse logique. D'autre part, quelques études ont considéré l'emploi des modèles d'interconnexion basée sur des bus [Ewer90]. Peu de travaux ont étudié l'utilisation des modèles mixtes [MoBr96][KuPa90]. Les optimisations des interconnexions sont plus efficaces si elles sont faites en même temps que l'allocation et l'affectation des unités fonctionnelles et de stockage. L'utilisation d'heuristique fait qu'il n'y a aucun modèle d'interconnexion qui est capable de fournir la meilleure solution pour toutes les applications. Ce qui signifie que, l'outil de synthèse doit supporter divers schémas d'interconnexion, afin d'offrir une meilleure exploration de l'espace des solutions.

La plupart des systèmes de synthèse comportementale sont basés sur un modèle d'interconnexion simple. Un modèle d'interconnexion basé sur des bus est employé par [TsSi86]. Son algorithme de synthèse d'interconnexion établit un graphe où chaque nœud représente un raccordement point par point. Il y aura un arc reliant deux nœuds si, et seulement si, les deux raccordements associés ne sont jamais employés simultanément. Après le découpage en cliques, un bus est affecté à chaque clique. Cette approche n'adresse pas la minimisation de cellules à trois états. [ToWi77] décrit une approche qui utilise la

programmation dynamique pour trouver la solution optimale en termes de bus et cellules à trois états. Elle est basée sur une énumération efficace de l'espace des solutions en utilisant un graphe de raccordement auxiliaire. L'information du graphe permet de réduire l'espace de recherche après chaque affectation de bus. Malgré cela, cette solution reste trop onéreuse pour les grandes applications.

Des modèles particuliers composés de segments de bus reliés par des interrupteurs, sont présentés par [Ewer90]. Dans ce cas, la synthèse d'interconnexion doit tenir compte de la position relative des composants du chemin de données. L'affectation des entrées des registres et des unités fonctionnelles peut être employée pour améliorer la solution. **AMICAL** [JDKR97] emploie cette approche. Le chemin de données produit contient un petit nombre de bus mais il emploie un nombre excessif de cellules à trois états.

Une autre possibilité est le modèle mixte, dans ce cas, les interconnexions basées sur des bus et sur des multiplexeurs coexistent. Ce modèle est employé dans MABAL [KuPa90] et *Cathedral-II* [VRBG93]. MABAL emploie une approche constructive du type gloutonne : il commence avec un chemin de données vide et ajoute les unités fonctionnelles, les éléments de stockage et les unités d'interconnexion d'une façon incrémentielle. *Cathedral-II* a un modèle plus restreint d'interconnexion : les interconnexions à l'intérieur des EXUs sont basées sur des multiplexeurs et les raccordements globaux sont basés sur des bus. Il utilise un algorithme itératif qui essaye de réduire le nombre de bus, en fusionnant les bus utilisés sporadiquement avec les bus les plus utilisés. Ces algorithmes manquent de techniques de minimisation de cellules à trois états.

Elf [GiKn84] emploie une technique de type gloutonne de découpage en cliques pour traiter un modèle à plusieurs niveaux de bus. Hormis quelques cas spéciaux, l'emploi des interconnexions à plusieurs niveaux de bus ne présente pas un grand intérêt. Dans la plupart des cas, le délai est plus grand et il y a plus de cellules à trois états que dans l'architecture à un niveau unique de bus. Un modèle d'interconnexion généralisé est présenté par [LEG90], mais les délais des fils ne sont pas explicitement pris en compte pour le calcul du coût d'interconnexion. [KJRD93] a montré comment employer l'information à propos de la disposition des cellules au niveau RT pour tenir compte des délais d'interconnexion. Néanmoins, il n'y a aucun bon estimateur de délai d'interconnexion au niveau comportemental. [MoBr92] a proposé un modèle RC distribué global pour l'évaluation des coûts d'interconnexion. Cependant, il suppose un style spécifique (*bit slice*) de dessin des masques, alors il peut employer des informations géométriques explicites.

La synthèse d'interconnexion dans MUSIC est basée sur un modèle flexible. Il peut générer deux types d'architectures : à base de multiplexeur ou de bus. Les deux modèles d'architectures sont représentées en utilisant un modèle générique de partie opérative. Les cellules d'interconnexion peuvent être de deux sortes : des cellules à trois états et des multiplexeurs. Le choix du modèle architectural le mieux adapté aux besoins de l'application est essentiel pour l'optimisation globale du circuit.

4.3 Les techniques d'optimisation

La disponibilité des multiples couches de routage dans les nouvelles technologies a diminué le coût relatif des fils et des bus en termes de surface. Comme nous avons montré dans la section précédente, l'objectif de la synthèse d'interconnexion est souvent de réduire le nombre de bus. Cette directive d'optimisation ne semble plus raisonnable avec la disponibilité des multiples couches de routage. Il est à noter également que le coût relatif des cellules d'interconnexion augmente (particulièrement sur des technologies *standard-cell*). Ainsi, les nouvelles technologies imposent de nouvelles directives d'optimisation. Nous voulons démontrer que réduire le nombre de cellules d'interconnexion devient plus important que réduire le nombre de bus.

Pour montrer la pénalité de performance provoquée par l'utilisation de techniques d'optimisation précédentes nous avons effectué quelques mesures. Dans cette section, nous comparons les solutions qui ont été obtenues avec une technique qui essaye de réduire le nombre de bus aux solutions qui optimisent le nombre de cellules.

Un groupe de petits exemples d'application a été employé pour comparer les diverses techniques d'optimisation. La complexité relative de ces conceptions peut être évaluée en utilisant le Tableau 6.

Exemple	Nombre de lignes VHDL (comportamental)	Nombre de transferts	Nombre minimum de bus	Nombre de sources de données	Nombre de destinations de données
Gcd	59	12	3	7	6
Pid	163	64	3	17	18
Bubble	87	65	5	13	16
Answer	287	95	4	14	19
Es	57	22	6	14	13
Sqrt	133	44	8	14	15
C	49	24	9	16	18
abr	168	25	7	12	9
fpu	146	82	7	12	13

Tableau 6 – Les caractéristiques des exemples d'applications

Nous avons testé quatre algorithmes pour l'allocation et l'affectation des interconnexions :

1. **BUS** : c'est l'algorithme utilisé par l'outil AMICAL [Park92]. La technique d'optimisation employée est la réduction du nombre de bus. Les bus sont segmentés et le style de dessin de masques ciblé est le *bit-slice*.
2. **ICR** : cet algorithme emploie une heuristique du type gloutonne. La technique d'optimisation employée est la réduction du nombre de cellules à trois états. Les bus ne sont pas segmentés et le nombre de bus de la solution fournie est minimal.
3. **FBS** : cet algorithme emploie une heuristique de fusionnement de bus. La technique d'optimisation employée est la réduction du nombre de cellules à trois états. Les bus ne sont pas segmentés et le nombre de bus de la solution fourni n'est pas minimal. La solution initiale a un bus alloué à chaque source de données.
4. **FBD** : les caractéristiques générales sont les mêmes qui pour l'algorithme FBS. La solution initiale a un bus alloué à chaque destination de données.

Les algorithmes de synthèse d'interconnexion développés (ICR, FBS et FBD) seront expliqués en détail dans la section 5.4.

En dépit de l'emploi d'une technologie *standard-cell* relativement ancienne (CMOS 0,8µm avec deux couches de métal), il est possible de montrer l'impact de chaque technique d'optimisation sur la surface du circuit synthétisé. Les réductions de surface moyennes par rapport à la technique de réduction du nombre de bus (BUS, la pire solution) sont montrées

dans le Tableau 7. Pour l'heuristique du type gloutonne (ICR), le gain moyen de surface a été de 12% par rapport à la solution BUS. L'heuristique de fusionnement de bus qui commence avec un bus par source de données (FBS) a été 13% meilleure que la solution BUS, en moyenne. Pour cette technologie, la meilleure solution est l'heuristique de fusionnement de bus qui commence avec un bus par destination de données (FBD). Dans ce cas, il y avait une grande réduction de la surface de routage (38%) et un gain significatif par rapport à la surface totale (19%).

Technique d'optimisation	Routage	Chemin de données	Contrôleur	Total
BUS	100%	100%	100%	100%
ICR	-24%	-13%	-10%	-12%
FBS	-28%	-14%	-12%	-13%
FBD	-38%	-19%	-16%	-19%

Tableau 7 – La comparaison entre les différentes techniques d'optimisation

Les mauvais résultats de l'algorithme BUS viennent du fait qu'il présuppose un modèle de dessin des masques du type *bit-slice*. Ceci montre clairement l'importance de la technologie cible pour le choix des techniques d'optimisation pour les diverses tâches de synthèse.

4.4 L'évaluation de la qualité du circuit au niveau comportemental

La qualité du circuit synthétisé dépend davantage des décisions prises aux niveaux d'abstraction les plus élevés que des optimisations qui suivent aux niveaux plus bas. Ainsi, il est important de mesurer la qualité du circuit pendant les étapes intermédiaires de la conception. Seulement une bonne évaluation de qualité peut garantir que les bonnes décisions ont été prises aux niveaux plus élevés. Les compromis de conception doivent être faits tôt dans le cycle de synthèse pour éviter les itérations très lentes aux bas niveaux d'abstraction [Berg95]. Pour mesurer la qualité du circuit au niveau comportemental nous devons définir un groupe de métriques de qualité et de modèles d'estimation associés.

4.4.1 Les métriques de qualité

La surface est directement proportionnelle au coût de fabrication du circuit. Le délai définit le cycle d'horloge et il est donc directement lié à la performance du circuit. Pendant longtemps, la surface et le délai étaient les seules métriques de qualité pour évaluer le coût et la performance du circuit. La croissance du marché des dispositifs portatifs actionnés par des batteries a induit l'introduction de la puissance comme métrique de qualité. La puissance est devenue aussi une contrainte physique, parce que la dissipation de la puissance peut limiter la densité du circuit [Keu94].

[Cam96] indique que quelques travaux emploient la mesure de la fiabilité contre l'electromigration comme métrique de qualité. Il est souhaitable d'avoir une fiabilité élevée pour augmenter le temps moyen avant défaut (MTF, pour *mean time between failures*). [DaKa96] montre que la maximisation du MTF entre en conflit avec la minimisation de la puissance, ce qui implique de nouveaux compromis de conception pour les applications concernées.

4.5 Le modèle architectural utilisé par MUSIC

Cette section présente une représentation générique de l'architecture cible utilisée par MUSIC. L'architecture cible est divisée en deux parties : une partie opérative, également appelée partie d'exécution ou chemin de données, et une partie de contrôle ou contrôleur. Le contrôleur se compose d'une MEF qui traite les signaux de contrôle externes et les comptes-rendus du chemin de données et commande la partie opérative par ses signaux de contrôle. En principe, un chemin de données peut être défini par deux aspects : la structure interne et l'organisation fonctionnelle. Le premier aspect correspond aux types d'unités permises et le deuxième aux types de transferts admis, comment ils sont exécutés et comment la communication avec le contrôleur est effectuée.

4.5.1 Le modèle du chemin de données

L'architecture du chemin de données est définie comme un ensemble de composants associés à une topologie d'intercommunication. L'organisation conceptuelle de base de l'architecture est montrée sur la Figure 22.

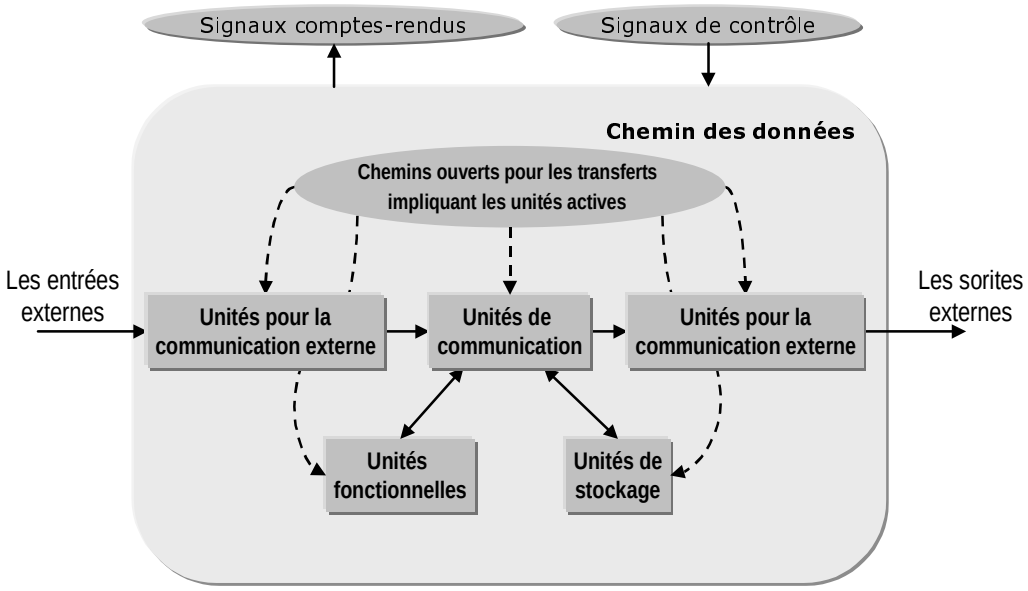


Figure 22 – L'organisation conceptuelle de l'architecture du chemin de données

Des échanges de données entre les composants s'établissent via les unités de communication. Les transferts suivent les chemins ouverts par les signaux de contrôle. En résumé, chaque unité lit les données, les transforme et les transmet ou stocke ces données. Par conséquent, chaque type d'unité se distingue des autres principalement par sa fonctionnalité, alors que le raccordement avec les composants environnants sera toujours fait par un arrangement semblable. Les composants principaux du chemin de données sont [Gaj92][JDKR97] :

1. Les unités fonctionnelles (**FU**s) qui vont exécuter les opérations de la description comportementale.
2. Les unités de stockage (**SU**s), ainsi que les registres, les bancs de registres, RAMs et ROM sont les composants qui peuvent stocker les valeurs des variables ou fournir des constantes spécifiées dans la description comportementale initiale.
3. Les unités pour la communication externe (**ECU**s) lient la partie opérative avec le monde externe.
4. Les unités de communication (**CU**s) ou de transmission sont employées pour contrôler les transferts entre les autres types d'unités, des fils et des bus sont les supports de ces transferts de données. La Figure 23a montre la forme générique d'une unité de transmission. Les types d'unités de communication les plus utilisés sont les cellules à trois états et les multiplexeurs (voir respectivement la Figure 23b et la Figure 23c).

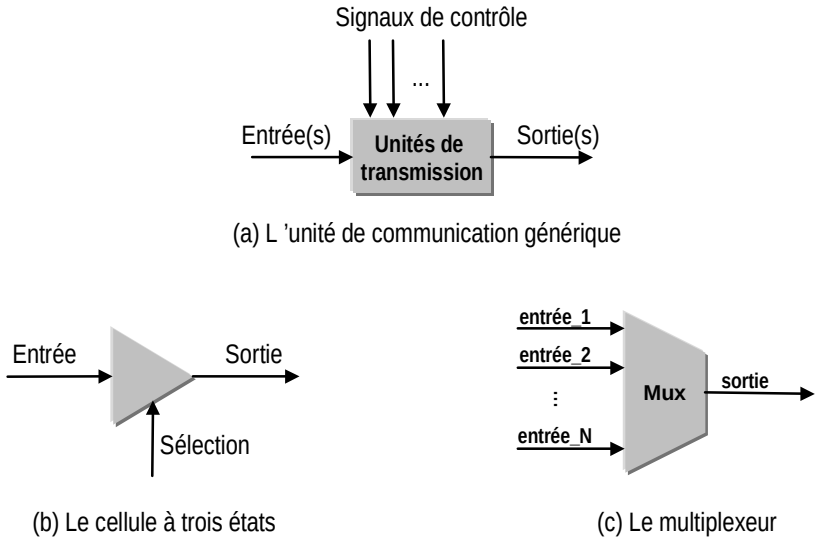


Figure 23 – La forme générique d'une unité de transmission

4.5.2 Le modèle de transfert du chemin de données

A chaque cycle d'horloge, le chemin de données exécute une instruction, elle-même peut être composée d'un ensemble de transferts parallèles. La puissance de calcul d'un chemin de données peut être fixée par le modèle de transfert associé à celui-ci, et par sa capacité de parallélisme [Anc86]. Le « jeu d'instruction » peut être défini par l'ensemble de transferts exécutables par le chemin de données. Par conséquent, en plus de l'information structurelle, la définition du chemin de données doit inclure un modèle de transfert.

Les transferts sont définis par trois composants : la source, la destination et le chemin de données. Pendant un transfert, les données peuvent être échangées de façon séquentielle ou transformées. Cette définition implique un chemin de transfert entre les unités sources et les destinations. Un chemin peut se composer d'autres unités telles que des CUs et des FUs. Un transfert peut donc être représenté comme un graphe, où les nœuds sont les composants du chemin de données, et les arcs les liens physiques entre eux. Chaque transfert peut alors être décomposé en plusieurs échanges de données atomiques à partir de l'unité de source à l'unité de destination. Nous définissons la source comme la première unité impliquée dans le transfert après le début du cycle d'horloge ; et l'unité destination comme la dernière unité atteinte avant la fin du dernier cycle d'horloge du transfert. Si nous considérons les composants définis précédemment, la forme générale d'un transfert est la suivante :

TRANSFERT

Source ⇒ *Chemin* ⇒ *Destination*

[***ECU***, ***SU***, ***FU***] → ***CU*** → { [***FU***, ***CU***] → } [***ECU***, ***SU***, ***FU***]

où : [a,b]: signifie « choisis a ou b »
{a}: signifie quelque répétition de 'a'

Il est évident que ce modèle peut s'appliquer aux schémas d'interconnexion basés sur des bus et sur des multiplexeurs. Il est facile d'étendre ce modèle à d'autres schémas existants, tels que le schéma à plusieurs niveaux de multiplexeurs ou le schéma qui mélange des bus et des multiplexeurs. Dans la section suivante, nous considérerons les modèles les plus utilisés : celui basé sur des bus et celui basé sur des multiplexeurs.

4.5.3 Les modèles d'interconnexion

Pour le modèle d'interconnexion basé sur des bus, les raccordements internes entre les unités se composent de bus et de cellules à trois états. Admettons l'utilisation des commutateurs unidirectionnels seulement, quelques transferts de base permis par le modèle d'interconnexion sont détaillés sur la Figure 24. Dans cette figure, **[Sw]** représente un commutateur facultatif. Des transferts complexes peuvent être réalisés en combinant plusieurs transferts de base (voir Figure 24b). Un modèle de transfert pour les parties opératives basées sur des multiplexeurs est montré dans la Figure 25.

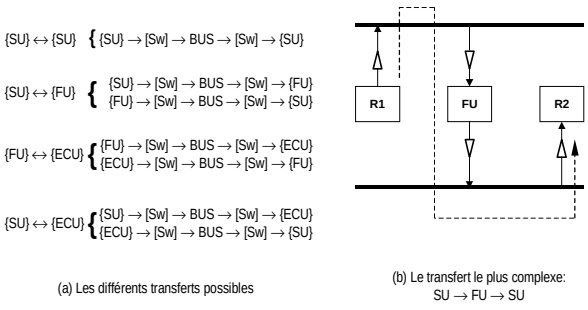


Figure 24 – Le modèle de transfert basé sur des bus

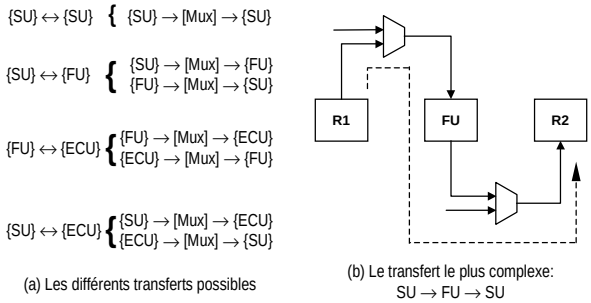


Figure 25 – Le modèle de transfert basé sur des multiplexeurs

Les unités de communication sont des représentations abstraites des composants physiques. D'un point de vue conceptuel, l'unité de communication est un objet qui peut exécuter un transfert de données. Cet objet a un ensemble de propriétés : le nombre de bits de ses entrées et sorties, le délai de transmission, la disponibilité lors de chaque étape de contrôle, etc. La synthèse d'interconnexion peut être traitée d'une manière plus générale quand les détails physiques des unités de communication sont abstraits. En outre, cette

abstraction fournit plus de flexibilité quant au type d'éléments de raccordement qui peuvent être employés par l'outil de synthèse. Ce modèle généralisé peut être adapté à différents schémas d'interconnexion : le schéma mixte bus et multiplexeur, le schéma utilisant des chemins de largeur variable et le schéma utilisant des chemins internes aux unités fonctionnelles.

A partir de ces transferts simples, nous pouvons constituer des expressions complexes pour modéliser le chaînage des opérations.

4.6 Le modèle d'estimation de la surface d'interconnexion

De bons modèles d'estimation sont essentiels parce qu'ils guident le concepteur pendant l'exploration de l'espace des solutions. MUSIC emploie une architecture cible flexible, ce qui impose une méthodologie d'estimation associée à chaque modèle d'interconnexion. Cette section présente un modèle d'estimation au niveau comportemental pour la surface d'interconnexion.

Nous n'avons pas développé la métrique pour les délais faute de bons modèles du dessin des masques. La connaissance de la géométrie exacte des fils est cruciale pour obtenir un bon modèle d'estimation de délai [KuGaj93][McFar86]. Par exemple, [RaKur94b] emploie un modèle de prédiction basé sur le dessin des masques qui prend en compte les effets de câblage. [Najm98] présente un modèle d'estimation de délai indépendant du floorplan, il est basé sur un nouveau concept appelé mesure de délai (pour, *delay measure*) d'une fonction booléenne. Cependant, leur approche exige une caractérisation précise de la technologie cible par rapport au timing. Cette caractérisation doit être faite par des exécutions extensives de la synthèse logique. En outre, la méthode s'applique seulement aux parties qui peuvent être décrites par des équations booléennes. Une approche complètement indépendante de la technologie est montrée par [LaPot98]. L'inconvénient ici est le fait que plusieurs étapes d'optimisations indépendantes de technologie doivent être exécutées pour obtenir les estimations, pénalisant ainsi l'efficacité d'exécution.

Nous avons voulu rester indépendants de la synthèse physique, ainsi nous avons focalisé notre analyse à une seule métrique de qualité : la surface. Les modèles d'estimation de surface pour le contrôleur et le chemin de données sont détaillés dans les prochaines sections.

4.6.1 Le modèle d'estimation de surface du contrôleur

Au niveau comportemental, le contrôleur est habituellement modélisé comme une MEF avec un codage symbolique des états. La synthèse du contrôleur est répartie en trois tâches : le codage des états, la synthèse logique et la transposition technologique. Le codage d'état assigne des valeurs binaires aux noms symboliques des états [Dev87]. Après le codage, nous obtenons une table d'état qui est traitée comme table de vérité par les techniques d'optimisation logique [Bray87]. Puis, la transposition technologique produit une réalisation optimisée pour la bibliothèque cible dépendant de la technologie.

Le défi de faire un bon modèle d'estimation pour le contrôleur est de prévoir résultat des optimisations faites pendant la synthèse logique. [Some92] présente une technique d'estimation de surface indépendante de la technologie, mais elle se base sur le résultat de la synthèse logique, c.-à-d., le réseau des cellules logiques ou des blocs interconnectés. [RaKur94] propose une approche du type gloutonne : il suppose un ordonnancement fixe et permet certaines transformations de ré-affectation sur le chemin de données qui changent seulement une partie de la logique du contrôleur. La synthèse du contrôleur est faite une seule fois, par la suite une heuristique rapide est employée pour la synthèse logique et la transposition technologique seulement dans la partie modifiée à chaque fois. Notre approche utilise la méthode présentée par [VaGaj95]. L'estimation de la surface du contrôleur peut être obtenue en fonction de trois paramètres de conception :

- Le nombre d'états de contrôle possibles ;
- Le nombre de lignes de contrôle entre le contrôleur et le chemin de données ;
- Et un tableau, où chaque ligne de contrôle est associée au nombre d'états pour lesquels la ligne doit être activée.

Notre modèle d'estimation de surface du contrôleur peut être exprimé par l'équation (4-1).

$$S_C = T \cdot (k_1 \cdot CL + k_2 \cdot \log_2(S)) \qquad (4\text{-}1)$$

où : *T est le nombre de transitions d'état ;*
CL est le nombre de lignes de contrôle entre le contrôleur et le chemin de données ;
S est le nombre d'états ;
k_1 et k_2 sont les facteurs de correction.

Les facteurs k_1 et k_2 permettent d'adapter le modèle d'estimation à différentes technologies et outils de synthèse du contrôleur (il faut prendre en compte les optimisations faites pendant la synthèse logique). Ces facteurs peuvent être calculés par des méthodes statistiques. Les architectures basées sur des multiplexeurs ont l'avantage théorique d'employer moins de lignes de contrôle entre le contrôleur et le chemin de données. Dans la pratique, cet avantage n'a pas une grande influence sur la surface totale du circuit.

4.6.2 Le modèle d'estimation de surface du chemin de données

La surface du chemin de données est généralement divisée en deux parties par les algorithmes d'estimation : la surface active et la surface de câblage. La première partie correspond à la surface des cellules des unités fonctionnelles (FUs), des unités de stockage (SUs), des unités externes de communication (ECUs) et des unités de communication (CUs). La surface de câblage correspond à l'espace occupé par les fils de routage. Nos expériences [WOC97] avec des technologies *standard-cell* ont montré que la surface de câblage est approximativement un facteur constant (le facteur de routage) de la surface active. La valeur de ce facteur dépend de la technologie utilisée et du modèle de routage du dessin des masques. Les résultats expérimentaux employant une technologie *standard-cell* CMOS 0,8-μm avec deux couches de métal, ont montré qu'environ 70% de la surface totale du circuit correspond à la surface de câblage. Evidemment, les technologies qui utilisent plusieurs couches de routage réduisent considérablement les coûts de routage. Le facteur de routage de ces technologies peut même être égal à zéro, puisque tout le routage peut être fait au-dessus des cellules. De ces données, l'estimation de surface du circuit peut être déduite en utilisant l'équation (4-2).

$$S_T = \frac{(A_D + A_C)}{1-\alpha} \tag{4-2}$$

où : S_T *est la surface totale du circuit*
A_D *est la surface des cellules du chemin de données*
A_C *est la surface des cellules du contrôleur*
α *est le facteur de routage*

A partir de l'information présente dans la bibliothèque de composants, il est facile d'obtenir une estimation précise de la surface active du chemin de données en ajoutant les valeurs de la surface de chaque cellule, comme montré par l'équation (4-3).

$$A_D = \sum_i A_{FU}(i) + \sum_j A_{SU}(j) + \sum_k A_{ECU}(k) + \sum_l A_{CU}(l) \tag{4-3}$$

où : A_D *est la surface des cellules du chemin de données*
$A_{FU}(i)$ *est la surface des cellules des unités fonctionnelles*
$A_{SU}(j)$ *est la surface des cellules des unités de stockage*
$A_{ECU}(k)$ *est la surface des cellules des unités de communication externe*
$A_{CU}(l)$ *est la surface des cellules des unités de communication*

La technique d'estimation montrée ici a été employée dans MUSIC pour choisir le meilleur modèle d'interconnexion pour chaque application. Dans la section 4.5 nous avons présenté un modèle générique de chemin de données capable de représenter une variété de schémas d'interconnexions. L'efficacité de chaque schéma dépend de l'application synthétisée. En général, des interconnexions basées sur des multiplexeurs sont plus efficaces pour les applications avec un facteur de partage des ressources bas. D'autre part, les applications qui utilisent un grand contrôleur ou un grand jeu d'instructions pour permettre des facteurs de partage de ressources élevés, donnent de meilleurs résultats avec des interconnexions basées sur bus. Nos expériences ont montré que la comparaison employant un modèle d'estimation de niveau comportemental est suffisant pour mesurer la qualité d'une architecture donnée et établir une classification qualitative des solutions [WOC97].

Ce qui fait la différence entre les modèles d'interconnexion basés sur des multiplexeurs et basés sur des bus, c'est la surface des unités de communication (CUs) et la surface de câblage. Les performances (surface) des multiplexeurs et des cellules à trois états sont montrées dans la Figure 26 (technologie *standard-cell* CMOS 0,8-μm avec deux couches de métal). La taille d'un multiplexeur avec deux entrées est plus petite que l'ensemble de deux cellules à trois états pour cette technologie. Cependant, quand il y a plus de cinq entrées cette relation est inversée, c.-à-d., l'utilisation de cellules à trois états devient plus

avantageuse. Dans notre modèle d'estimation, une équation de régression linéaire (comme celle représentée sur la Figure 26) est employée pour calculer la surface de cellules des CUs.

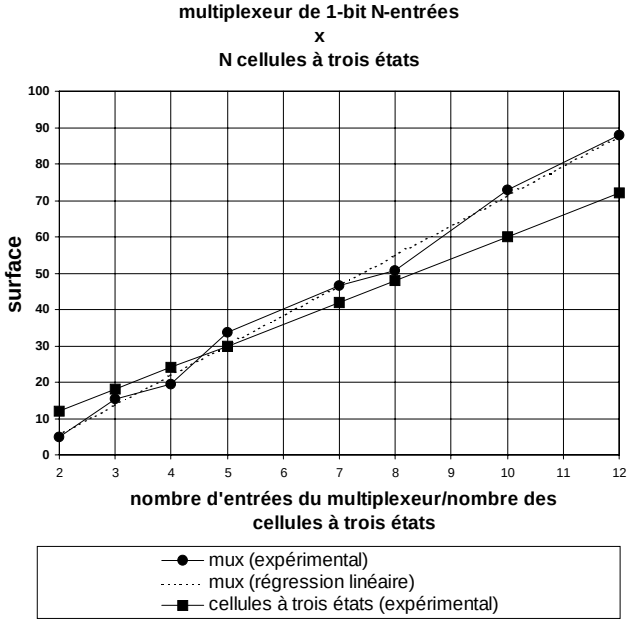


Figure 26 – La comparaison entre les multiplexeurs et les cellules à trois états

4.7 La comparaison entre les modèles d'interconnexion

Dans cette section, les interconnexions basées sur des bus et sur des multiplexeurs sont évaluées pour déterminer quelle solution est la plus appropriée à chaque style d'application. Le modèle d'estimation présenté dans la section précédente est validé par comparaison avec l'estimation au niveau RT. Les données présentées ont été obtenues avec la synthèse comportementale de MUSIC et un outil commercial de synthèse RTL (*Design Compiler* de **Synopsys**) pour une technologie *standard-cell* CMOS 0,8-µm avec deux couches de métal.

Trois petits exemples d'application et trois études de cas ont été synthétisés. Le Tableau 8 donne les résultats de la synthèse RTL pour ces exemples synthétisés avec les deux modèles d'interconnexion. Les exemples d'application sont :

- Le plus grand diviseur commun (GCD, pour *greatest common divisor*) ;
- Une unité de calcul de point fixe (FPU, pour *fixed-point unit*) qui fait l'addition, la soustraction, la multiplication et la division ;
- Et l'algorithme de calcul de tri à bulles (*Bubble sort*).

Les études de cas s'appellent **RegN** (**Reg5**, **Reg6**, et **Reg11**), ce qui signifie qu'ils ont **N** registres et chaque registre échange des données avec tous les autres. En conséquence, chaque valeur des registres peut être transférée à tous les autres registres. Lors de la réalisation avec des interconnexions basées sur des multiplexeurs, chaque registre aura un multiplexeur de (**N**-1) entrées (4, 5 et 10 respectivement) connecté à son entrée. Cette structure simule des applications comme les contrôleurs complexes et les microprocesseurs avec un grand jeu d'instructions. En fait, il arrive souvent le cas que les registres soient normalisés dans ce genre d'application.

Le nombre de cellules à trois états, le nombre de multiplexeurs à deux entrées et le nombre de fils (*nets*) donnent une idée des complexités d'interconnexion requises pour chaque application. La surface de cellules et la vitesse (chemin critique) viennent des informations structurelles obtenues après la synthèse RTL.

Exemple	Solution basée sur des multiplexeurs				Solution basée sur des bus				Surface des cellules mux / bus	Délai mux / bus
	# mux à deux entrées	# fils	Chemin critique (ns)	Surface des cellules	# cellules à trois états	# fils	Chemin critique (ns)	Surface des cellules		
GCD	5	125	9,17	220	12	154	8,34	340	65%	110%
Bubble	21	234	14,73	538	25	254	14,72	811	66%	100%
FPU	25	801	43,42	2329	26	876	31,77	3111	75%	137%
Reg5	22	147	8,54	438	9	121	4,64	405	108%	184%
Reg6	31	172	9,93	574	10	131	4,99	454	126%	199%
Reg11	100	283	12,18	1530	15	181	6,76	698	219%	180%

Tableau 8 – Les résultats de la synthèse RTL pour le chemin de données

A partir du Tableau 8, il est facile de voir qu'il n'y a aucun modèle d'interconnexion approprié pour toutes les applications. Le modèle basé sur des multiplexeurs donne les meilleurs résultats pour les deux premiers exemples. Tandis que le modèle basé sur des bus est meilleur pour les trois derniers exemples. En général, les chemins de données avec un facteur de partage de ressources élevé sont plus appropriés au modèle d'interconnexion basé sur des bus. D'autre part, des applications telles que le GCD, le FPU et le BUBBLE qui ont un facteur de partage de ressources bas, donnent de meilleurs résultats avec le modèle basé sur des multiplexeurs. Pour le FPU, la solution basée sur des multiplexeurs donne un meilleur résultat en termes de surface, tandis que la solution basée sur des bus fournit une meilleure solution en termes de délai.

La comparaison des résultats de notre outil d'estimation aux résultats de la synthèse RTL a deux objectifs : mesurer la fiabilité de nos méthodes d'estimation et étudier la possibilité de les employer pour l'exploration de l'espace des solutions. Les résultats sont résumés dans le Tableau 9. Pour chaque exemple, nous présentons l'estimation de la surface : du chemin de données, du contrôleur et du routage. Nous montrons les résultats produits par la synthèse RTL et par notre outil d'estimation. La fidélité de l'estimation calculée avec la mesure de fidélité présentée par [GaVa94] est de 100% pour la surface des cellules du chemin de données et de 97% pour la surface totale.

Exemple	Solution basée sur des multiplexeurs						Solution basée sur des bus						L'estimation comportamentale	
	RTL				comportamental		RTL				comportamental			
	Chemin de données (% total)	Contrôleur (% total)	Routage (% total)	total	Chemin de données (% total)	total	Chemin de données (% total)	Contrôleur (% total)	Routage (% total)	total	Chemin de données (% total)	total	mux / bus (chemin de données)	mux / bus (total)
GCD	220 (19%)	93 (8%)	829 (73%)	1142	196 (23%)	842	340 (21%)	101 (6%)	1151 (73%)	1592	310 (24%)	1268	63%	72%
Bubble	538 (17%)	313 (11%)	2233 (72%)	3084	499 (17%)	3012	811 (21%)	303 (8%)	2732 (71%)	3846	728 (18%)	3954	69%	79%
FPU	2329 (24%)	218 (3%)	7015 (73%)	9562	2223 (26%)	8639	3111 (26%)	245 (2%)	8758 (72%)	12114	3042 (26%)	11640	73%	75%
Reg5	438 (22%)	112 (6%)	1415 (72%)	1965	419 (22%)	1868	405 (24%)	114 (7%)	1178 (69%)	1697	395 (22%)	1763	106%	106%
Reg6	574 (23%)	116 (4%)	1834 (73%)	2524	523 (21%)	2508	454 (24%)	125 (7%)	1299 (69%)	1878	443 (21%)	2160	118%	120%
Reg11	1530 (21%)	369 (5%)	5267 (74%)	7166	1251 (16%)	7915	698 (21%)	339 (10%)	2330 (69%)	3367	686 (15%)	4635	182%	167%

Tableau 9 – Les résultats des estimations de surface

Le Tableau 9 montre deux résultats importants de ce travail :

- L'estimation de la surface des cellules du chemin de données est tout à fait précise par rapport à l'estimation au niveau RT. La différence entre la surface estimée au niveau comportemental et au niveau RT est d'environ 10%.
- Le modèle d'estimation développé peut être employé comme un critère objectif pour choisir les solutions architecturales pendant la synthèse comportementale. Les deux dernières colonnes dans le Tableau 9 montrent le rapport entre les évaluations au niveau comportemental des solutions basées sur des multiplexeurs et ceux basées sur des bus. Ils sont très près du rapport calculé après la synthèse

RTL (dernière colonne du Tableau 8). La constatation que la surface des cellules du chemin de données soit suffisante pour la classification qualitative du résultat de la synthèse comportementale, peut être expliqué par deux faits. D'abord, la surface de routage semble être un facteur constant par rapport à la surface du circuit entier (aux alentours de 70%). En second lieu, la taille du contrôleur est presque la même pour les deux modèles d'interconnexion utilisés.

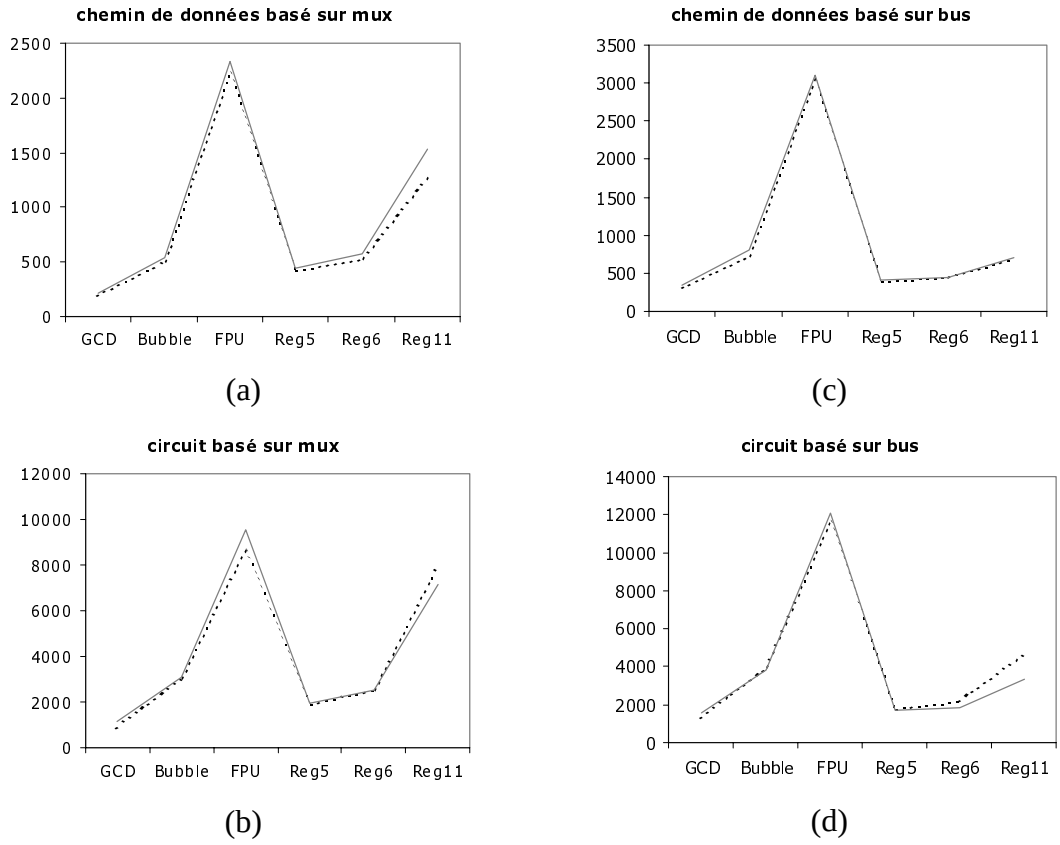


Figure 27 – La fidélité des estimations
(--- l'estimation comportamentale, — l'estimation RTL)

La Figure 27 montre la haute fidélité de notre modèle d'estimation de surface des cellules. Dans le cas de l'estimation de la surface du chemin de données pour les architectures basées sur des multiplexeurs (Figure 27a) et basées sur bus (Figure 27c), la fidélité était 100%. En estimant la surface totale, la fidélité était de 100% pour l'architecture basée sur des multiplexeurs (Figure 27b) mais seulement de 93% pour l'architecture basée sur des bus (Figure 27d). Dans ce cas, l'estimation de la surface totale a été compromise par l'inexactitude de l'estimation de la surface du contrôleur pour l'exemple Reg11. La fidélité totale, si nous prenons en considération toutes les estimations, était de 99%. Ceci montre que notre modèle d'estimation de niveau

comportemental est suffisamment précis pour être employé dans l'exploration de l'espace des solutions.

4.8 Conclusion

Ce chapitre a présenté une étude comparative des modèles d'interconnexion basés sur des multiplexeurs et sur des bus. MUSIC utilise le modèle générique du chemin de données présenté dans la section 4.5. Ce modèle a été conçu pour représenter un grand sous-ensemble de modèles architecturaux. Il est clair que le choix du modèle d'interconnexion influencera considérablement l'efficacité de la solution générée. Notre méthodologie permet de décider au niveau comportemental, quel est le meilleur modèle à employer pour chaque application. Cette possibilité est essentielle pour la synthèse des grands circuits. En fait, le temps d'exécution de la synthèse RTL peut être un facteur important de la durée totale de la conception.

Cette étude a prouvé que l'efficacité de chaque modèle d'interconnexion dépend des caractéristiques de l'application. Pour les grands chemins de données avec de petits contrôleurs, la solution basée sur des multiplexeurs semble être la plus efficace. D'autre part, quand le contrôleur inclut un grand jeu d'instructions, le modèle basé sur des bus semble produire la meilleure solution.

Chapitre 5

Les Outils pour la Synthèse Flexible

Créer est aussi difficile que d'être libre.

Triolet (Elsa), La Mise en mots (Skira).

Ce chapitre présente les outils développés pour mettre en œuvre le flot flexible pour la synthèse architecturale présenté dans le troisième chapitre. Trois outils ont été développés : un outil de ré-ordonnancement, un outil d'allocation et d'affectation des unités fonctionnelles et un outil d'allocation et d'affectation des interconnexions et de génération d'architecture.

5.1 Introduction

Dans le troisième chapitre, nous avons montré les avantages du flot flexible de conception et de l'interface flexible avec des outils de synthèse RTL. Le flot flexible de la synthèse comportementale décrit dans la section 3.4, a été mis en application et testé dans l'outil de synthèse au niveau système MUSIC. Ce flot flexible de conception doit fournir une description VHDL synthétisable après chaque tâche de synthèse. En outre, l'ordre de l'exécution des différentes tâches de conception n'est pas prédéfini.

MUSIC est une évolution de l'outil de synthèse comportementale itérative AMICAL [Park92]. Ce dernier emploie un ordre fixe des tâches de synthèse : macro-ordonnancement, allocation et affection des unités fonctionnelles, micro-ordonnancement suivi de la génération et la personnalisation de l'architecture. Nous avons dû développer de nouveaux outils de synthèse afin de supporter la méthodologie de synthèse flexible utilisée par MUSIC.

Puisque l'ordre de l'exécution des outils de synthèse comportementale n'est pas connu d'avance, il était fondamental d'augmenter la modularité de ces outils. Cette modularité permet une certaine indépendance entre les divers outils, ce qui est essentiel pour notre méthodologie de synthèse. Notre approche pour augmenter la modularité des outils de synthèse suit deux directions principales :

- L'utilisation de SOLAR [JeOB94] comme modèle de représentation unique (comme est expliqué dans la section 2.2.2) ;
- Le modèle d'entrée du circuit pour chaque outil est flexible, de façon à s'adapter aux différents flots de synthèse possibles.

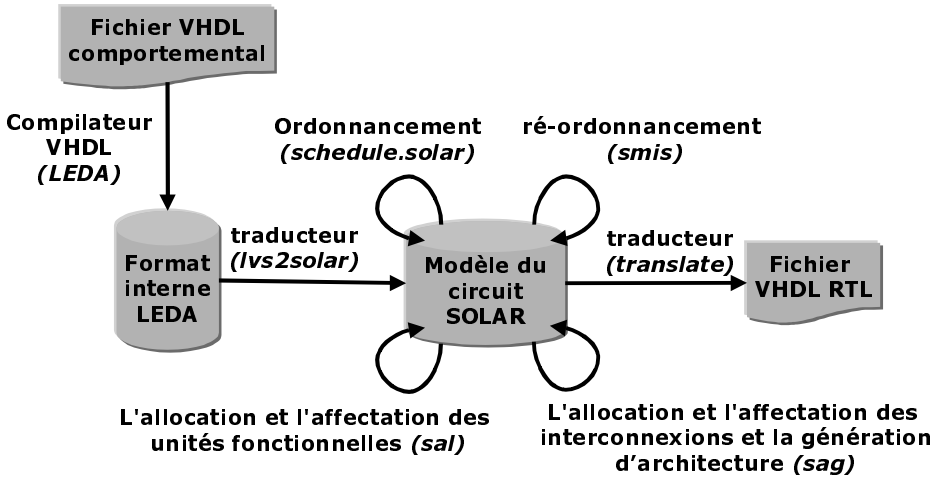


Figure 28 – Le modèle SOLAR et les outils de synthèse de MUSIC

Dans MUSIC, la spécification au niveau comportemental est un fichier en VHDL standard qui respecte les restrictions discutées dans la section 2.3.1.1. Les spécifications sont analysées en utilisant le compilateur VHDL de **LEDA**. Ensuite, le format interne LEDA est traduit en format SOLAR avec l'outil ***lvs2solar***. A partir de là, tous les modèles du circuit auront leur représentation SOLAR et une représentation équivalente en VHDL. Le modèle VHDL est généré par l'outil ***translate***. La synthèse comportementale de MUSIC se compose de quatre outils de synthèse :

- L'outil d'ordonnancement : ***schedule.solar*** ;
- L'outil d'allocation et d'affectation des unités fonctionnelles : ***sal*** ;
- L'outil de ré-ordonnancement : ***smis*** ;
- L'outil d'allocation et d'affectation des interconnexions et de génération de l'architecture : ***sag***.

Si l'entrée VHDL respecte les restrictions présentées dans la section 2.3.1.1, la conversion de VHDL en SOLAR est directe. Le processus VHDL qui sera synthétisé est converti dans une unité de conception (*Design unit*) SOLAR décrite au niveau comportemental. Les déclarations de fonction et de procédure VHDL, les définitions des variables et des constantes, les structures de contrôle et de décision, des expressions, les appels de fonction et de procédure ont des représentations sémantiquement équivalentes dans SOLAR.

L'ordonnancement est réalisé par l'algorithme d'ordonnancement à boucles dynamiques (DLS) [ROA94] et emploie l'approche basée sur des chemins proposée par [Cam91] et [Fis81]. DLS transforme la description comportementale d'entrée en une BFSM de type Mealy. Un état est introduit à chaque instruction d'attente (*wait*) de la description VHDL. La section 2.3.1.2 a expliqué l'étape d'ordonnancement.

Les outils développés dans le contexte de cette thèse seront expliqués plus en détail dans les prochaines sections. La section 5.2 présente l'algorithme pour l'allocation et l'affectation des unités fonctionnelles. La section 5.3 discute du ré-ordonnancement, les trois heuristiques développées pour l'allocation et l'affectation d'interconnexion sont montrées dans la section 5.4. En conclusion, la section 5.5 présente le procédé de génération d'architecture.

5.2 L'allocation et l'affectation des unités fonctionnelles

L'outil d'allocation et d'affectation des unités fonctionnelles (***sal***) définit les types et le nombre d'unités fonctionnelles pour le chemin des données. Dans MUSIC, cette tâche peut être exécutée après l'ordonnancement ou après le ré-ordonnancement, et elle traite toutes les opérations de la BFSM d'entrée. Deux types de modifications peuvent être effectués dans le modèle du circuit : l'introduction des instances des unités fonctionnelles et la traduction de chaque action de la BFSM, qui référence une unité fonctionnelle, en une liste d'assignations de signaux aux ports des unités fonctionnelles. L'algorithme employé dans ***sal*** prend en compte à la fois des considérations locales et globales pour optimiser une fonction de coût qui dépend de trois métriques de qualité : coûts de surface, de délai et d'interconnexion. Cet algorithme est basé sur l'algorithme qui définit la correspondance pour un graphe bipartite pondéré (BWGM, pour *bipartite weighted graph matching*) présenté par [Hsu90].

5.2.1 La bibliothèque des unités fonctionnelles

Les informations qui décrivent les interfaces des unités fonctionnelles doivent être placées dans un fichier SOLAR (appelé FULIB dorénavant). La syntaxe de ce fichier est décrite en détail dans l'annexe **A**. Les détails de réalisation des unités fonctionnelles ne sont pas importants pour les algorithmes de synthèse. L'interface d'accès et quelques propriétés de mise en œuvre (surface et délai) nous suffisent. Cette caractéristique facilite la réutilisation des composants synthétisés avec MUSIC ou avec n'importe quel autre outil de conception.

Le fichier FULIB doit contenir la liste complète des unités fonctionnelles qui peuvent être employées par les algorithmes de synthèse. Les seuls opérateurs implicites sont le SLICE (pour choisir quelques bits d'un signal) et CONCAT (pour rassembler des tranches de bits dans un nouveau signal) parce que nous supposons qu'ils sont réalisés avec des fils et ne prennent pas de temps d'exécution. Une unité fonctionnelle peut réaliser de multiples fonctions (*methods*, en syntaxe SOLAR) ; chacune peut avoir son propre délai d'exécution et son interface, mais la surface est une propriété globale. Les fonctions sont considérées comme des unités logiquement séparées pour la description dans le fichier FULIB, c.-à-d., leur description ne dépend pas de leur réalisation physique. Chaque fonction a son propre ensemble de ports d'entrée et de sortie et une table d'états qui décrit le protocole d'interface pour exécuter cette fonction.

Un exemple est donné sur la Figure 29, l'unité fonctionnelle ***FU1*** peut exécuter deux fonctions : FCT1 et FCT2. L'interface avec le monde externe est établie avec quatre paramètres : (**a,b**) pour l'entrée, et (**c,d**) pour la sortie. L'unité fonctionnelle peut avoir deux propriétés : le temps d'exécution maximal et la surface. Chaque état dans la table d'états représente une étape de contrôle pour le contrôleur principal. Dans cet exemple, la fonction FCT1 prend deux cycles de contrôle et FCT2 en prend trois. La fonction FCT1 lit ses entrées dans la première étape de contrôle et écrit ses sorties dans la deuxième étape de contrôle. FCT2 lit son entrée dans la première étape de contrôle, écrit la première sortie dans la deuxième étape de contrôle et écrit la deuxième sortie dans la troisième étape de contrôle.

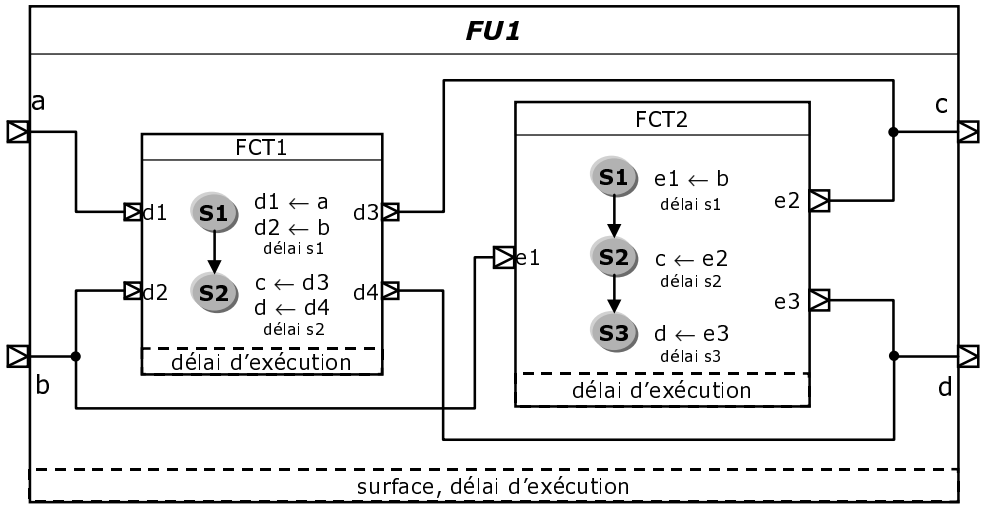


Figure 29 – Un exemple de description d'une unité fonctionnelle

Les unités fonctionnelles multicycle et/ou pipelinées sont supportées, mais les entrées doivent être stockées à durant le cycle dans lequel elles sont lues. La raison de cette restriction est que les valeurs des entrées ne restent pas stables quand le contrôleur principal fait des changements d'état.

5.2.2 L'algorithme du graphe bipartite pondéré

L'algorithme du graphe bipartite pondéré (BWGM) est une approche basée sur l'application de la théorie des graphes au problème de l'allocation et de l'affectation des unités fonctionnelles. Le circuit est modélisé comme un graphe bipartite pondéré et traité avec la méthode hongroise [FreTa87]. Le problème de correspondance peut être formulé comme suit :

Soit le graphe dirigé G, deux listes A et B de nœuds et un tableau qui indique pour chaque arc un poids. Tous les arcs de G doivent être dirigés des nœuds de A vers des nœuds de B, c.-à-d., G doit être un graphe bipartite. Le problème consiste à trouver un ensemble bipartite de G avec la correspondance maximale, c.-à-d., un ensemble d'arcs M de telle sorte que la somme des poids de tous les arcs en M soit maximale, et que deux arcs en M ne partagent pas un même point final.

L'allocation et l'affectation des unités fonctionnelles peuvent être modélisées comme un problème de correspondance. Les nœuds du graphe bipartite sont divisés en deux ensembles : un qui représente les opérations (ensemble A, voir Figure 30) et un autre qui représente les instances des unités fonctionnelles (ensemble B). Si l'unité fonctionnelle $\boldsymbol{fu_j}$ est capable d'exécuter l'opération $\boldsymbol{op_i}$, il y a un arc $\boldsymbol{e_{ij}}$ entre les nœuds correspondants. Le poids $\boldsymbol{w_{ij}}$ associé à l'arc $\boldsymbol{e_{ij}}$, correspond à la valeur de la fonction coût pour l'affectation de l'opération $\boldsymbol{op_i}$ à l'unité fonctionnelle $\boldsymbol{fu_j}$. Les arcs en gras sur la Figure 30 correspondent à la solution de correspondance maximale pour cet exemple. L'algorithme qui résout le problème de la correspondance a un temps d'exécution limité par O(|V|*|E|) [MehNä99], où V est le nombre de nœuds et E le nombre d'arcs du graphe bipartite.

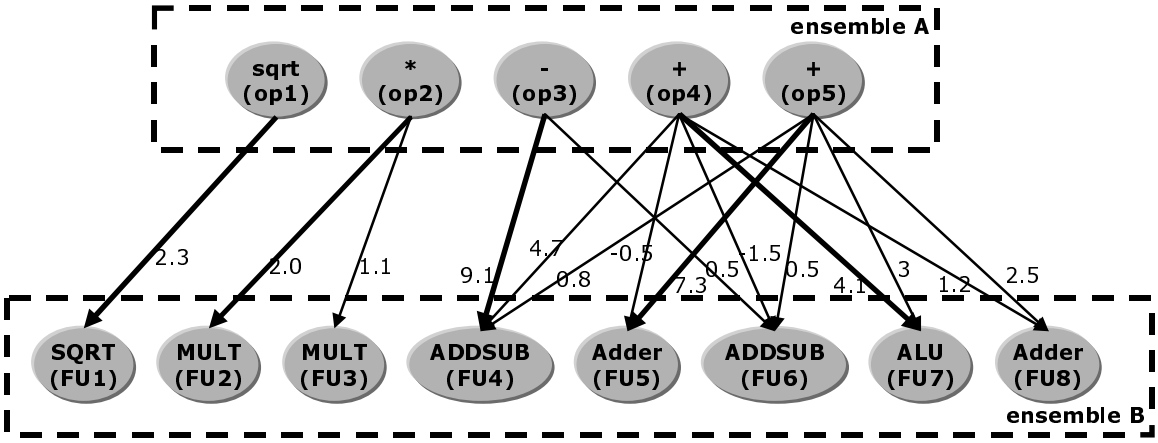


Figure 30 – Le modèle de graphe bipartite pondéré

Dans l'approche BWGM, les opérations sont affectées aux unités fonctionnelles par étape de contrôle, jusqu'à ce que toutes les étapes de contrôle aient été parcourues. L'algorithme initial travaille de manière locale avec une fonction de coût basé uniquement sur des facteurs globaux. Nous avons étendu cet algorithme pour définir une fonction de coût qui tient compte des facteurs locaux.

5.2.2.1 Le calcul des poids des arcs du graphe bipartite

Etant donné que l'allocation et l'affectation des registres sont implicitement faites par l'ordonnancement, l'effet de l'affectation des opérations aux unités fonctionnelles sur le coût des interconnexions devient clair. Par exemple, si nous affectons deux opérations qui emploient des registres d'entrée différents, dans une même unité fonctionnelle, un multiplexeur sera nécessaire au port d'entrée de l'unité fonctionnelle (ce qui représente un gain négatif). Si nous affectons deux opérations dont les variables de sortie ont été liées à un même registre, dans une même unité fonctionnelle, aucun multiplexeur n'est nécessaire au port d'entrée du registre (un gain positif). BWGM construit un tableau bidimensionnel (tableau de gain, ***GT***) pour garder cette information pour chaque paire d'opérations, selon l'équation (5-1).

$$GT(i,j)|_{nxn} = \alpha \cdot (|\, IR(op_i) \cap IR(op_j)\,| - \beta) + \gamma \cdot |\, OR(op_i) \cap OR(op_j)\,| \qquad (5\text{-}1)$$

où : *n est le nombre d'opérations ;*
α *est un paramètre définissable par le concepteur, il correspond au poids du facteur relatif à l'entrée des unités fonctionnelles ;*
β *est égal à max(|IR(op$_i$)∩IR(op$_j$)|) dans notre réalisation ;*
γ *est un paramètre définissable par le concepteur, il correspond au poids du facteur relatif à l'entrée des registres ;*
IR(op) est l'ensemble de registres alloués pour les variables d'entrée d'op ;
OR(op) est l'ensemble de registres alloués pour les variables de sortie d'op.

Dans l'approche de [Hsu90], le poids de l'arc $\boldsymbol{e_{ij}}$ ($\boldsymbol{w_{ij}}$) est simplement la somme des valeurs du tableau des gains pour l'opération $\boldsymbol{op_i}$ et toutes les opérations qui pourraient être exécutées sur l'unité fonctionnelle $\boldsymbol{fu_j}$, ce qui est donné par l'équation (5-2). Une fonction de coût basé uniquement sur l'équation (5-2) ne prend pas compte de la surface et du délai d'exécution des unités fonctionnelles.

$$w_{ij} = \sum_{op_k \in OPFU(fu_j)} GT(k,i) \qquad (5\text{-}2)$$

où : *OPFU(fu$_j$) est l'ensemble des opérations que l'unité fonctionnelle fu$_j$ est capable d'exécuter.*

Dans notre approche, le poids des arcs est une somme pondérée composée de trois facteurs (les poids sont définissables par le concepteur) : le facteur de surface, le facteur de délai et le facteur d'interconnexion. La fonction de coût utilisée est donné par l'équation (5-3).

$$w_{ij} = wa \cdot (1 - \frac{a_{fu_j}}{\max(A_{fu}^{\ i})}) + wd \cdot (1 - \frac{d_{fu_j^i}}{\max(D_{fu}^{\ i})}) + wi \cdot I_{ij} \qquad (5\text{-}3)$$

où : *wa est le poids du facteur de surface ; wd est le poids du facteur de délai ; wi est le poids du facteur d'interconnexion ;*
$A_{fu}^{\ i}$ *est un ensemble composé des surfaces des toutes les unités fonctionnelles capables d'exécuter l'opération i ;*
a_{fuj} *est la surface de l'unité fonctionnelle* fu_j *;*
$D_{fu}^{\ i}$ *est un ensemble composé des délais des toutes les unités fonctionnelles capables d'exécuter l'opération i ;*
$d_{fuj}^{\ i}$ *est le délai de l'unité fonctionnelle* fu_j *pour exécuter l'opération i ;*
I_{ij} *est le facteur d'interconnexion.*

Tous ces facteurs sont des mesures relatives (leur valeur maximale est égale à un), le concepteur peut définir leur poids pour établir une fonction de coût pour chaque application. Le facteur de surface est inversement proportionnel à la surface de l'unité fonctionnelle. Le facteur de délai est inversement proportionnel au délai de l'unité fonctionnelle. Le facteur d'interconnexion a deux composants : le composant global (***ig*$_{ij}$**) est donné par l'équation (5-2) et considère toutes les affectations possibles ; le composant local (***il*$_{ij}$**) tient compte des affectations faites pendant les étapes précédentes. Le facteur d'interconnexion correspondant à l'affectation de l'opération ***i*** à l'unité fonctionnelle ***j*** est donné par l'équation (5-4).

$$I_{ij} = 2 - \frac{il_{ij}}{\max(il_{ij})} + \frac{ig_{ij}}{\max(ig_{ij})} \qquad (5\text{-}4)$$

où : $il_{ij} = \sum_{op_k \in OPAFU(fu_j)} GT(k,i)$; $ig_{ij} = \sum_{op_k \in OPFU(fu_j)} GT(k,i)$;
OPFU(fu_j*) est l'ensemble des opérations que l'unité fonctionnelle* fu_j *est capable d'exécuter ;*
OPAFU(fu_j*) est l'ensemble des opérations affectées à l'unité fonctionnelle* fu_j*.*

5.3 Le ré-ordonnancement

Dans MUSIC, l'outil de ré-ordonnancement (***smis***) réalise deux tâches en parallèle : l'expansion des transitions pour tenir compte des unités fonctionnelles multicycle et l'ordonnancement des graphes de flot de données sous contrainte de ressources partiellement définies. L'outil est basé sur l'algorithme bien connu d'ordonnancement de liste (pour *list scheduling*) [Hu61] avec le coût des registres comme fonction de priorité. Dans le flot flexible de MUSIC, le ré-ordonnancement peut être exécuté après l'ordonnancement ou après

l'allocation et l'affectation des unités fonctionnelles. Alors, ***smis*** traite également les opérations non affectées et les instances d'unités fonctionnelles. Le concepteur peut définir :

- Une contrainte globale de timing : la période de l'horloge ;
- Les contraintes locales de timing : le temps d'exécution de chaque opération ou unité fonctionnelle ;
- Et les contraintes de ressource : le nombre maximum de chaque opération ou unité fonctionnelle.

Afin de respecter les contraintes, le ré-ordonnancement peut employer le chaînage, c.-à-d., ordonner des opérations avec dépendance de données dans un même cycle de contrôle. Chacune des transitions de la BFSM produite par l'ordonnancement peut contenir un graphe de flot de données. Par exemple, la description VHDL montré dans la Figure 31a contient deux états et quatre transitions (voir Figure 31b). La transition de l'état **S1** à l'état **S2** contient le graphe de flot de données **GFD1** (voir Figure 31c) et la transition de **S2** à **S1** contient le graphe **GFD2** (voir Figure 31d). Chaque calcul dans ces GFDs doit s'exécuter seulement quand la transition associée est activée.

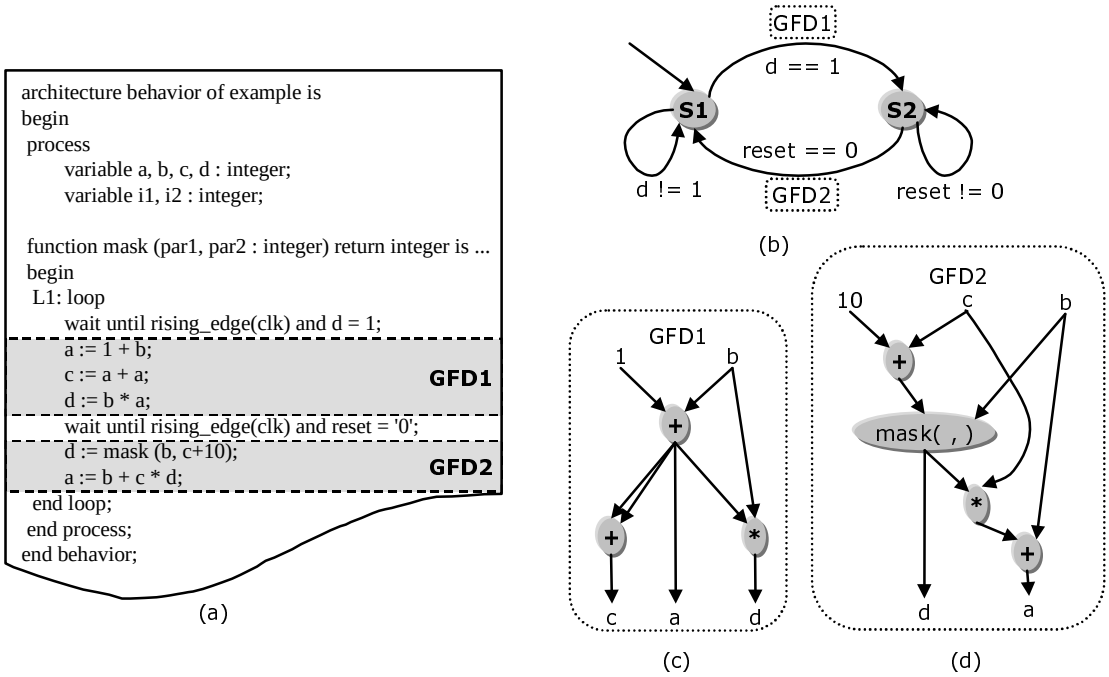


Figure 31 – Les graphes de flot de contrôle et de données

Il y a trois phases dans le ré-ordonnancement exécuté par MUSIC : l'ordonnancement des opérations et des opérateurs, l'expansion des transitions du contrôleur maître et

l'ordonnancement des transferts de données simples. Le *pseudo* code sur la Figure 32 décrit l'algorithme utilisé. Les différentes étapes seront détaillées dans la suite.

```
begin
détermine la période du cycle d'horloge ;
for each transition du GFC
  begin
   construis le GFD ;
   ASAP ;
   ALAP ;
   LIST ;
   crée les nouveaux registres ;
   exécute l'expansion des transitions du contrôleur ;
   schedule simple data transfers ;
   ordonne les transferts de données simples ;
  end;
end.
```

Figure 32 – L'algorithme de ré-ordonnancement

5.3.1 La détermination de la période du cycle d'horloge

La période du cycle de l'horloge du circuit peut être fournie par le concepteur comme une contrainte sur la ligne de commande qui sollicite ***smis***. Elle peut aussi être déterminée par calcul. Elle est déduite de la fonction la plus lente exécutée dans les unités fonctionnelles. Les temps d'exécution maximales pour les opérations non affectées doivent être indiqués sur la ligne de commande qui sollicite ***smis***. Dans le cas des unités fonctionnelles multicycle, l'étape de contrôle la plus lente désignée dans le fichier FULIB sera prise en compte. Si aucune valeur relative au temps d'exécution du circuit ne peut être obtenue par les méthodes décrites auparavant, ***smis*** considère que toutes les opérations non affectées et unités fonctionnelles ont le même temps d'exécution, mais ce temps restera non spécifié.

5.3.2 Le graphe de flot de données utilisé par ***smis***

Le graphe de flot de données utilisé par ***smis*** est un graphe classique, dirigé acyclique. Les nœuds représentent des opérations ou des unités fonctionnelles et les arcs représentent la relation de dépendance de données entre les nœuds (voir Figure 33). Il y a un arc allant du nœud ***i*** au nœud ***j*** si les données produites par l'opération ***i*** sont utilisées par l'opération ***j***. Il y a des nœuds et des arcs spéciaux qui sont employés pour représenter les unités fonctionnelles multicycles. Ils ont une autre interprétation : les nœuds sont des *pseudo* unités fonctionnelles

associées à chaque étape de contrôle et les arcs lient les nœuds dans l'ordre indiqué sur le protocole d'interface de l'unité fonctionnelle.

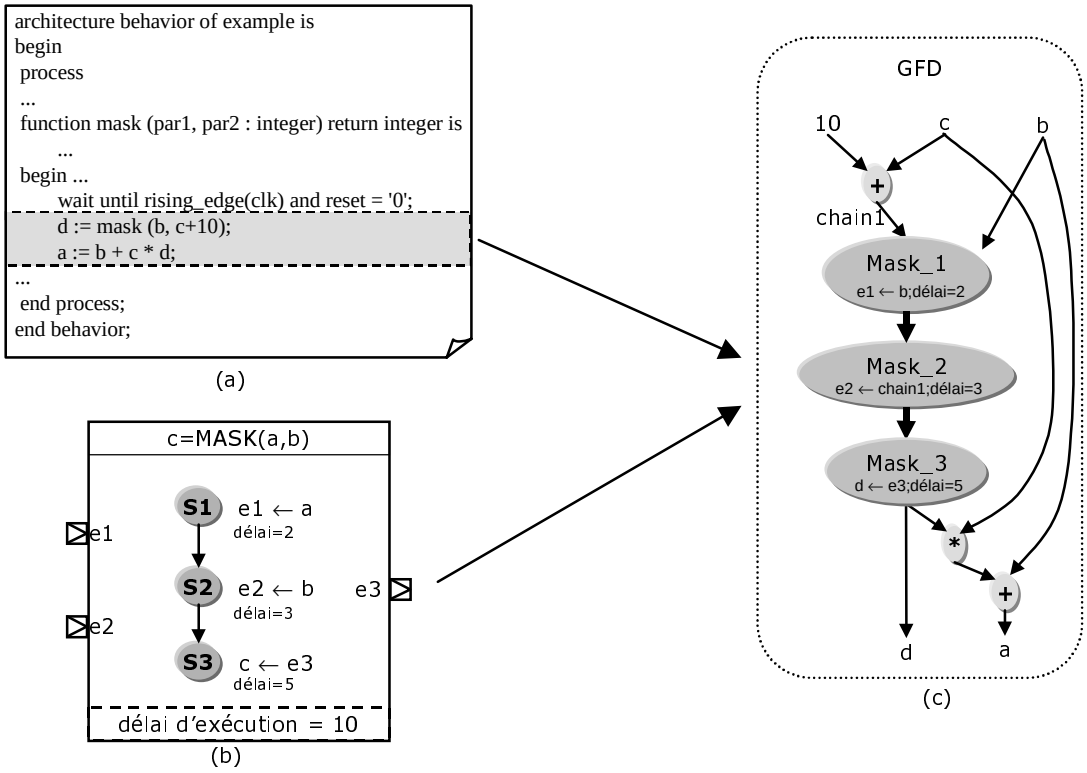


Figure 33 – Le graphe de flot de données utilisé par ***smis***

Chaque nœud possède un temps d'exécution maximal, il peut être fourni par le concepteur dans le fichier FULIB ou sur la ligne de commande qui sollicite ***smis***. Les nœuds associés aux unités fonctionnelles possèdent une liste de transferts qui met en œuvre le protocole d'interface pour la fonction exécutée (les paramètres sont substitués par les noms de signaux donnés au moment de l'appel la fonction ou le procédé correspondant). Par exemple, dans la Figure 33a les deux lignes de la description comportementale en gris seront traduites dans un GFD (voir Figure 33c). Nous supposons que l'appel de fonction ***mask()*** sera exécuté dans l'unité fonctionnelle ***MASK*** (décrite par la Figure 33b). L'unité fonctionnelle ***MASK*** requiert trois étapes de contrôle pour s'exécuter, ainsi elle est représentée par trois *pseudo* nœuds dans le GFD : Mask_1, Mask_2 et Mask_3. Les arcs épais sont des arcs spéciaux, c.-à-d., ils indiquent que les *pseudo* opérations ne peuvent pas être séparées par plus d'un cycle d'horloge.

5.3.3 L’ordonnancement aussitôt que possible

L’ordonnancement aussitôt que possible (ASAP, pour *As Soon As Possible*) et l’ordonnancement aussi tard que possible (ALAP, pour *As Late As Possible*) doivent être exécutés avant l’ordonnancement de liste (LIST) pour fournir la mobilité de chaque opération ou unité fonctionnelle. La mobilité est définie comme l’amplitude de la différence entre la dernière et la première étape de commande dans laquelle une opération peut être exécutée. La mise en œuvre de l’algorithme ASAP tient compte de la possibilité d’enchaîner des opérations ou des unités fonctionnelles. La Figure 34 montre l’algorithme en tant que *pseudo* code. Une opération est « prête » à s’exécuter si toutes les opérations qui fournissent ses entrées ont déjà été ordonnées.

```
begin
l’étape de contrôle courante est la première ;
do
   begin
      ordonne toutes les opérations « prêtes » à l’étape de contrôle
        courante ;
        /* la première fois, les opérations « prêtes » seront
            les opérations primaires, c.-à-d., ceux qui ne sont pas
            dépendantes des données venant d’autre opérations. */
      ordonne dans le cycle courant toutes les opérations dépendantes
        des données et qui sont à une distance de temps inférieure
        ou égale à la période du cycle ;
      incrémente le numéro de l’étape de contrôle ;
   repeat until toutes les opérations ont été ordonnées ;
   end;
end.
```

Figure 34 – L’algorithme ASAP

5.3.4 L’ordonnancement aussi tard que possible

L’ordonnancement ALAP peut fonctionner d’une manière très semblable au ASAP. Dans ce cas-ci, une opération est « prête » pour l’algorithme d’ordonnancement si toutes les opérations qu’elle alimente ont été déjà ordonnées (voir le *pseudo* code dans la Figure 35).

```
begin
l'étape de contrôle courante est la première ;
do
  begin
     ordonne toutes les opérations « prêtes » à l'étape de contrôle
       courante ;
       /* la première fois, les opérations « prêtes » seront
          ceux dont le résultat n'alimente aucune autre
          opération */
     ordonne dans le cycle courant toutes les opérations qui
       fournissent les entrées et qui sont à une distance de temps
       inférieure ou égale à la période du cycle ;
     incrémente le numéro de l'étape de contrôle ;
  repeat until toutes les opérations ont été ordonnées ;
  end;
renverse la numérotation des étapes de contrôle ;
end.
```

Figure 35 – L'algorithme ALAP

5.3.5 L'ordonnancement à base de liste

L'ordonnancement à base de liste (LIST) est employé dans MUSIC pour faire le ré-ordonnancement, appelé aussi « *re-timing* de haut niveau ». Il peut éclater certaine transition en plusieurs étapes de contrôle si le cycle d'horloge n'est pas suffisamment long pour exécuter toutes les opérations ordonnées dans cette transition. L'algorithme est décrit par le *pseudo* code montré sur la Figure 36. La mise en œuvre de l'algorithme LIST tient compte de la possibilité d'enchaîner des opérations ou des unités fonctionnelles.

```
begin
l'étape de contrôle courante est la première ;
do
   begin
     établir une « liste d'opérations prêtes » qui peuvent être
       ordonnées dans l'étape de contrôle courante ;
     joins les opérations dépendantes des données et qui sont à une
       distance de temps inférieure ou égale à la période de cycle
       à « liste d'opérations prêtes » ;
     établis la priorité pour chaque opération dans la « liste
       d'opérations prêtes » ;
     trie la « liste d'opérations prêtes » par priorité
       décroissante ;
     choisis les opérations de la « liste d'opérations prêtes » ;
     incrémente le numéro de l'étape de contrôle ;
  repeat until la « liste d'opérations prêtes » est vide ;
  end;
end.
```

Figure 36 – L'algorithme LIST

L'algorithme fonctionne avec une « liste d'opérations prêtes ». Cette liste contient les opérations qui ont toutes leurs prédécesseurs déjà ordonnées et les opérations qui sont à une distance de temps inférieure ou égale au cycle d'horloge. La « liste d'opérations prêtes » est triée par priorité décroissante, c'est la fonction de priorité utilisée qui définit le comportement de l'algorithme. La fonction de priorité utilisée essaye de réduire au minimum le nombre de registres qui doivent être créés pour stocker des résultats intermédiaires. Cette fonction dépend de deux facteurs : la mobilité et le facteur d'enchaînement. La mobilité est donnée par les ordonnancements ASAP et ALAP. Les priorités plus élevées sont accordées aux opérations avec moins de mobilité. Une opération peut être enchaînée avec d'autres si toutes leurs prédécesseurs étaient ordonnées (dans le cycle courant inclus). Si c'est le cas, la priorité augmente proportionnellement au nombre de prédécesseurs ordonnés dans le cycle courant. Par exemple, dans la Figure 37a il est supposé que l'ordonnancement doit employer seulement deux additions et une multiplication dans un même cycle d'horloge. Sans chaînage, il produira une solution qui a besoin de trois registres pour stocker des résultats intermédiaires. Dans ce cas précis, avec le chaînage des additions, seulement un registre sera nécessaire (voir Figure 37b).

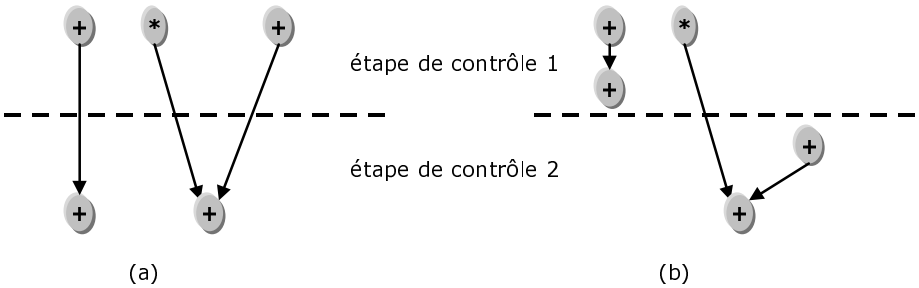


Figure 37 – Un exemple de chaînage

Pour choisir les opérations de la « liste d'opérations prêtes » qui seront ordonnées, l'algorithme représenté dans la Figure 38 est suivi. Une *pseudo* opération sera ordonnée si leur prédécesseur est aussi une *pseudo* opération qui a déjà été ordonné, indépendamment de sa priorité. Dans ce cas-là, les contraintes de ressources ont été prises en considération pendant l'ordonnancement de la première *pseudo* opération de la chaîne. Cette méthode garantit qu'il n'y aura pas plus d'une étape de contrôle entre deux *pseudo* opérations reliées par un arc spécial, en d'autres termes, l'opération multicycle ne sera pas cassée en morceaux.

```
begin
   vérifie la présence des pseudo opérations ;
   ordonne les pseudo opérations ;
   prends la première opération dans la « liste d'opérations
     prêtes » ;
do
   begin
     while ( la limite de ressources a été atteinte pour
             l'opération choisie )
       choisis l'opération prochaine dans la « liste d'opérations
         prêtes » ;
     while ( l'opération choisie a un prédécesseur non ordonnée )
       choisis l'opération prochaine dans la « liste d'opérations
         prêtes » ;
     ordonne l'opération choisie ;
     mets à jour les priorités des opérations ;
     trie à nouveau la « liste d'opérations prêtes » ;
   repeat until la « liste d'opérations prêtes » est vide
          or c'est impossible de choisir une opération ;
  end;
end.
```

Figure 38 – L'algorithme qui choisit les opérations de la « liste d'opérations prêtes »

5.3.6 La création de nouveaux registres

Si les valeurs produites à la sortie des opérations enchaînées sont consommées en dehors du cycle de production, il peut être nécessaire de créer des registres pour stocker les valeurs intermédiaires. Dans ce cas-ci, la taille du registre (nombre de bits) doit être déterminée par l'outil. Les problèmes impliqués ici sont les suivants :

- Si la consommation implique d'autres opérations, elle sera traitée automatiquement par l'algorithme de ré-ordonnancement, autrement cela doit être un simple transfert de données à une variable définie par le concepteur. Dans ce dernier cas :

  → S'il y a de multiples consommations de la valeur intermédiaire dans une même étape de contrôle en dehors de son étape de contrôle de production, alors la valeur sera stockée dans un registre (il s'agit d'une sous-expression) ;

  → S'il y a une consommation de la valeur intermédiaire en dehors de son étape de contrôle de production, alors un registre sera créé (il est appelé variable automatique) ;

  → Si toutes les consommations de la valeur intermédiaire sont dans la même étape de contrôle de la production alors aucun registre ne sera créé.

- Tous les transferts de données qui consomment des valeurs intermédiaires doivent être ordonnés à une des étapes de contrôle qui suivent la production de ces valeurs. La réduction du nombre de transferts parallèles dans chaque étape de contrôle peut réduire la consommation d'énergie du circuit.
- La taille des registres (c.-à-d., le nombre de bits) créés pour stocker les valeurs intermédiaires doit être déterminée par deux méthodes :

  → Si la valeur est stockée dans les registres définis par l'utilisateur, la taille du registre le plus grand est adoptée ;

  → La taille peut être déterminée automatiquement si la valeur intermédiaire est produite par une unité fonctionnelle, dans ce cas-ci la taille de chaque sortie est connue ;

  → Si la taille ne peut pas être déterminée par les techniques précédentes, elle doit être déduite à partir du type d'opération et de la taille des entrées.

5.3.7 L'expansion des transitions et l'ordonnancement des transferts

Le support des unités fonctionnelles multicycles requiert l'expansion des transitions, ce qui signifie, créer de nouveaux états pour insérer de nouvelles étapes de contrôle. Après cette expansion, des transferts simples de données doivent être ordonnés dans les nouvelles étapes de contrôle créées. Il y aura deux cas :

- S'il n'y a aucune consommation des valeurs intermédiaires produites par les opérations chaînées, alors il n'y a aucune dépendance de données. Dans ce cas-là, le transfert de données peut être ordonné dans l'une des étapes de contrôle créées. L'outil essayera de distribuer uniformément ce genre de transferts sur les étapes de contrôle créées ;
- S'il y a de la consommation des valeurs intermédiaires produites par les opérations chaînées, alors la dépendance des données oblige à ordonner ce genre de transferts dans les étapes de contrôle qui suivent celle de la production des valeurs intermédiaires. Dans ce cas-là, il y a deux possibilités :

→ Si la valeur intermédiaire est stockée dans un registre, l'outil sélectionnera l'étape de contrôle valide avec le plus petit nombre de transferts déjà ordonnés ;

→ Autrement, l'outil sélectionnera l'étape de contrôle de production de la valeur intermédiaire.

5.4 L'allocation et l'affectation des interconnexions

Cette section présente quelques techniques d'optimisation utilisées pour l'allocation et l'affectation des interconnexions. Il existe deux approches pour la synthèse d'interconnexions dans les parties opératives. La première génère une architecture à base de bus. La seconde produit une architecture à base de multiplexeurs. Dans ce dernier cas, l'optimisation est généralement laissée à la synthèse logique. Il existe aussi des approches qui combinent les deux solutions. L'outil qui fait l'allocation et l'affectation des interconnexions (***sag***) utilise trois modèles d'interconnexion pour la partie opérative :

1. **Basé sur des multiplexeurs :** l'heuristique de fusionnement des multiplexeurs est utilisée ;
2. **Basé sur des bus :** trois algorithmes peuvent être choisis : l'heuristique itérative, l'algorithme FBS et l'algorithme FBD ;
3. **Style mixte :** la limite indiquant le nombre d'entrées qui change un multiplexeur en cellules à trois états peut être fournie sur la ligne de commande qui sollicite ***sag***.

Le reste de cette section détaille les trois heuristiques de synthèse d'interconnexion utilisées dans MUSIC.

5.4.1 L'heuristique de fusionnement de multiplexeurs

Habituellement, les systèmes qui emploient le modèle architectural basé sur des multiplexeurs limitent leurs optimisations de haut niveau à la réduction du nombre de multiplexeurs. Ensuite, au niveau RT, les puissantes optimisations de la synthèse logique peuvent être employées sur la totalité de l'architecture. Certains systèmes de synthèse de niveau RT vont encore plus loin en faisant l'optimisation à travers les frontières des macro-cellules (additionneurs, ULAs, multiplicateurs, multiplexeurs, etc). Pour réduire le nombre de

multiplexeurs, il est possible de profiter de la propriété commutative de certaines opérations arithmétiques et logiques pour changer l'ordre des entrées des unités fonctionnelles.

Deux multiplexeurs peuvent être fusionnés s'ils ont une séquence de sorties « compatibles », c.-à-d., si à chaque étape de contrôle une des trois cas suivants se présente :

1. les valeurs des sorties sont les mêmes ;
2. un d'entre eux n'est pas utilisé, c.-à-d., la valeur de sortie n'est pas utilisée dans le circuit ;
3. les deux multiplexeurs ne sont pas utilisés.

Les outils de synthèse de haut niveau peuvent profiter des informations produites par l'ordonnancement pour créer une liste des valeurs de sortie pour chaque multiplexeur ordonnée selon les étapes de contrôle. Il est probable que les multiplexeurs seront utilisés seulement quelques cycles pendant le temps total d'exécution. En conséquence, beaucoup de paires de multiplexeurs peuvent être fusionnées dans un multiplexeur plus grand qui respecte les valeurs de sortie de tous les multiplexeurs fusionnés.

Cependant, les fusions ne seront pas toutes avantageuses, parce que l'introduction d'un multiplexeur plus grand peut signifier une augmentation du chemin critique du circuit. Quelques conditions doivent être réunies pour s'assurer qu'une fusion donnée mènera à une solution plus avantageuse. La Figure 39 donne un exemple, les multiplexeurs **A** et **B** peuvent être remplacés par le multiplexeur **C**. La valeur *X* signifie que la valeur de sortie du multiplexeur n'est pas utilisée dans l'étape de contrôle correspondant. Admettons que la mise en œuvre des multiplexeurs utilisera seulement des multiplexeurs à deux entrées. Dans ce cas-ci, le nombre de multiplexeurs à deux entrées nécessaires a été réduit de quatre (dans le cas de la mise en œuvre de **A** et **B**) à trois (pour la mise en œuvre de **C**).

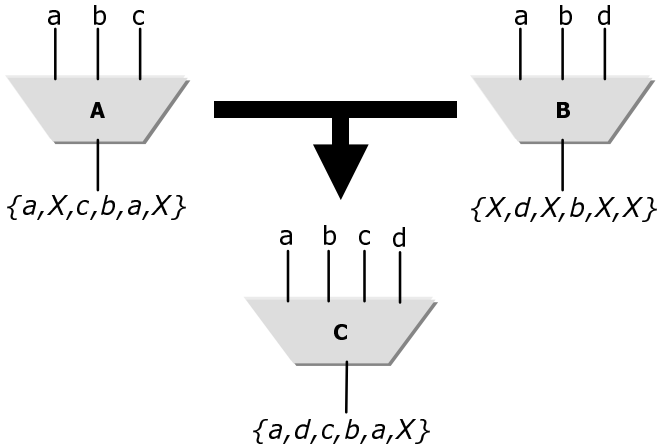


Figure 39 – Le fusionnement des multiplexeurs compatibles

Si nous supposons qu'un multiplexeur avec ***n*** entrées (***n*** ≥ 2, ***n*** ∈ **N**) sera mis en œuvre comme un arbre équilibré de multiplexeurs à deux entrées. Alors (***n***-1) multiplexeurs seront nécessaires pour un multiplexeur à ***n*** entrées (voir Figure 40). La taille de l'arbre ***T*** sera donnée par l'équation (5-5), et elle est proportionnelle au délai (voir Figure 41). Un modèle très simple d'évaluation peut être déduit de ces considérations. Dans ce cas-ci, la surface et le délai du multiplexeur seront évalués en fonction de son nombre d'entrées.

$$T = \text{int}\left(\log_2\left(n-1\right)\right)+1 \quad \left\{n \geq 2, n \in \text{N}\right\} \tag{5-5}$$

où : *n est le nombre d'entrées du multiplexeur.*

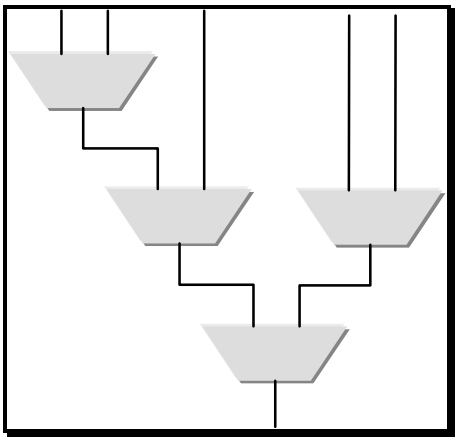


Figure 40 – La réalisation d'un multiplexeur à cinq entrées

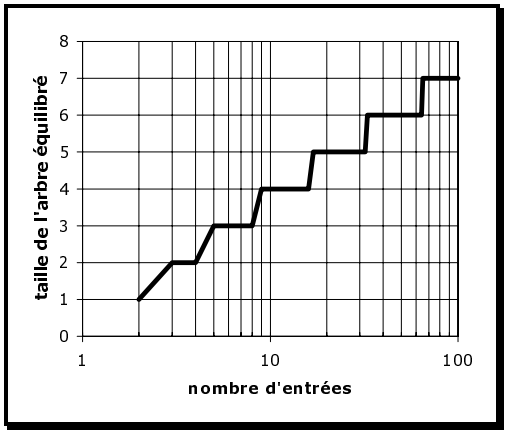


Figure 41 – Le modèle de délai pour les multiplexeurs

Pour fusionner deux multiplexeurs sans changer le délai de la réalisation par un arbre équilibrée, la condition montrée dans l'équation (5-6) doit être respectée.

$$\text{int}\left(\log_2\left(n_1 + n_2 - n_c - 1\right)\right) \leq \text{int}\left(\log_2\left(\max\left(n_1, n_2\right) - 1\right)\right) \tag{5-6}$$

où : *n_1 et n_2 sont le nombre d'entrées des multiplexeurs ;*
n_c est le nombre d'entrées communes.

Il faut noter que cette formulation ne tient pas compte des modifications des fonctions de commande des multiplexeurs. Certains changements peuvent générer des coûts additionnels.

5.4.2 L’heuristique de fusionnement de bus

Comme nous avons montré dans la section 4.2, peu de travaux considèrent la réduction du nombre de cellules à trois états en tant que technique principale d’optimisation. Dans le cas des architectures à base de bus, la plupart des travaux se sont concentrés sur la réduction du nombre de bus. Vu les technologies actuelles permettant plusieurs couches de routage, le nombre de bus devient beaucoup moins critique que le nombre de cellules à trois états. Nous avons développé deux algorithmes orientés vers la réduction du nombre de cellules à trois états, tout en essayant de réduire le nombre de bus en conséquence de ce processus. L’utilisation des bus partagés ne représente aucun gain pour les styles de dessins des masques avec plus de liberté d’emplacement que le style *bit-slice*. Par conséquent, les algorithmes présentés ici supposent que les bus ne sont pas partagés. L’insertion des répéteurs est employée pour augmenter la vitesse des bus longs, au prix d’une augmentation de la surface et de la consommation d’énergie. Nous avons choisi de ne pas insérer des répéteurs parce que dans ce cas il nous faut des informations géométriques sur la longueur des bus. Par conséquent, nous supposons que l’insertion de répéteurs aura lieu après la phase d’optimisation logique. Ces choix ont été faits pour maintenir l’indépendance de nos algorithmes par rapport aux informations géométriques du dessin des masques.

Les algorithmes développés utilisent une heuristique de fusionnement de bus, bien que le but soit de réduire le nombre de cellules à trois états et pas le nombre de bus. Pour réaliser l’optimisation des interconnexions nous utilisons des diagrammes d’interconnexions composés des sources, des bus et des destinations. Chaque fois qu’un élément est alimenté par deux sources différentes, des cellules à trois états sont nécessaires.

La Figure 42 montre des diagrammes d’interconnexions : les transferts exécutés par le chemin de données sont représentés par des flèches entre l’ensemble de sources de données, l’ensemble de bus et l’ensemble de destinations. L’algorithme est interactif, il commence par une première solution qui associe un bus pour chaque destination de données (voir Figure 42a). Dans cette solution (qui nous l’appellerons l’algorithme FBD), il y a des cellules à trois états entre les sources de données et les bus partagés seulement. Une variante de cet algorithme commence avec une solution qui associe un bus pour chaque source de données (nous l’appellerons l’algorithme FBS).

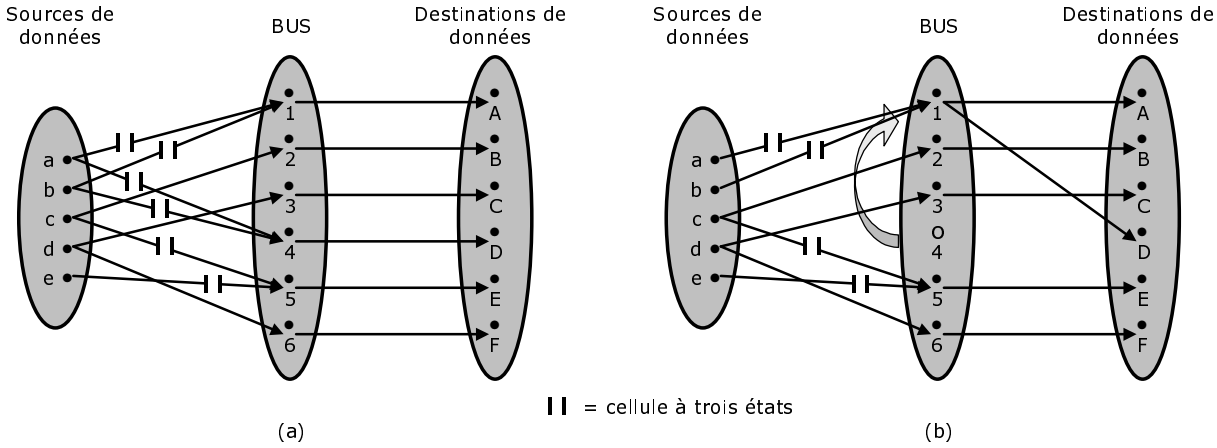


Figure 42 – Un exemple de fusionnement de bus

Dans le cas où on ne supporte pas le chaînage des opérations, le concept de temps reste implicite dans la Figure 42. L'ensemble des sources de données est lié à l'ensemble de destinations de données par l'ensemble de bus. Si dans un cycle donné nous devons faire ***n*** transferts, nous aurons besoin de ***n*** bus, parce que les transferts sont faits en parallèle. Le nombre minimal de bus est égal à la valeur maximale de ***n***. Des cellules à trois états doivent être ajoutées dans deux cas :

1. Pour les sources de données qui partagent un bus,
2. Et pour les destinations de données reliées à plus d'un bus.

Par ailleurs, leur utilisation peut être évitée dans les cas suivants :

1. Pour les bus reliés à seulement une source de données,
2. Et pour les destinations de données reliées à un bus seulement.

Il est uniquement possible de fusionner deux bus si à chaque cycle au moins un d'eux est libre, c.-à-d., qu'il ne réalise aucun transfert. La solution initiale de l'algorithme FBD est toujours réalisable, puisque, dans tous les cycles, il est interdit de relier plus d'une source de données à une destination. Pour la même raison, la solution initiale de l'algorithme FBS est également toujours réalisable. Le nombre de cellules à trois états après la fusion de deux bus ***a*** et ***b*** peut être calculé en utilisant l'équation (5-7), il est supposé que la fusion soit possible.

$$TSn(a,b) = \sum_{i=1}^{BN} ds(a,b)_i + \sum_{j=1}^{DDN} db(a,b)_j \qquad (5\text{-}7)$$

où : *TSn(a,b) est le nombre de cellules à trois états après le fusionnement des bus **a** et **b***
BN est le nombre de bus
*$ds(a,b)_i$ est le nombre de sources de données pour le bus **i** (si il y en a deux ou plus)*
DDN est le nombre de destinations de données
*$db(a,b)_j$ est le nombre de destinations de données pour le bus **j** (si il y en a deux ou plus)*

Pour trouver la solution optimale avec l'heuristique de fusionnement de bus, nous devons évaluer tous les ordres possibles de fusion et sélectionner celui qui aura comme conséquence le nombre minimal de cellules à trois états. La solution exhaustive doit tester un grand nombre de possibilités (***Ne***), donné par l'équation (5-8). Par exemple, pour le GCD avec l'algorithme FBS, Ne est égale à 56.700, et la solution exhaustive est encore possible (dans ce cas-ci, la solution optimale a quatre bus et douze cellules à trois états). Pourtant, pour le calcul de tri à bulles (*Bubble sort*), il y aura environ $4x10^{13}$ possibilités. Il est clair que ce type de démarche est difficile à appliquer pour les grands exemples.

$$N_e = \prod_{b=BN}^{\min BN} \left\lfloor \frac{(b-1)\cdot b}{2} \right\rfloor \qquad (5\text{-}8)$$

où : *BN est le nombre initial de bus ;*
minBN est le nombre minimal de bus.

La solution initiale des heuristiques de fusionnement de bus génère un nombre réduit de cellules à trois états et un grand nombre de bus. Après cette première allocation, les bus sont fusionnés pour obtenir une solution avec moins de cellules à trois états. Par exemple, la Figure 42b montre que le fusionnement des bus ***1*** et ***4*** peut éliminer deux récepteurs. Le *pseudo* code sur la Figure 43 décrit l'heuristique de fusionnement de bus. Pour éviter les minimums locaux, le processus de fusion est fait pendant qu'il est possible, même si parfois il augmente le nombre de cellules à trois états. Toutes les solutions intermédiaires sont stockées. En raison des considérations d'efficacité, l'ensemble des fusions impossibles est stocké pour la prochaine itération de l'algorithme. L'équation (5-7) doit être évaluée pour toutes les fusions possibles, quand il n'y a plus de fusion possible où le nombre minimal de bus est atteint, le processus de fusion est fini. Alors, la séquence des fusions est analysée pour trouver la solution avec le nombre minimal de cellules d'interconnexion. Les algorithmes présentés dans cette section sont tout à fait rapides. Ils peuvent manipuler de grandes applications dans un temps de CPU raisonnable, c.-à-d., quelques secondes sur un poste de travail SPARC 20.

```
begin
détermine le nombre de bus minimal ;
alloue un bus pour chaque source ou destination de
  données ;
do
  begin
     choisis la fusion qui provoque la plus grande
       réduction du nombre de cellules à trois états ;
     fusionne les bus choisis ;
   repeat until c'est impossible de fusionner
          or le nombre de bus est égal au minimal ;
   end;
choisis la séquence de fusions qui amène à la solution
  avec le nombre plus petit de cellules à trois états ;
end.
```

Figure 43 – L'heuristique de fusionnement de bus

5.4.3 L'heuristique itérative

L'heuristique itérative est une approche constructive pour résoudre le problème d'allocation des interconnexions. La solution est établie en ajoutant, à chaque étape, le moins possible de cellules à trois états. La Figure 44 montre l'algorithme en *pseudo* code.

```
begin
détermine le nombre de bus minimal ;
alloue les bus aux sources de données le plus
  utilisées;
do
  begin
     choisis le transfert le plus approprié selon les
       règles suivantes :
       c1 : le nombre de transferts de la même étape de
              contrôle déjà alloués est maximal ;
       c2 : la source de données est déjà allouée à un
              bus ;
       c3 : la destination de données est déjà allouée
              à un bus ;
       c4 : l'allocation de ce transfert à un bus
              disponible ajoutera le nombre le plus
              petit de cellules à trois états à la
              solution courante ;
   repeat until tous les transferts ont été alloués ;
   end;
end.
```

Figure 44 – L'algorithme pour l'heuristique itérative

L'avantage principal de l'heuristique itérative est que la solution proposée a le nombre minimal de bus (supposant que les bus ne soient pas partagés). Cependant, cette approche a un

inconvénient majeur : la solution peut avoir des cellules à trois états supplémentaires dues à la réutilisation excessive des bus.

La première étape de l'algorithme est le calcul du nombre minimal de bus. Puis, la priorité d'allocation est accordée aux transferts qui ont les sources de données les plus utilisés. Il s'agit d'attribuer d'abord les bus disponibles aux sources de données les plus utilisés, parce qu'ainsi quelques cellules à trois états peuvent être économisées.

La prochaine étape consiste à trouver le transfert le plus approprié pour procéder à l'allocation. Celui qui a le moins d'impacts sur le nombre final de cellules à trois états sera choisi. Les quatre règles énumérées dans la Figure 44 (**c1**, **c2**, **c3** et **c4**) sont employées pour évaluer les transferts. La première règle (**c1**) accorde plus de priorité aux transferts qui ont le plus bas degré de liberté pour l'allocation. Les règles **c2** et **c3** accordent la priorité aux transferts qui ont la source ou la destination de données (respectivement) déjà allouées à un bus. La règle **c4** accorde la priorité aux transferts qui ajoutent moins de cellules à trois états dans la solution courante. La somme pondérée des règles décide quel transfert sera alloué et à quel bus. Si plus d'un transfert a la priorité d'allocation, alors un d'entre eux est arbitrairement choisi parce que l'algorithme doit allouer un transfert dans chaque itération. Cette restriction garantie que l'algorithme se terminera et sortira de la boucle. Naturellement, plus de règles de décision peuvent facilement être ajoutées à l'ensemble de base pour améliorer la performance de l'algorithme.

Afin d'étudier l'importance relative de ces règles, nous avons réalisé plusieurs variantes qui utilisent un sous-ensemble de ces règles. Ces variantes seront appelées $\mathbf{c_{ijkl}}$ où **i**, **j**, **k** et **l** désignent les règles utilisées. Par exemple, $\mathbf{c_{134}}$ désigne une variante qui utilise les règles **c1**, **c3** et **c4** et qui n'utilise pas **c2**.

Le Tableau 10 montre le nombre de cellules à trois états généré par chacune des variantes pour l'ensemble de circuits de test présentés dans le Tableau 6.

Circuit	c_{123}	c_{1234}	c_{134}	c_{234}	C_{124}	$c_{12\text{-}34}$	$c_{12\text{-}3}$	meilleur résultat
Gcd	13	13	13	15	15	14	13	13
Pid	24	23	23	21	24	23	18	18
Abr	26	26	27	28	27	43	41	26
multi	30	28	28	30	32	46	48	28
es	30	32	31	33	31	32	29	29
fpu	31	31	35	42	38	42	44	31
Bubble	37	37	44	39	35	28	30	30
sqrt	38	39	36	36	42	28	28	28
Answer	37	45	44	48	46	39	50	37

Tableau 10 – Les résultats pour les variantes de l'heuristique itérative

La Figure 45 montre le pourcentage moyen de cellules à trois états supplémentaires, par rapport à la meilleure solution, pour chaque variante. La variante $\mathbf{c_{123}}$ a eu la meilleure performance moyenne pour cet ensemble d'exemples d'application.

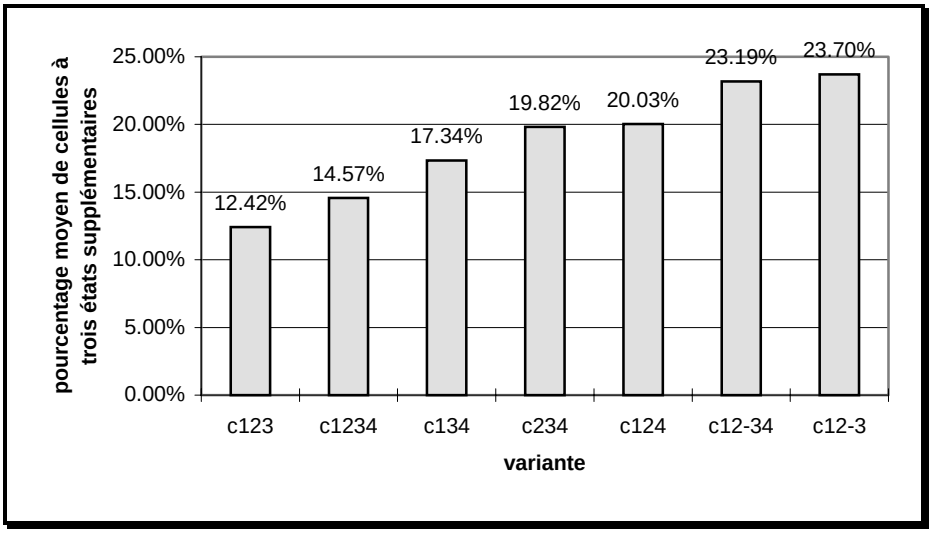
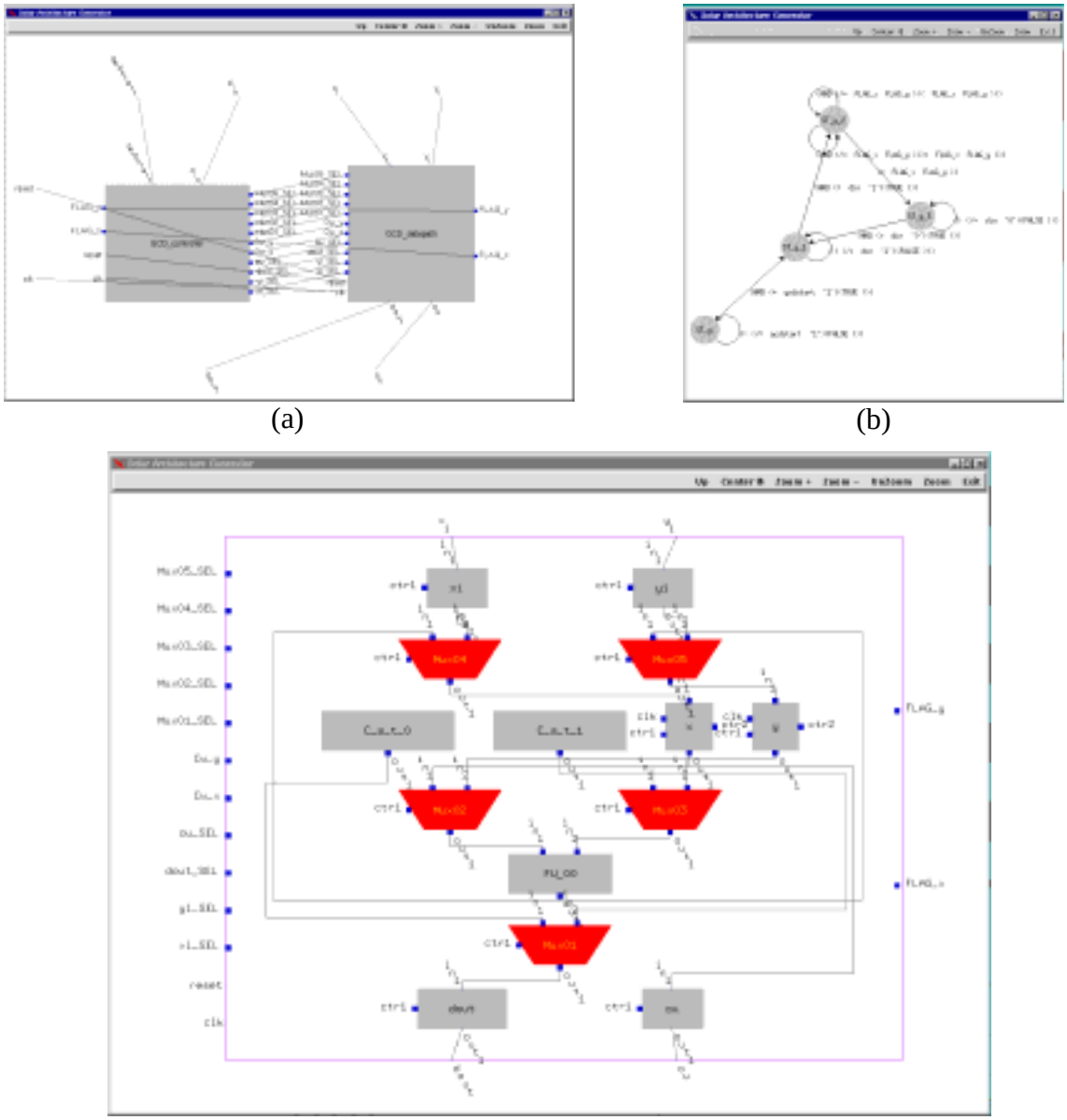


Figure 45 – La comparaison entre la performance moyenne des variantes

Les résultats montrent clairement que l'utilisation de la règle **c4** n'est pas avantageuse puisqu'elle est de loin la plus lente à calculer. Cependant, cette conclusion est valide seulement dans l'espace des solutions limitée représentée par nos neuf petits exemples. Dans la réalisation finale, le concepteur a toujours la possibilité de choisir les valeurs des poids pour combiner les quatre règles.

5.5 La génération de l'architecture

La génération d'architecture produit une structure composée d'un contrôleur et d'un chemin de données à partir des informations générées par l'ordonnancement, l'allocation et l'affectation des unités fonctionnelles et des interconnexions. Le circuit généré a la même interface que celle qui a été définie au niveau comportemental (voir Figure 46a).



(a) (b)

(c)

Figure 46 – L'outil de génération d'architecture

Il contient une instance du contrôleur (Figure 46b) et une instance du chemin de données (Figure 46c). L'outil de génération de l'architecture (***sag***) fait l'allocation et l'affectation des interconnexions pendant son flot d'exécution. Le modèle d'interconnexion

du chemin de données peut être basé sur des bus ou sur des multiplexeurs (comme c'est le cas sur la Figure 46c). Le concepteur peut choisir entre trois modèles d'interconnexion :

- **Basé sur des multiplexeurs :** l'heuristique de fusionnement des multiplexeurs est utilisée ;
- **Basé sur des bus :** trois algorithmes peuvent être choisis : l'heuristique itérative (les poids pour les critères 1 à 4 peuvent être indiqués sur la ligne de commande qui sollicite ***sag***), l'algorithme FBS et l'algorithme FBD ;
- **Style mixte :** la limite indiquant le nombre d'entrées qui change un multiplexeur en cellules à trois états peut être fournie sur la ligne de commande qui sollicite ***sag***.

Le *pseudo* code décrivant l'algorithme mis en œuvre dans l'outil de génération de l'architecture est montré sur la Figure 47. Dans la première étape, les ports du circuit sont créés en correspondance avec les ports de l'entité décrits dans la description comportementale. Les ports du signal d'horloge et du signal de remise à zéro seront identifiés s'ils étaient marqués dans la description initiale par des attributs. Autrement, un avertissement sera donné au concepteur. La prochaine étape est la création du chemin de données et du contrôleur. En conclusion, l'architecture est écrite dans le format SOLAR.

```
begin
  crée les ports du circuit ;
  /* crée le chemin de données */
    crée les ports du chemin de données;
    crée les unités fonctionnelles ;
    crée les registres à lecture seule ;
    crée les registres ;
    exécute l'allocation et l'affectation des interconnexions ;
  /* crée le contrôleur */
    crée les ports du contrôleur ;
    construis l'interface entre le contrôleur et le chemin
      de données ;
  write le fichier de sortie en SOLAR ;
end.
```

Figure 47 – L'algorithme de génération de l'architecture

Les ports du chemin de données et les unités de communication externes (ECUs) sont créés à partir des ports du circuit qui sont référencés dans les actions de la BFSM d'entrée. Un port d'entrée (ou de sortie) du circuit et l'ECU correspondant sont créés dans le chemin de données, si le port d'entrée (ou de sortie) est lu (écrit) dans la description comportementale. Les instances des unités fonctionnelles ont été créées pendant l'étape d'allocation et

d'affectation, elles sont simplement déplacées de la BFSM au chemin de données. Les constantes numériques présentes dans les actions de la BFSM créent des registres à lecture seule. Des registres sont créés selon l'utilisation des variables dans les actions de la BFSM : si la variable est référencée dans une condition, alors elle sera stockée dans un registre compte-rendu et un port de contrôle de sortie est ajouté au chemin de données. Sinon, la variable est stockée dans un registre commun. Dans la prochaine étape, les algorithmes d'allocation et d'affectation des interconnexions décrites dans la section 5.4 sont appelés selon le choix du concepteur.

Les ports du contrôleur sont créés pour chaque port du circuit référencé dans une condition quelconque de la BFSM. La prochaine étape est la génération de l'interface entre le contrôleur et le chemin de données. Pour chaque signal de contrôle de registre, de multiplexeur, de cellule à trois états et d'unité fonctionnelle ; un port de contrôle de sortie est créé dans le contrôleur et un port de contrôle d'entrée correspondante est créé dans le chemin de données. Les valeurs des registres comptes-rendus du chemin de données sont dirigées vers les ports de contrôle d'entrée du contrôleur. Finalement, l'architecture générée est écrite dans un fichier de sortie en langage SOLAR.

5.6 Conclusion

Ce chapitre a présenté l'ensemble des algorithmes pour la synthèse comportementale développés dans le cadre de cette thèse. Trois outils ont été développés : ***sal*** pour l'allocation et l'affectation des unités fonctionnelles, ***smis*** pour le ré-ordonnancement et ***sag*** pour l'allocation et l'affectation des interconnexions et la génération d'architecture. Dans l'ensemble, ces outils représentent environ 20000 lignes de code C++. La plupart des algorithmes de manipulation de graphes utilisent la bibliothèque LEDA [MehNä99] du *Max-Planck-Institut für Informatik*. Le modèle de représentation unique en SOLAR a permis de casser l'interdépendance entre les outils de synthèse comportementale. L'écriture d'outil pour un flot de synthèse flexible n'est pas une tâche facile parce qu'ils doivent être capables de s'adapter aux différents flots possibles. Les paramètres fournis sur la ligne de commande qui sollicite les outils peuvent changer le comportement des algorithmes. En outre, la façon modulaire d'implémenter les tâches de la synthèse comportementale est essentielle pour notre méthodologie de synthèse flexible. L'utilisation des techniques de conception pour les deux grands domaines d'applications (le flot de contrôle et le flot de données) permet de traiter un spectre d'applications beaucoup plus large.

Chapitre 6

L'Estimation de Performance au Niveau Système[1]

La principale cause qui enchérit toutes choses en quelque lieu que ce soit est l'abondance de ce qui donne estimation et prix aux choses.

Bodin (Jean), Réponse au paradoxe de monsieur de Malestroit.

Ce chapitre présente une méthode pour l'estimation de performance au niveau système. L'estimation de performance joue un rôle fondamental pour l'exploration de l'espace des solutions. C'est l'estimation de performance qui permet de guider l'optimisation du partitionnement du système en parties matérielles et parties logicielles. En outre, elle permet une évaluation rapide de l'affectation des canaux de communication aux différents protocoles de la bibliothèque de canaux.

[1] Ce chapitre a été publié comme un rapport de recherche au laboratoire TIMA : « Estimation de performance au niveau système », Amer Baghdadi, Wander Cesário, Márcio Teruya, Thiery Roudier, Nacer-Eddine Zergainoh et Ahmed Jerraya. La contribution de l'auteur a consisté en la définition de la démarche globale. Márcio Teruya a réalisé le premier exemple. Amer Baghdadi a complété les expérimentations et mis en forme ce document.

6.1 Introduction

Dans cette introduction nous présentons les motivations et les objectifs de notre travail. L'état de l'art ainsi que notre contribution seront détaillés. En fin, le plan de ce chapitre sera exposé.

6.1.1 Motivations et objectifs

L'exploration de l'espace de solutions architecturales est une composante primordiale dans un flot de conception conjointe matérielle/logicielle. Le problème à résoudre dans le flot de conception des systèmes complexes consiste à trouver la meilleure architecture du système incluant le découpage fonctionnel du système, la détermination des protocoles de communication, la topologie du réseaux de processeurs et le placement/ordonnancement. Pour un système de ***n*** processus, le nombre de partitions différentes que l'on peut construire est donné par l'équation (6-1).

$$U_n = \sum_{q=1}^{n} \sum_{i=0}^{q} \frac{(-1)^{q-i}.(i)^n}{(q-i)!.(i)!} \qquad (6\text{-}1)$$

Si maintenant nous supposons que nous disposons de ***p*** types de processeurs différents (matériels/logiciels), le nombre d'architectures possibles sera donné par l'équation (6-2).

$$V_n^p = \sum_{q=1}^{n} p^q \sum_{i=0}^{q} \frac{(-1)^{q-i}.(i)^n}{(q-i)!.(i)!} = \sum_{i=1}^{n} i^n \frac{p^i}{i!} \sum_{j=0}^{n-i} \frac{(-p)^j}{j!} \qquad (6\text{-}2)$$

La Figure 48 montre la variation du nombre d'architectures possibles quand le nombre de processus et le nombre de types de processeurs varie de 1 a 10. A titre d'exemple, pour un modèle composé de 10 processus et une technologie comportant 3 types de processeurs (deux types de microprocesseurs pour le logiciel et la réalisation matérielle) l'espace des solutions contient $4{,}872.10^7$ éléments. Nous remarquons que ce nombre s'accroît exponentiellement avec ***n*** et ***p***. Si nous considérons encore que nous disposons de ***c*** protocoles de communication, le nombre d'architectures sera encore plus énorme. En plus, le temps de synthèse et de cosimulation au niveau RTL d'une architecture peut prendre quelques jours. Donc, nous ne pouvons pas se permettre de synthétiser et de cosimuler, au niveau RTL, chaque architecture pour évaluer sa performance. Ces chiffres ont été la base de notre motivation pour les travaux présentés dans ce chapitre.

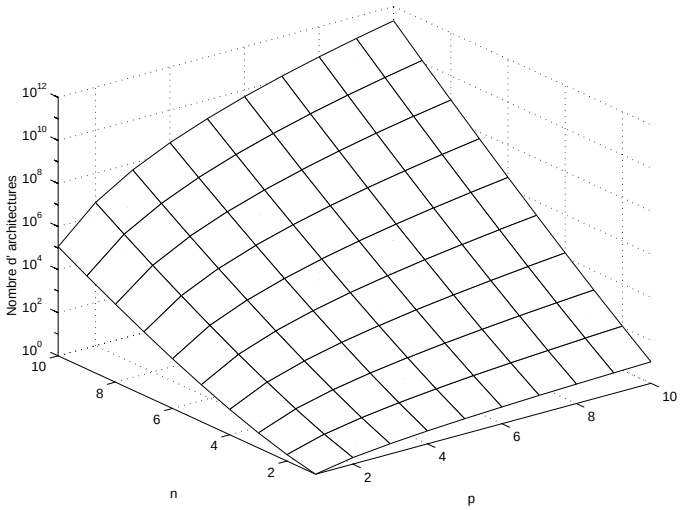


Figure 48 – Le nombre d'architectures possibles

D'ici vient le besoin d'un outil d'estimation de performances suffisamment rapide pour effectuer cette tache d'exploration d'architectures. Un outil d'estimation au niveau système qui soit assez précis sera un très bon candidat pour répondre à nos besoins de rapidité et de précision. La combinaison d'un tel outil avec un outil de codesign et un outil de cosimulation constituera un environnement parfait pour la conception conjointe matérielle/logicielle de systèmes complexes. L'objectif est, donc, de trouver une méthode d'estimation de performance qui soit rapide et précise et qui s'intègre facilement dans un flot de codesign pour assister le concepteur dans son choix architectural.

6.1.2 L'état de l'art concernant l'estimation de performance

Les méthodes d'estimations que l'on trouve dans la littérature peuvent être classées dans trois catégories: statiques, dynamiques et mixtes :

- **Dynamique :** les mesures de performance d'une architecture est le résultat de l'exécution d'un modèle (exemple : simulation).
- **Statique :** l'estimation de performance d'une architecture est le résultat d'une analyse statique d'une spécification (exemple : analyse de chemins dans une spécification de flot de contrôle).
- **Mixte dynamique/statique :** c'est l'utilisation de quelques éléments des deux approches précédentes pour l'analyse de performance d'une architecture.

Les approches dynamiques sont en général très précises. Leur inconvénient majeur est le temps nécessaire pour l'obtention du modèle à simuler (synthèse, génération, compilation),

ainsi que le temps de la simulation. Ce qui les rend, en pratique, inutilisable dans le contexte particulier de l'exploration où le nombre de modèles à analyser est énorme. D'un autre coté, les approches statiques sont certes très rapides (pas de génération de modèles à simuler, ni de simulation), mais les tâches de modélisation et d'estimation sont complexes à cause de la distance qui sépare les concepts de spécification de l'implémentation.

D'autre part, nous trouvons dans la littérature très peu de travaux visent l'analyse de performance en vue de l'exploration d'architectures pour la conception conjointe matérielle/logicielle. Cependant, beaucoup de travaux ont étés réalisés pour résoudre des problèmes séparés comme l'estimation du temps d'exécution du logiciel, l'analyse de performance des circuits ASIC ou encore l'architecture de systèmes complexes. Nous pouvons classer les travaux existants dans le domaine de l'analyse de performance en vue de l'exploration d'architectures pour la conception conjointe matérielle/logicielle dans deux catégories selon la complexité de l'architecture cible : les travaux visant des architectures cibles monoprocesseur et les travaux visant des architectures cibles multiprocesseurs.

6.1.2.1 Les travaux visant des architectures cibles monoprocesseur

Dans cette catégorie on peut citer PMOSS [EHGR96], COSYMA [YeEr95][HeEr95] and LYCOS [MGKP97]. L'architecture cible est monoprocesseur (une seule unité de contrôle). Il y a donc pas de difficultés liées au parallélisme par rapport aux architectures multiprocesseurs. Cependant, les analyses de performance des parties logicielles et matérielles sont réalisées conjointement.

PMOSS [EHGR96] se contente de calculer l'accélération due au co-processeur (partie matérielle), sur la performance globale du système. Pour cela, il utilise, pour le logiciel, des analyses statiques (calcul du temps d'exécution basé sur le code assembleur) et dynamiques (profilage). Pour le matériel, il utilise des analyses statiques (calcule du temps d'exécution basé sur la description de la machine de contrôle). Et pour les communications, des analyses dynamiques (profilage), sont utilisées.

COSYMA [YeEr95][HeEr95] calcule des métriques séparées pour le logiciel, le matériel et la communication. En suite, ces métriques sont combinées dans des équations particulières pour procéder à une partition basée sur une méthode de recuit simulé (pour *simulated annealing*). Des mesures de temps dans le pire cas sont calculées pour les implémentations logicielles et matérielles en utilisant plusieurs variantes de techniques

d'analyse de chemins. Le temps de communication est estimé pour un modèle particulier (mémoire partagée).

LYCOS [MGKP97] procède à des estimations de performance en utilisant des techniques de profilage et d'estimations de temps d'exécution à bas niveau pour le matériel, le logiciel et la communication.

Malgré leur performance, ces méthodes ne permettent pas de traiter des architectures complexes pouvant contenir plus qu'un seul processeur.

6.1.2.2 Les travaux visant des architectures cibles multiprocesseurs

Dans cette catégorie nous trouvons SpecSyn [GVNG98], Polis [SSV96] et la méthode créée par Yen et al [Ywolf95]. L'architecture cible est multiprocesseur complexe.

SpecSyn [GVNG98] admet des architectures avec un nombre quelconque de microprocesseurs et de co-processeurs. L'approche utilisée pour l'estimation de performance est mixte statique/dynamique. Elle est faite en deux étapes :

- Pre-estimation : elle est réalisée avant la phase d'exploration d'architectures. Un profilage de la description du système est réalisé pour obtenir des temps d'exécution pour différents niveaux (processus, bloc de base, communication).
- Estimation en ligne : elle est faite durant la phase d'exploration d'architectures. Les résultats obtenus durant la phase de pre-estimation sont utilisés dans des expressions complexes pour le calcul de la performance globale du système.

Le problème d'une telle approche est son incapacité à capturer les changements dynamiques du comportement temporel durant la phase d'exploration d'architectures. Car durant cette phase, des méthodes statiques sont utilisées (le temps global est la somme des temps d'exécution partiels des différentes ressources d'exécution). Par exemple, cette méthode n'est pas capable d'estimer le temps d'attente d'un processus pour qu'un autre finisse son exécution. Le passage sur un tel comportement dynamique peut introduire une grande imprécision sur les résultats de l'estimation.

Polis [SSV96] est capable de surmonter le problème mentionné ci-dessus (capture du comportement dynamique), en combinant une simulation de haut niveau avec des estimations de bas niveau (approche statique/dynamique). Cette approche est similaire à notre méthodologie, mais l'architecture cible de Polis est beaucoup moins complexe (un seul

microprocesseur et plusieurs co-processeurs). En plus, le niveau d'abstraction du langage de spécification à l'entrée de Polis est plus bas que dans le cas de notre approche.

Yen et al [Ywolf95], attaquent le problème d'un point de vue générique. Ils analysent, au niveau système, l'interaction entre les différents processus en donnant le meilleur et le pire délai pour chacun d'entre eux. En suite, en partant d'un graphe acyclique représentant les dépendances de données entre les processus, et à l'aide d'informations sur le partitionnement (la distribution sur les unités de traitement), ils calculent le temps d'exécution, dans le pire cas, pour le système entier. Cette méthode est précise et capable de prendre en compte les délais de communications. Malheureusement, elle est limitée à des applications pour lesquelles il est suffisant de connaître les délais dans le pire cas. De plus, les processus doivent être représentables par graphes acycliques.

L'estimation de performance peut être faite sur plusieurs niveaux d'abstraction. En effet, dans un environnement de conception conjointe matérielle/logicielle, nous partons d'un niveau d'abstraction système pour arriver au niveau d'implémentation RTL. Au niveau RTL, l'analyse de performance se caractérise par une grande précision, mais elle consomme beaucoup de temps. En remontant dans les niveaux d'abstraction, le temps de l'analyse de performance diminue, mais la précision diminue également. Aussi, avec l'écart important entre les deux niveaux d'abstraction : système et RTL, l'analyse de performance au niveau système peut devenir très imprécise voir inexploitable.

6.1.3 Contribution

Afin d'exploiter les avantages de toutes les approches, nous proposons de combiner les deux niveaux d'abstractions (niveau système et niveau implémentation), tout en utilisant un modèle hybride statique/dynamique. Notre objectif est d'obtenir une méthodologie d'estimation de performance au niveau système à la fois rapide et précise. L'idée sous-jacente est d'utiliser, d'une part, l'approche dynamique pour l'évaluation des tâches qui ne peuvent être déterminées qu'à l'exécution (événements dépendants des données). D'autre part, utiliser l'approche statique (modélisation et estimation), pour l'évaluation du reste du système.

Une nouvelle méthode pour l'estimation du temps d'exécution d'une application à partir de sa spécification au niveau système a été développée. Cette méthode nous permet d'effectuer l'exploration de l'espace d'architectures au niveau système. Elle s'intègre bien dans le flot du système de codesign MUSIC (voir Figure 49). Dans ce flot nous partons d'une

spécification du système en SDL (SDL_1) et nous générons une architecture matérielle/logicielle. Notre méthode d'estimation/exploration est faite en deux étapes :

1. **La génération de la bibliothèque de performance :** le processus de conception conjointe doit être exécuté deux fois pour produire les deux réalisations de référence : tout en logiciel et tout en matériel. Le précalcul des délais d'exécution partiels en fonction de la technologie est réalisé à partir de ces deux réalisations de référence. Ces données sont stockées dans une bibliothèque de performance ;
2. **L'exécution du modèle de performance :** le partitionnement logiciel/matériel et les choix des canaux de transmission sont employés pour créer un modèle de performance exécutable. Les informations de délai de la bibliothèque de performance sont annotées dans ce modèle selon les choix réalisés pendant le partitionnement logiciel/matériel ;

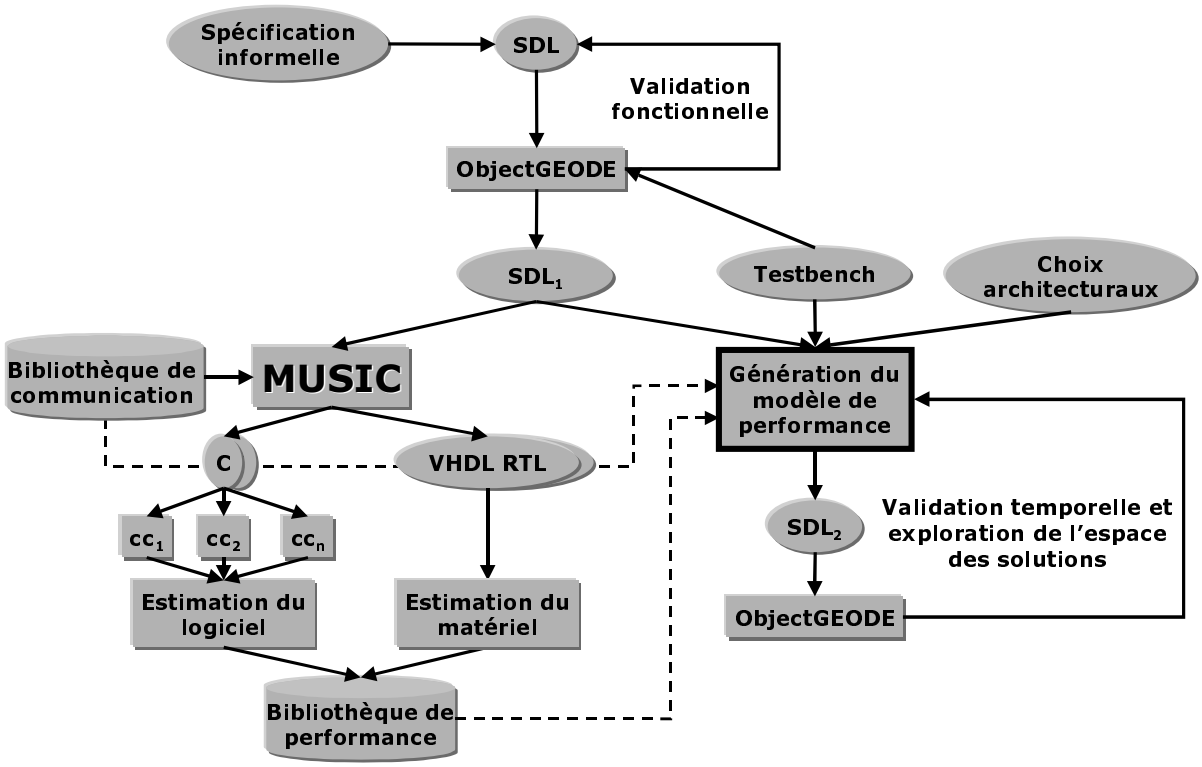


Figure 49 – Le flot d'estimation de performance de MUSIC

L'estimation de performance pour une architecture donnée consiste à enrichir la spécification du système en SDL par des annotations temporelles extraites d'un passage de cette spécification dans l'outil de codesign MUSIC. La nouvelle spécification annotée (SDL_2), et l'outil de simulation d'ObjectGEODE (de **Verilog**) nous permettent l'exploration de

l'espace des architectures d'une façon très rapide et avec une grande fidélité. La méthode d'estimation/exploration que nous proposons est limitée au critère de vitesse d'exécution. Cependant, elle peut facilement s'étendre à d'autres critères de performance. De même, elle peut s'appliquer à d'autres environnements de codesign.

Le reste de ce chapitre est organisé en quatre sections. Dans la section 6.2, nous allons présenter l'environnement de codesign dans lequel nous avons développé et validé notre méthodologie d'estimation/exploration. Cette méthodologie sera détaillée dans la section 6.3. La validation de cette méthodologie par un exemple d'application sera présentée dans la section 6.4. Une analyse appropriée des résultats sera également présentée dans cette section.

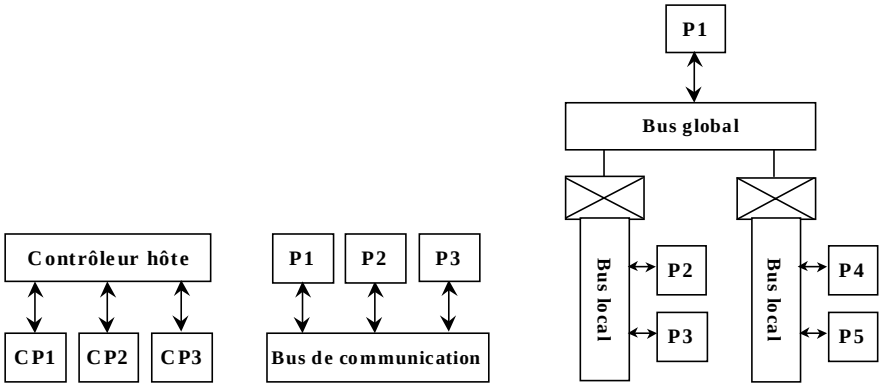
6.2 Méthodologie de codesign basée sur SDL

Dans cette section nous présentons l'environnement de conception dans lequel nous avons développé et validé notre méthodologie d'estimation/exploration. Nous exposerons tout d'abord les modèles architecturaux utilisés dans le monde de codesign et par MUSIC. Ensuite, nous allons donner une brève présentation du langage SDL. En fin, nous présentons brièvement l'environnement d'ObjectGEODE, et nous décortiquons le module d'analyse de performance introduit récemment au simulateur SDL.

6.2.1 Modèles d'architectures pour le codesign

Nous allons présenter une aperçue générale sur les modèles architecturaux envisageables dans le cadre du codesign. L'espace d'architectures envisageables dans la conception de systèmes complexes est très large. D'une manière générale, une architecture se constitue de trois parties : *composants, liens* et *schéma d'organisation*. Les composants sont les processeurs logiciels (microprocesseur, microcontrôleur), les processeurs matériels (co-processeur, périphérique, interface) et les mémoires. Les liens sont les fils, les canaux abstraits ou encore des protocoles spécifiques.

La plus part des outils de conception conjointe matérielle/logicielle traitent des architectures cibles simplifiées. Ces architectures sont en général monoprocesseur. Elles contiennent un seul microprocesseur (contrôleur hôte), qui prend en charge la coordination des activités des autres composants (co-processeurs, voir Figure 50a). Malgré la grande utilité de telles architectures dans plusieurs domaines d'applications, leur performance reste limitée par le type de parallélisme fourni.



a) Architecture monoprocesseur

b) Multiprocesseurs monobus

c) Multiprocesseurs multibus

Figure 50 – Les modèles d'architectures pour la conception conjointe matérielle/logicielle

Très peu d'outils de conception abordent des architectures multiprocesseurs comme celle montrée Figure 50b. Ce modèle est plus flexible et plus performant grâce à la distribution du calcul sur les différents processeurs. Cependant, il reste toujours quelques restrictions dues à l'utilisation d'un seul bus de communication. Le nombre de microprocesseurs et de co-processeurs n'est limité que par la capacité du bus. MUSIC utilise une architecture cible de ce type.

Nous travaillons actuellement à étendre le spectre d'architectures supportées par MUSIC pour couvrir des modèles plus généraux, comme celui montré à la Figure 50c. Dans ce modèle, une architecture est composée de plusieurs processeurs communicants via un réseau de communication très sophistiqué. Cette architecture (multiprocesseurs, multibus), est très flexible, aucune limitation n'est imposée sur le schéma de communication.

Dans la suite, un processeur désignera soit un processeur logiciel (microprocesseur, microcontrôleur) soit un processeur matériel (ASIC). L'estimation de performance et l'exploration d'architectures sont particulièrement importants pour les outils de conception conjointe matérielle/logicielle. La section 2.2 a présenté, brièvement, l'outil de conception conjointe matérielle/logicielle, MUSIC. Cet outil est utilisé pour la validation de notre méthodologie d'estimation de performance. Il fera partie du flot d'estimation/exploration. Comme nous l'avons déjà cité, la contribution de ce travail sera dans la phase du choix de l'architecture. La combinaison d'un outil d'estimation/exploration d'architectures avec l'outil de conception MUSIC constituera un environnement parfait pour la conception conjointe matérielle/logicielle de systèmes complexes.

6.2.2 Le langage SDL

Le langage SDL (pour *Specification and Description Language*) [FaeO94], est dédié à la modélisation et à la simulation des systèmes temps réel distribués pour les télécommunications. Il est standardisé par l'ITU [ITU93]. Un système décrit en SDL est composé d'un ensemble de processus concurrents communicants à travers des signaux. Le langage SDL supporte les différents concepts permettant la description des systèmes (la structure, le comportement et la communication).

6.2.2.1 La structure du langage SDL

La structure statique d'un système décrit en SDL est hiérarchique. L'entité la plus haute de la hiérarchie est appelée « *système* ». Une instance de système contient un ensemble d'instances de « *blocs* ». Une instance de bloc peut contenir d'autres instances de blocs ou un ensemble d'instances de « *processus* » en utilisant une hiérarchie de blocs. Un bloc peut contenir d'autres blocs ou bien un ensemble de processus. Les différents processus d'un même bloc sont connectés entre eux et jusqu'à la frontière du bloc par des « *routes* ». Les blocs sont connectés entre eux par des « *canaux* ». Les routes et les canaux sont des vecteurs de « *signaux* ». Les signaux échangés par les processus suivent un chemin composé de routes et de canaux. SDL offre, également, des structures dynamiques tel que la création et la destruction de processus, ou l'adressage dynamique. La Figure 51 illustre la structure d'un système SDL.

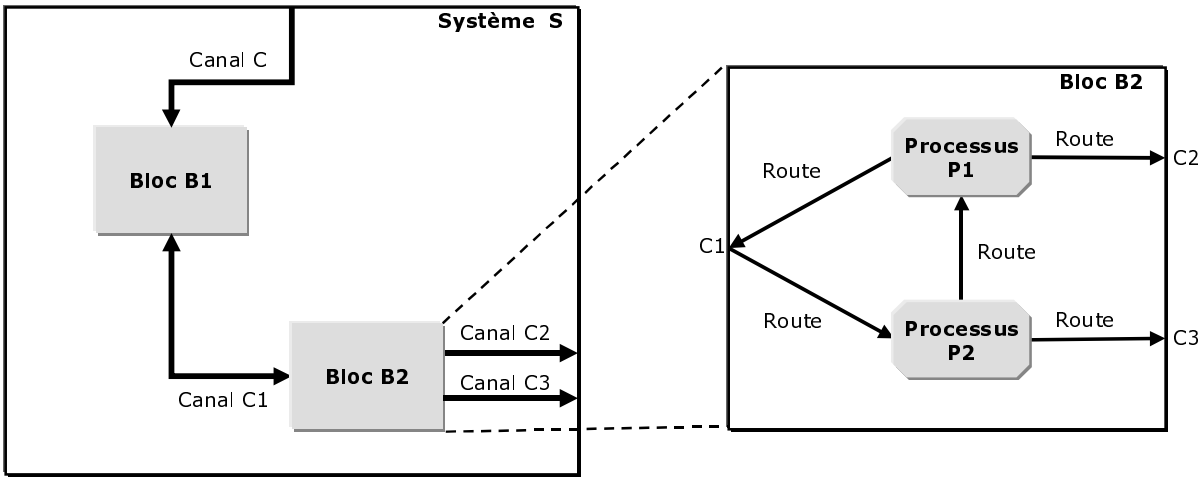


Figure 51 – La structure d'un système SDL : blocs, processus, routes, canaux et signaux

6.2.2.2 Le comportement du système

Le comportement d'un système est représenté par la combinaison des comportements de l'ensemble de processus autonomes et concurrents du système. Un processus est décrit par un automate d'états finis qui communique avec les autres processus, à travers des signaux, de manière asynchrone (voir Figure 52). Chaque processus est composé d'un ensemble d'états et de transitions. Il possède une « *file d'attente* » en entrée, de type FIFO (*First-In-First-Out*) de taille infinie, dans la quelle les signaux sont stockés à leur arrivée. L'arrivée d'un signal attendu dans la file d'attente détermine et valide la transition à exécuter. Le signal qui a initié la transition est retirée de la file d'attente et le processus peut alors exécuter un ensemble d'opérations telles que la manipulation de variables, des appels de procédures ou émission de signaux.

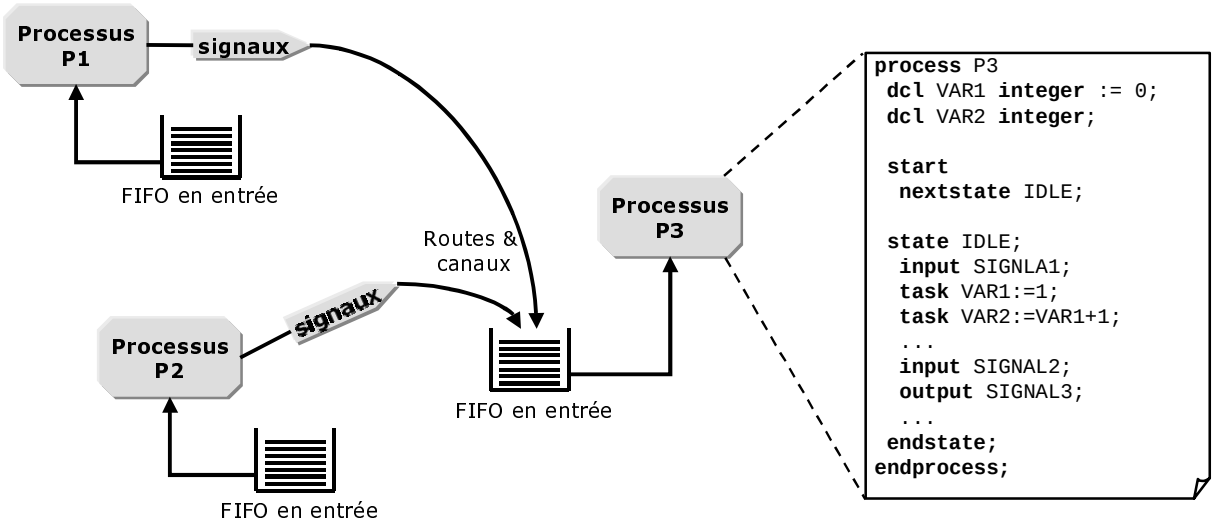


Figure 52 – Le comportement d'un système SDL

6.2.2.3 La communication inter-processus

La communication inter-processus est basée sur le modèle de « *passage de messages* ». Les signaux sont l'unique moyen de synchronisation inter-processus. Un signal transporte toujours implicitement l'adresse du processus émetteur, l'adresse du destinataire si elle est spécifiée explicitement ainsi qu'un ensemble éventuel de paramètres. Deux autres concepts sont utilisés comme support de transfert des signaux : les routes et les canaux. Les routes assurent la connexion des processus du même bloc et se terminent à la frontière du bloc. Les canaux connectent les blocs entre eux. Tous les deux peuvent être unidirectionnels ou bidirectionnels. Pour réaliser une communication entre deux processus situés dans des

blocs différents, les signaux doivent emprunter les canaux et les routes. Un canal peut être connecté à plusieurs routes, alors qu'une route ne peut être connectée qu'à un seul canal. Une communication à travers une route s'effectue en un temps nul, alors qu'à travers un canal, elle s'effectue en un temps non déterministe. Les délais et l'ordre d'arrivée des signaux utilisant deux chemins différants ne peuvent pas être prédits.

6.2.3 L'estimation de performance à partir du modèle SDL

Grâce au module d'analyse de performance qui a était introduit dans le simulateur d'ObjectGEODE, il est devenu possible d'utiliser ce simulateur pour l'évaluation de performance d'un système décrit en SDL. Ici, nous introduisons l'environnement utilisé pour le développement de modules en SDL, ObjectGEODE de **Verilog**. Nous allons particulièrement décortiquer le module d'analyse de performance.

6.2.3.1 L'environnement ObjectGEODE

L'environnement d'ObjectGEODE résulte de la combinaison de l'environnement GEODE pour SDL et de l'environnement LOV pour OMT (pour *Object Modeling Technique*) [Rumb91], tous les deux en provenance de la société **Verilog**. ObjectGEODE est composé des outils suivants :

- Un environnement SDL, incluant un éditeur graphique, un simulateur et un générateur de code.
- Un environnement OMT, incluant un générateur de squelette C++.
- Un éditeur MSC (*Message Sequence Chart*, International Telecommunication Union (ITU) recommandation Z.120).
- Un outil de suivi de projet, permettant de diviser une description OMT en plusieurs fichiers.

6.2.3.2 L'environnement SDL

L'éditeur fournit une interface graphique de développement de systèmes en SDL. Le simulateur s'appuie sur la sémantique du langage SDL (ITU recommandation Z.100 et Z105). Cependant, quelques différences existent lorsque les recommandations Z.100 (ou Z.105) sont ambiguës, et lorsque l'interprétation par le simulateur implique l'introduction de concepts ou

de vocabulaires qui ne sont pas décrites dans la recommandation Z.100. Exemples de telles différences :

- La concurrence est simulée par l'exécution entrelacée des différents processus.
- Les types de données sont interprétés selon le modèle du langage Pascal à la place du modèle algébrique proposé dans les recommandations Z.100 et Z.105.
- Les extensions SDL.

Le simulateur offre les options suivantes :

- La trace d'une simulation peut être conservée pour être exécutée de nouveau.
- La commande « *undo* » permet de revenir en arrière dans l'exécution.
- Les extensions SDL :

  → Les modes de communications (rendez-vous, diffusion et diffusion sélective).

  → Le module d'analyse de performance.

  → La possibilité d'intégrer de code externe sous la forme de types et fonctions en **C** (types de données abstraites).

6.2.3.3 Le module d'analyse de performance

Le langage SDL (recommandation Z.100 et Z105) ne contient pas de notions de temps. Aussi, les canaux de communication sont basés sur un protocole FIFO de taille infinie. Donc, le langage, tel que définit par les standards, n'est pas adapté à l'estimation de performance ni à la modélisation d'architectures. Cependant, un module d'analyse de performance a été développé comme un nouveau composant du simulateur d'ObjectGEODE. Il s'agit de nouvelles extensions SDL qui ne sont pas reconnues que par le simulateur d'ObjectGEODE, et qui permet à ce dernier d'exécuter des commandes spécifiques pour l'évaluation de performance et la modélisation d'architectures.

6.2.3.3.1 L'Introduction de la performance dans le langage SDL

Pour que l'introduction de la performance dans le langage SDL n'affecte pas les outils déjà existants (par exemple : l'éditeur SDL), les extensions SDL ne doivent pas modifier la syntaxe standard de SDL. Ceci est réalisable grâce aux directives (COMMENT phrases), qui sont reconnues uniquement par le simulateur d'ObjectGEODE. La forme générique de ces extensions est la suivante :

```
<special comment> ::=   COMMENT  ‘ [ <free comment prefix> ]
                        <directive>
                        [ <free comment suffix> ] ‘
<directive>       ::=   # <directive> [ ( <directive parameters> ) ]
```

Un des points les plus forts de cette approche est que ces directives peuvent être attachées aux actions (TASK) à l’intérieure d’une transition SDL. Cela nous permet de réaliser des estimations relativement précises (à bas niveau) par rapport aux résultats obtenus avec un outil comme OPNET (outil commercial développé par **MIL3**) [Bacq97]. Dans ce dernier, les caractéristiques de performance sont attachées à des entités de haut niveau, par exemple : les blocs en SDL.

Du point de vue sémantique, les nouvelles extensions de SDL ont un sens beaucoup plus puissant que les « *timers* » qui existent déjà en SDL. Ces derniers donnent l’accès à une horloge de référence globale pour mesurer le temps actuel (global). En plus, ils ne permettent pas de modéliser le partage de ressources d’exécution entre les différentes entités SDL. Un tel partage est devenu modélisable grâce à la directive NODE. Dans la section suivante nous présentons les nouvelles directives NODE, PRIORITY et DELAY.

6.2.3.3.2 Les nouvelles directives pour l’analyse de performance

La directive NODE permet d’identifier les ressources d’exécution d’un modèle, que l’on appelle « *nœuds* ». Un nœud peut être associé aux systèmes, types de système, blocs, types de bloc, processus, types de processus. Tous les processus à l’intérieure d’un nœud partagent les mêmes ressources d’exécutions. Les processus qui ne sont pas à l’intérieure d’un nœud (explicitement), sont considérés comme des nœuds (par défaut). La syntaxe de cette directive est la suivante :

```
<node directive> ::= #node
```

La directive PRIORITY permet d’assigner un ordre de priorité à chaque processus SDL, à l’intérieur d’un même nœud, pour ordonnancer leur exécution. Le choix d’une transition entre toutes les transitions valides de toutes les instances de processus à l’intérieur d’un nœud (i.e. la stratégie d’ordonnancement) suit une distribution aléatoire uniforme. La directive PRIORITY peut être associée à un processus ou à un type de processus, à l’intérieur d’un nœud pour changer ce choix aléatoire. La syntaxe de cette directive est la suivante :

```
<priority directive> ::= #priority (<integer constant expression>)
```

La Figure 53 (SDL en format textuel), montre comment on peut utiliser les directives NODE et PRIORITY. A la simulation, le processus P1 commence toujours le premier car il a une priorité supérieure à celle du processus P2 (par défaut, c'est la priorité la plus petite, zéro). Le processus P2 peut commencer l'exécution seulement quand le processus P1 s'arrête.

```
SYSTEM sys;
  /* BLOCK B1 : P1 et P2 */
  BLOCK B1   COMMENT '#node';
    PROCESS P1 COMMENT '#priority (1)';
...                                          Processus 1
    ENDPROCESS;
    PROCESS P2;
...                                          Processus 2
    ENDPROCESS;
  ENDBLOCK;
ENDSYSTEM;
```

Figure 53 – Les directives NODE et PRIORITY

La directive DELAY est associée aux actions (TASK) en SDL, pour spécifier le temps d'exécution de l'action. Par défaut, le temps d'exécution d'une action est nul. Lors de la simulation, l'arrivée à une action annotée par cette directive cause le blocage du nœud contenant cette action pendant un temps spécifié comme paramètre de la directive. Après ce retard de temps (temps global d'exécution progresse toujours), l'action est exécutée. La syntaxe de cette directive est la suivante :

```
<delay directive>  ::= #delay ( <duration expression> )
        | #delay ( <duration expression 1>, <duration expression 2>
              [ , <distribution> ] )
        | #delay ( <duration expression>,  *  [ , <distribution> ] )
<distribution> ::= <distribution NAME> [ ( <constant  expression  list> ) ]
```

La durée de l'action est spécifiée comme paramètre :

- *delay (<duration expression>)* : la durée de l'action est spécifiée par *<duration expression>* ;
- *delay (<duration expression 1>, <duration expression 2>)* : la durée est choisie aléatoirement entre *<duration expression1>* et *<duration expression2>* ;

- *delay (<duration expression>, *)* : la durée est choisie aléatoirement entre *<duration expression>* et une valeur pseudo infinie, définie par une variable d'environnement du simulateur. Cette valeur doit être choisie soigneusement selon les contraintes temporelles définies dans le modèle. Le choix aléatoire de la durée est pris selon la *<distribution>* spécifiée. Les distributions possibles sont les suivantes :

  → Distribution uniforme (par défaut) ;

  → Distribution exponentielle ;

  → Distribution normale.

Les expressions de délai dans une directive DELAY peuvent être actives (i.e. contenir des variables). Donc les contraintes sur la durée d'une action peuvent varier selon les variables manipulées dans d'autres actions. Par exemple, la durée d'une instruction « *output* » peut varier selon les valeurs des paramètres du signal envoyé.

Il y a quelques restrictions dans l'utilisation de la directive DELAY en SDL. La spécification des délais avec une distribution aléatoire empêche l'utilisation de quelques caractéristiques du simulateur, telles que les conditions « *filter* », « *undo/redo* », « *replay* » d'un scénario et les simulations exhaustives.

6.3 La méthodologie d'estimation et d'exploration

Le principe de notre méthodologie est de combiner les deux niveaux d'abstractions (niveau système et niveau implémentation), tout en utilisant une approche hybride statique/dynamique. Le but est d'obtenir un compromis entre la rapidité d'exécution et la fidélité, les deux points de conflit dans une approche d'estimation de performance. La rapidité est assurée par l'utilisation d'une simulation au niveau système. La fidélité est obtenue aussi par la simulation au niveau système (prise en compte des comportements dynamiques), mais aussi par des informations de bas niveau sur quelques implémentations du système. Dans la suite, nous allons donner une vue d'ensemble sur le cadre dans lequel nous avons appliqué notre méthode, ensuite, notre flot d'estimation/exploration sera présenté et les différentes étapes du flot seront détaillées.

6.3.1 Le flot d’estimation/exploration

L’idéal pour avoir une très grande précision, serait d’utiliser la simulation pour chaque événement dépendant de l’exécution, et ceci au niveau d’implémentation. Malheureusement, cette approche n’est pas faisable dans notre contexte, où nous cherchons à explorer largement l’espace d’architectures. Donc, nous avons décidé d’utiliser une simulation au niveau système, beaucoup plus rapide. Avec une telle simulation, nous pouvons estimer les comportements dynamiques dus à l’interaction entre les différents processus, indépendamment de la complexité de l’architecture. Pourtant cette simulation de haut niveau ne contient pas d’information sur la performance de l’implémentation. Ici vient l’importance de l’approche statique et des implémentations de très bas niveau. La réalisation et l’analyse de quelques implémentations nous permettront de récupérer des résultats partiels de performance. Ces résultats seront utilisés pour instrumenter la spécification de niveau système à simuler. Avec la nouvelle spécification instrumentée nous pouvons prédire la performance de toutes les architectures réalisables (indépendamment de leur complexité), avec une fidélité et une rapidité permettant une exploration efficace de l’espace des solutions.

Nous avons développé cette méthodologie d’estimation/exploration dans le cadre de notre environnement de conception MUSIC. Le langage de spécification au niveau système est SDL. La simulation au niveau système est réalisée par le simulateur ObjectGEODE. Le point de départ dans notre flot d’estimation/exploration est la description du système en SDL. La Figure 54 montre les caractéristiques principales de ce flot.

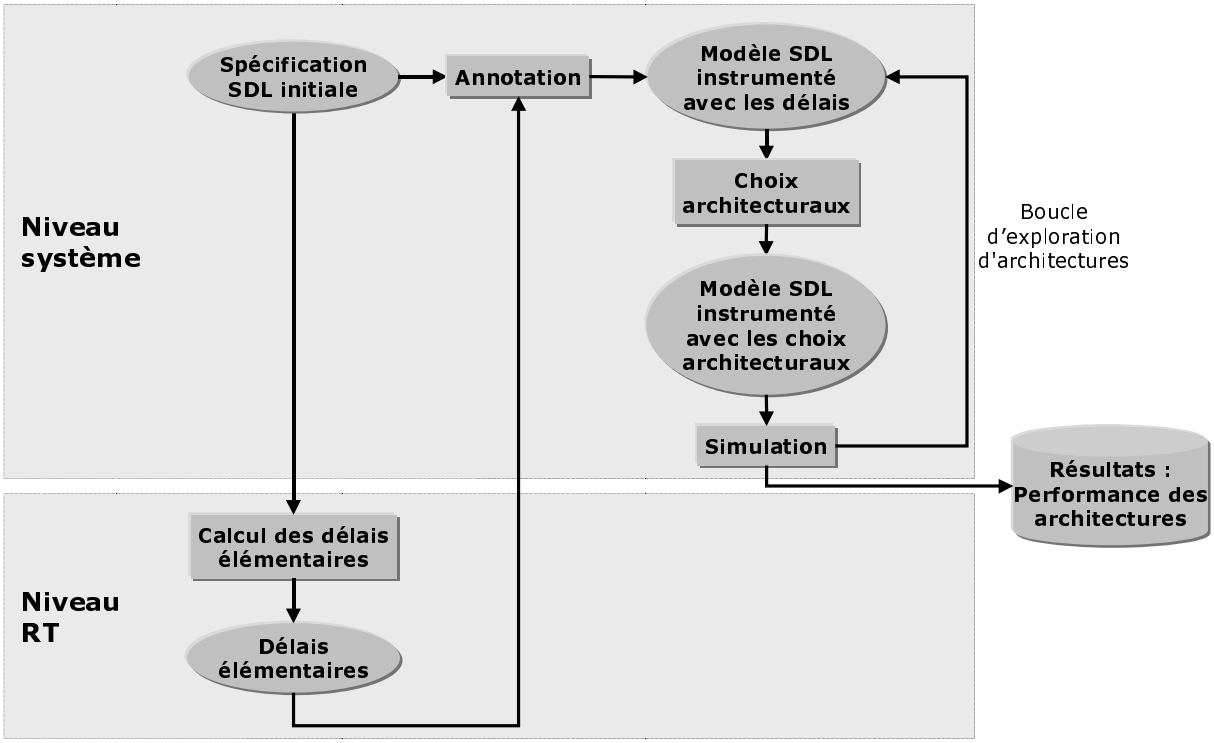


Figure 54 – Le flot global d'estimation/exploration

Nous pouvons facilement identifier les quatre étapes suivantes : calculs des délais élémentaires, annotation, choix architectural et simulation. Nous remarquons qu'il s'agit bien d'une approche hybride où nous utilisons deux niveaux d'abstractions très distants : le niveau système et le niveau RT (ou implémentation). La boucle d'exploration d'architectures se situe au niveau système, d'où la rapidité de cette phase. L'aspect statique de notre approche se manifeste dans les étapes de calcul des délais élémentaires et d'annotation. L'aspect dynamique se matérialise par la simulation et les choix architecturaux. Les quatre étapes que nous avons identifiées dans ce flot ne sont pas de complexité égale. Dans la suite de cette section, nous allons développer ces étapes en détail.

6.3.2 L'estimation des délais élémentaires

C'est la première étape et la plus consommatrice de temps. Cependant, c'est aussi un des points les plus forts de notre approche. Elle est basée sur la propriété du simulateur ObjectGEODE de pouvoir attacher des directives DELAY à des actions (TASKs) à l'intérieure d'une transition SDL. Cela nous permet de réaliser des estimations relativement précises (à bas niveau).

En analysant les techniques utilisées par MUSIC pour la génération de code C/VHDL, et en étudiant le comportement de l'exécution de l'implémentation physique

(ordonnancement, communication), et le modèle d'exécution du simulateur ObjectGEODE, nous arrivons à obtenir les résultats clés suivantes :

1. Les actions décrites en SDL peuvent être identifiées, relativement facilement, dans le code généré.
2. Nous pouvons regrouper une séquence contiguë d'actions qui ne contiennent ni d'instructions de contrôle (branchements et décision), ni d'instructions de communication, pour lui attribuer un seul délai global, sans modifier le comportement de l'exécution. Les séquences d'actions qui suivent la définition ci-dessus, sont appelées « *blocs de base* ». Ce résultat facilite l'estimation des délais élémentaires (leur nombre diminue considérablement), tout en gardant un degré de fiabilité très élevé.
3. Les communications doivent être estimées à part. Une action de communication sera éventuellement modélisée par une séquence de délais pour se rapprocher du modèle d'exécution de l'implémentation physique. Par exemple : modéliser l'ordonnancement des communications entre différents processus qui s'exécutent sur le même microprocesseur.

Il s'agit d'identifier les blocs de base dans la description du système (SDL), pour en suite calculer le délai de chacun d'entre eux et ceci pour toute implémentation envisageable. Les délais des communications sont aussi estimés pour tous les protocoles de communications possibles. Les résultats de cette étape constituent une base de donnée suffisante pour la suite des étapes du flot, et notamment, la boucle d'exploration d'architectures. Les délais élémentaires calculés sont stockés en nombre de cycles. Cela nous permet de mettre en paramètre, la fréquence d'horloge de chaque processeur. Donc, nous gagnions plus de souplesse et de réutilisabilité.

6.3.2.1 Les blocs de base

Ici nous allons présenter le flot que nous utilisons pour l'identification et le calcul du temps d'exécution des blocs de base pour toutes les implémentations envisageables. Dans la littérature, très souvent, on définit le bloc de base comme étant une séquence (maximale) d'instructions contiguës qui ne contiennent pas d'instructions de contrôle (branchements et décisions). Dans notre cas, un bloc de base ne contient pas non plus des instructions de communications. Dans un contexte de conception conjointe matérielle/logicielle, chaque processus peut avoir deux options d'implémentation :

- **Implémentation(s) logicielle(s) :** Le comportement du processus, spécifié en SDL, sera traduit en code C. La performance de l'implémentation logicielle dépend du microprocesseur et du compilateur qui seront utilisés. Donc pour chaque couple (microprocesseur, compilateur), nous générons le code assembleur correspondant (niveau registre). L'analyse statique (ou dynamique) de ce code nous permettra d'analyser sa performance. Le nombre d'implémentations logicielles (d'un processus) possibles dépend du nombre de microprocesseurs et de compilateurs disponibles.
- **Implémentation matérielle :** Le comportement du processus, spécifié en SDL, sera synthétisé en un code VHDL RTL. L'analyse statique (ou dynamique) de ce code est suffisante pour extraire les informations de performance. En effet, le code VHDL est généré par MUSIC, et donc il a une certaine forme spécifique, qui nous permet facilement d'analyser sa performance.

La première chose à faire est d'identifier les blocs de base dans la description SDL. La correspondance entre les actions en SDL et les instructions du code généré par MUSIC est très facile. Pour pouvoir identifier les blocs de base dans tous les niveaux d'abstraction (assembleur compris), d'une façon facile et automatisable, nous avons choisi d'ajouter des annotations de référence au début et à la fin de chaque bloc de base, dans la description SDL. Ces marques seront retrouvées dans le code généré (assembleur pour le logiciel, et VHDL RTL pour le matériel). Les marques choisies sont des simples affectations de variable. Une affectation est ajoutée au début et à la fin de chaque bloc de base. La valeur affectée indique s'il s'agit d'un début/fin d'un bloc de base, elle indique aussi son numéro. La Figure 55 montre un exemple qui illustre l'identification des blocs de base en SDL.

```
SYSTEM motor /*COMMENT '#node'*/;
  BLOCK PC;
    PROCESS host(1,1);
        SYNTYPE MIndexInt=natural Constants  0:7
        ENDSYNTYPE;
           NEWTYPE memory
            ARRAY (MIndexInt,integer);
           ENDNEWTYPE memory;
           dcl memoire memory;
      START ;
        TASK 'TEE - START' COMMENT '#delay (PC_delay1+0)';
        TASK bb:=65280; /* debut du bb1 0xff00 */
        TASK ind1:=0,
        memoire(0):=10,       memoire(1):=0,
        memoire(2):=0,        memoire(3):=0,    Bloc de base 1
        memoire(4):=10,       memoire(5):=9,
        memoire(6):=0,        memoire(7):=15;
        TASK bb:=65296; /* fin du bb1 0xff10 */
        NEXTSTATE DistCtrl;
      STATE DistCtrl;
        INPUT pid_pc_values via pc_pid.1;
        CALL PC_iPIDPC_sig;
          TASK 'TEE - START' COMMENT '#delay (PC_delay2+0)';
          TASK bb:=65281; /* debut du bb2 0xff01 */
          TASK ind := 0;                        Bloc de base 2
          TASK bb:=65297; /* fin du bb2 0xff11 */
        loop1:
          DECISION ind1 >= 14;
            ( false ):
              DECISION (ind1=7) and (ind > 0);
                ( true ):
                  NEXTSTATE DistCtrl;
                ( false ):
              ENDDECISION;
            ( true ):
              STOP ;
          ENDDECISION;
          DECISION ind1<=7;
            ( true ):
              TASK 'TEE - START' COMMENT '#delay (PC_delay3+0)';
              TASK bb:=65282; /* debut du bb3 0xff02 */
              TASK ind:=ind1;                   Bloc de base 3
              TASK bb:=65298; /* fin du bb3 0xff12 */
            ( false ):
              TASK 'TEE - START' COMMENT '#delay (PC_delay4+0)';
              TASK bb:=65283; /* debut du bb4 0xff03 */
              TASK ind:=ind1-7;                 Bloc de base 4
              TASK bb:=65299; /* fin du bb4 0xff13 */
          ENDDECISION;
          TASK 'TEE - START' COMMENT '#delay (PC_delay5+0)';
          TASK bb:=65284; /* debut du bb5 0xff04 */
          TASK ind1:=ind1+1,temp:=memoire(ind); Bloc de base 5
          TASK bb:=65300; /* fin du bb5 0xff14 */
          OUTPUT pc_pid_value (temp) via pc_pid.1;
          CALL PC_oPCPID_int;
        JOIN loop1;
      ENDSTATE;
    ENDPROCESS;
  ENDBLOCK;
ENDSYSTEM;
```

Figure 55 – L'identification des blocs de base dans une description SDL

Le flot de calcul des délais des blocs de base est présenté sur la Figure 56. En effet, après l'identification des blocs de base en SDL, on procède à deux implémentations en utilisant MUSIC. La première est complètement logicielle. La seconde est complètement matérielle. L'implémentation logicielle se devise en plusieurs implémentations selon le

nombre de microprocesseurs et de compilateurs disponibles. Les canaux de communications ne sont pas implémentés (pour simplifier l'analyse des blocs de base).

Finalement, les codes générés (assembleurs et VHDL RTL) sont inspectés pour identifier les blocs de base et estimer la performance de chacun d'entre eux. Donc, pour chaque bloc de base (SDL), nous allons obtenir plusieurs délais. Un délai correspond à l'implémentation matérielle, et le reste correspond aux différentes implémentations logicielles. Actuellement, les mesures de performance en ce qui concerne la réalisation sont effectuées manuellement. L'automatisation de cette étape ne représente aucune difficulté théorique.

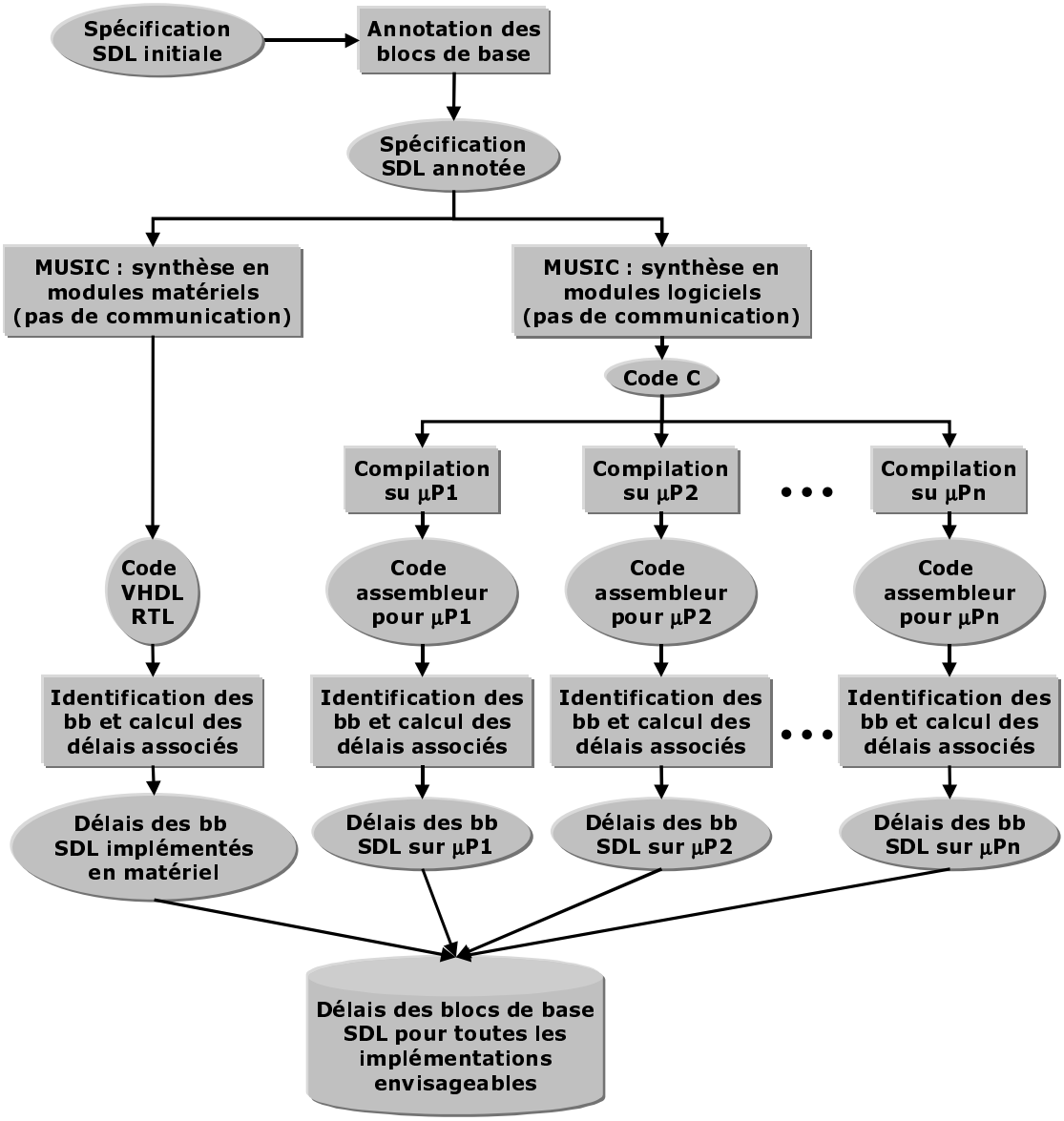


Figure 56 – Le flot de calcul des délais des blocs de base

Nous avons envisagé deux grandes difficultés dans la réalisation de cette étape. La première est de déterminer comment un bloc de base est implémenté. Si la distance (d'abstraction) entre la spécification du bloc de base (en SDL) et son implémentation est très grande, et si beaucoup d'optimisations ont étés réalisées entre ces deux niveaux, il sera difficile de retrouver comment le bloc de base était implémenté. La deuxième difficulté est comment, à partir de l'implémentation (listage assembleur ou code VHDL) d'un bloc de base, calculer son temps d'exécution avec précision. Car, même si au niveau système, un bloc de base ne contient pas de branchement, c'est très probable que ce ne sera plus le cas au niveau implémentation (par exemple : les branchements créés par la dépendance de donnée). Ce comportement dynamique sera mesuré statiquement durant cette étape d'estimation, d'où l'existence de quelques imprécisions inévitables.

Le modèle de synthèse de MUSIC, à l'état actuel, nous aide à alléger ces difficultés car il ne procède pas à des optimisations durant la génération du code C/VHDL. Donc, il est relativement facile de faire la correspondance entre le niveau système et le niveau C/VHDL. Ensuite, si on utilise les compilateurs (pour le C) avec des options d'optimisation allégées, et en utilisant les marques de début/fin de bloc de base, on peut facilement identifier les blocs de base au niveau RT (assembleur/VHDL RTL). La Figure 57 illustre cette correspondance des blocs de base entre les niveaux SDL et assembleur. En conséquence, nous arrivons à surmonter la première difficulté. Par ailleurs les imprécisions de calculs des temps d'exécution peuvent être négligées sans nuire à la qualité de l'estimation.

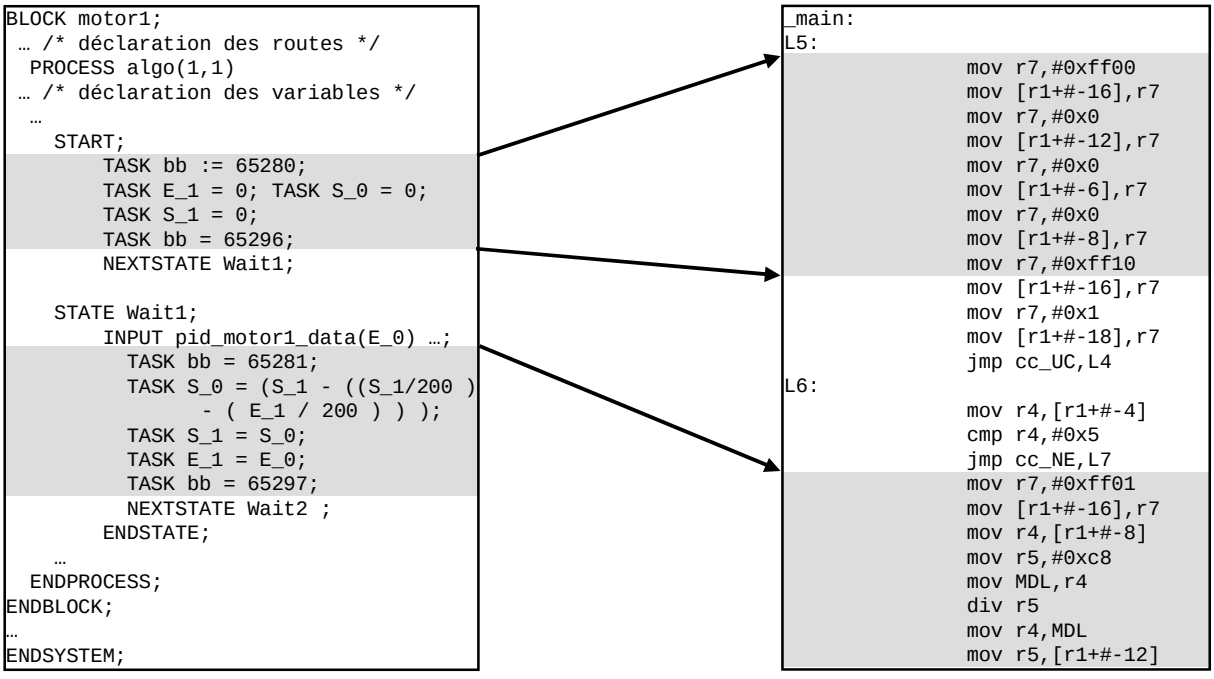


Figure 57 – La correspondance des blocs de base aux niveaux SDL/assembleur

Pour l'estimation de l'implémentation matérielle, MUSIC génère du code VHDL RTL, et à ce niveau d'abstraction, le parallélisme d'opérations se restreint au niveau cycle. Donc le modèle architectural considéré est simple : chaque transition en VHDL RTL dure un cycle d'horloge. Cela facilite énormément cette tâche, car elle est réduite au calcul du nombre de transitions.

L'estimation de l'implémentation logicielle demande des procédures plus sophistiquées. Il faut d'abord utiliser les compilateurs avec des options d'optimisation allégées pour permettre aux annotations introduites en SDL d'apparaître dans le listage d'assembleur et faciliter l'identification des blocs de base. Ayant le code en assembleur, et les spécifications du microprocesseur, on peut calculer la performance d'une séquence d'instructions d'assembleur en ajoutant le nombre de cycles nécessaire pour chaque instruction. Notons que l'estimation des délais à partir du code assembleur dépend fortement de l'architecture du microprocesseur. Pour certains microprocesseurs, cette tache est très simple car une instruction assembleur prend toujours un nombre bien précis de cycles (ex : 8051). Cependant pour d'autres, qui ont des architectures complexes, avec plusieurs niveaux de *pipeline* (ex : Pentium), c'est relativement difficile.

Une autre possibilité, pour ne pas restreindre les options d'optimisation des compilateurs, est d'isoler les blocs de base dans des procédures indépendantes, avant la compilation. Cela nous permet de retrouver les blocs de base quelles que soient les options d'optimisation des compilateurs. Cependant, cette solution est bien plus difficile à mettre en œuvre.

Nous avons noté un autre problème durant l'investigation du code assembleur. A l'intérieure d'une séquence de code assembleur (correspondant à un bloc de base), nous pouvons trouver des appels de procédure de la bibliothèque du compilateur. A l'intérieur de ces procédures on peut trouver des branchements de dépendance de données, ce qui rendent l'estimation de leurs délais plus difficile. La solution adoptée est de pré-calculer un délai moyen pour chacune de ces procédures.

6.3.2.2 Les communications

L'estimation de la communication est traitée à part. En effet, la durée d'une communication peut être divisée en trois parties :

1. **Interface :** C'est le délai nécessaire à l'exécution des instructions des procédures de communication (sans les temps d'attente s'ils existent) à l'intérieure du processeur

(matériel/logiciel). En effet c'est la partie fixe du protocole de communication, qui est la même pour toutes les instances de communications du même protocole. Elle dépend seulement du type de l'implémentation (logicielle/matérielle), elle ne dépend ni des données ni de l'environnement.

2. **Transmission de données :** C'est la partie de la communication qui dépend de la taille des données. Ce délai est complémentaire au délai précédent. Il faut noter que le temps nécessaire pour qu'une donnée traverse les fils physiques de communication (s'ils existent) est pratiquement nul, car nous sommes dans le cadre de circuits embarqués (ou même *system-on-a-chip*).

3. **Attente active :** Il contient tous les délais de blocage du processus tout au long de la communication (ex : le rendez-vous). C'est une partie très variable qui dépend de l'exécution dynamique du système.

Nous remarquons de cette analyse de la communication qu'une grande partie du délai de communication ne dépend pas des données. Aussi, les protocoles de communications dans l'environnement de MUSIC sont décrits dans une bibliothèque à part avec le langage intermédiaire SOLAR. Donc nous avons décidé de créer une bibliothèque d'estimation de la communication.

En effet, l'attente active est prise en compte par le simulateur d'ObjectGEODE. Par exemple pour un protocole rendez-vous, il existe un attribut *rendez-vous* que l'on peut ajouter lors de la déclaration d'une route en SDL, cet attribut provoque le blocage du processus lors de l'initialisation d'une communication (à travers cette route) jusqu'à l'arrivée de l'autre partie (rendez-vous), tout en avançant le temps global (attente active). Il existe d'autres attributs qui nous permettent de modéliser la plus part des schémas de communications.

Par conséquent, la bibliothèque d'estimation de la communication contiendra les délais d'interface et de transmissions de données, mentionnés ci-dessus. Le flot de cette étape est similaire à cela du calcul du temps d'exécution des blocs de base. Le flot est appliqué pour chaque protocole de communication. Cette procédure est décrite sur la Figure 58.

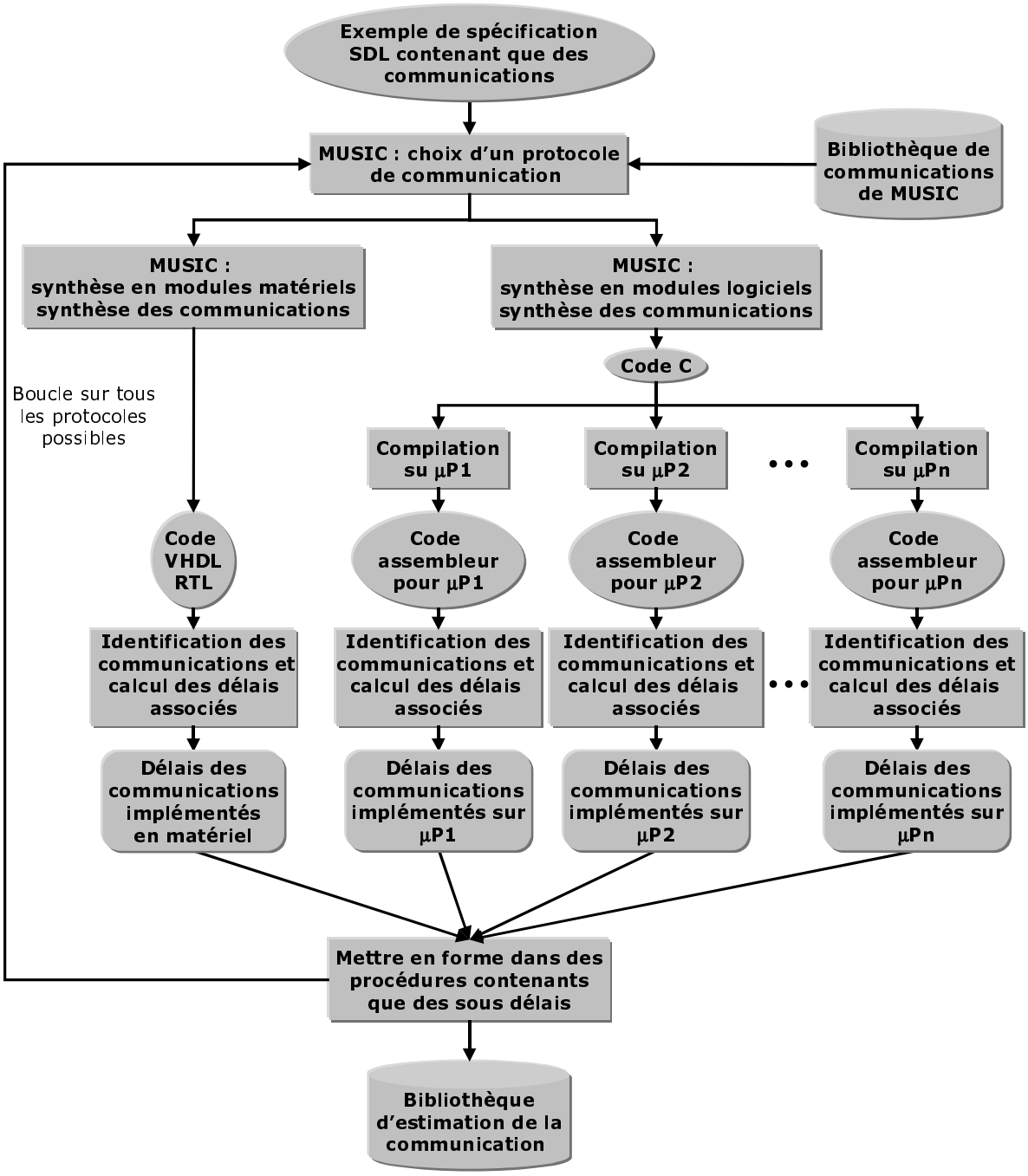


Figure 58 – Le flot de création de la bibliothèque d'estimation de la communication

En effet, il s'agit de créer une description SDL qui ne contient que des primitives de communications. Cette description sera ensuite synthétisée par MUSIC pour chaque protocole de communication de la bibliothèque de MUSIC. Et de la même façon que pour les blocs de base, nous générons toutes les implémentations possibles (matérielle/logicielle). La prochaine étape est l'analyse du code généré et le calcul des délais (comme pour les blocs de base). L'avantage de la génération de cette bibliothèque est qu'elle sera réutilisable par toutes les applications ultérieures.

Les délais de communication (pour un protocole spécifique) seront différents si les deux processus communicants sont sur le même processeur ou sur deux processeurs différents (le délai de l'interface n'est pas le même). Donc il faut implémenter les deux cas (par MUSIC) et calculer les deux types de délais. Pour certains protocoles de communications, les délais calculés ci-dessus peuvent contenir des expressions dépendant de la taille des données pour exprimer le délai de transmissions de données. Ces expressions ne posent pas de problèmes car elles sont supportées pas le module de performance du simulateur d'ObjectGEODE. La façon dont on a représenté les délais dans la bibliothèque est expliquée dans la section suivante.

6.3.3 L'annotation de la spécification SDL

Il s'agit d'instrumenter la spécification SDL avec les délais élémentaires obtenus dans l'étape précédente en utilisant les extensions de SDL introduites par ObjectGEODE. Nous pouvons aussi diviser cette phase en deux parties : annotation des blocs de base et annotations des communications. A la sortie de cette étape, nous allons obtenir une spécification SDL prête à être utilisée dans la boucle d'exploration d'architectures.

6.3.3.1 L'annotation des blocs de base

Pour chaque bloc de base en SDL, nous avons obtenu plusieurs délais correspondants aux différentes implémentations possibles. Ces délais seront introduits dans la spécification SDL grâce à la directive DELAY. En effet, nous déclarons tous les délais trouvés, au début de la spécification SDL comme des constantes. Une action (TASK) est rajoutée au début de chaque bloc de base, pour lui attacher la directive DELAY avec la bonne valeur (selon l'implémentation choisie). Quand le simulateur arrive à cette action, le processus se bloque pendant le délai précisé, tout en avançant le temps global. Ensuite le bloc de base est exécuté avec un temps nul. Donc tout se passe comme si le bloc de base était exécuté avec le délai précisé (délai de l'exécution de son implémentation). La Figure 59 illustre un code SDL où les blocs de base ont été annotés.

```
SYSTEM motor;
  BLOCK PC;
    PROCESS host(1,1);
      START ;
          DECISION ind1<=7;
            ( true ):
              TASK  'delay' COMMENT '#delay (PC_delay3+0)';
              TASK bb:=65282;
              TASK ind:=ind1;                          Bloc de base 1
              TASK bb:=65298;
            ( false ):
              TASK 'delay' COMMENT '#delay (PC_delay4+0)';
              TASK bb:=65283;
              TASK ind:=ind1-7;                        Bloc de base 2
              TASK bb:=65299;
          ENDDECISION;
          TASK 'delay' COMMENT '#delay (PC_delay5+0)';
          TASK bb:=65284;
          TASK ind1:=ind1+1,temp:=memoire(ind);        Bloc de base 3
          TASK bb:=65300;
    ENDPROCESS;
  ENDBLOCK;
ENDSYSTEM;
```

Figure 59 – L'annotation des blocs de base

6.3.3.2 L'annotation des communications

L'annotation de la communication est plus compliquée. Chaque protocole demande une réflexion différente pour pouvoir la modéliser avec les atouts disponibles. Par exemple, pour le protocole rendez-vous, nous devons ajouter l'attribut *rendez-vous* à la route correspondante. Avec cela, les deux actions *output* et *input* deviennent bloquantes. Ensuite, il y a deux possibilités :

1. Si les processus communicants sont sur deux processeurs différents, et donc s'exécutent en parallèle, le délai de communication peut être approché par l'équation (6-3).

$$D_{Comm\_P} = D_{attente\ active} + Moyenne(D_{in}, D_{out}) \qquad (6\text{-}3)$$

où : $D_{in} = D_{(\text{interface} + \text{tarns. données})\ input}$;
$D_{out} = D_{(\text{interface} + \text{tarns. données})\ output}$.

Donc pour estimer cette communication, il suffit de rajouter le délai $Moyenne(D_{in},D_{out})$ après les actions *output* et *input* dans les deux processus communicants. Donc, à la simulation, après une période d'attente active (éventuellement), la communication se fait avec un temps nul, ensuite les processus sont bloqués simultanément pendant la durée $Moyenne(D_{in},D_{out})$ (le temps global avance toujours).

2. Si les deux processus communicants sont sur le même processeur, et donc s'exécutent en série, le délai de communication peut être approché par l'équation (6-4).

$$D_{Comm\_S} = D_{attente\ active} + D_{out} + D_{in} \qquad (6\text{-}4)$$

Donc pour modéliser ce délai, il suffit de rajouter le délai D_{out} après l'action *output*, et le délai D_{in} après l'action *input*, dans les deux processus communicants. Comme les deux processus s'exécutent sur le même processeur, le délai total de la communication sera la somme des délais partiels. Ceci donne une modélisation assez précise et fiable pour ce cas.

Dans le cas où plusieurs processus s'exécuteraient sur un seul processeur, la version actuelle de MUSIC génère un code en utilisant une technique de synthèse de logiciel très particulière. Les flots d'exécution (*threads* en anglais) sont fusionnés par entrelacement. Même les lignes de codes correspondants à la communication sont entrelacées (éventuellement), avec du code d'autres processus. Pour pouvoir modéliser ce schéma d'ordonnancement, il faut diviser les délais D_{in} et D_{out} en unités plus petites qui représentent les délais des blocs de base du protocole de communication. Pour cela nous avons décidé de remplacer D_{in} et D_{out} par des appels de procédures. Ces procédures contiennent simplement une suite de directives DELAY représentant la suite des délais des blocs de base du protocole de communication correspondant. Avec cela, nous pouvons à la simulation suivre le même modèle d'entrelacement implémenté par MUSIC ou même n'importe quel autre modèle d'ordonnancement des processus.

Cette stratégie pour modéliser le protocole rendez-vous est tellement générale et abstraite que l'on peut utiliser pour n'importe quel autre protocole de communication.

6.3.4 Les choix architecturaux

Cette étape introduit les choix architecturaux dans la spécification système pour indiquer, au niveau système, la technologie choisie pour la réalisation des composants. Les choix architecturaux peuvent se résumer en trois : la définition des types et des nombres de microprocesseurs et de co-processeurs, l'attribution des processus aux processeurs et le choix du protocole et du type d'implémentation pour chaque canal de communication.

Trois types d'annotation sont effectués sur la spécification SDL, obtenue de l'étape précédente, pour obtenir une spécification SDL modélisant une certaine architecture. La simulation de cette spécification donne la performance de l'architecture choisie. Cette possibilité de modéliser, au niveau système, le comportement temporel d'une architecture à partir d'une spécification SDL est due aux extensions introduites dans le simulateur d'ObjectGEODE. Les trois modifications sont développées ci-dessous.

6.3.4.1 Le partitionnement du système

Cette tache modélise le partage des mêmes ressources par plusieurs processus. Ceci est faisable grâce à la directive NODE introduite par le simulateur d'ObjectGEODE. Cette directive, qui peut être associée aux déclarations SYSTEM et BLOCK en SDL, cause l'exécution sur un même processeur des processus contenus dans cette hiérarchie.

Donc, le partitionnement consiste à rassembler les processus, qui vont s'exécuter sur le même processeur, dans des BLOCKs en ajoutant la directive NODE à ces derniers. Cette procédure nécessite une restructuration du réseau de communication. Cette restructuration est facile à réaliser, la Figure 60 montre un exemple d'une spécification SDL qui illustre cette tâche.

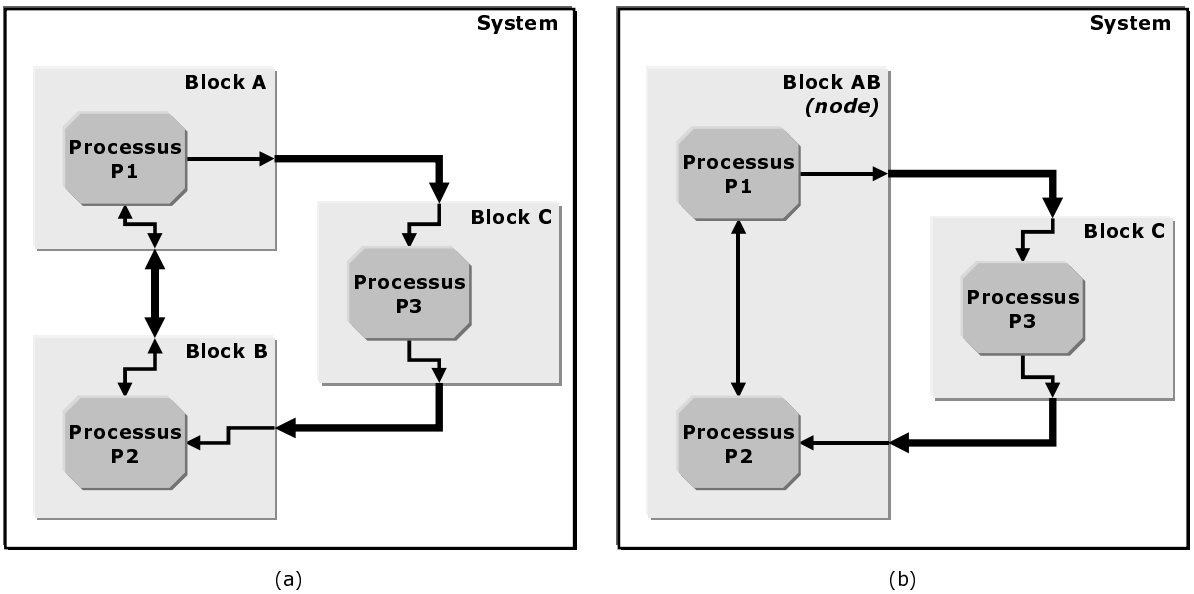


Figure 60 – Le partitionnement du système et son influence sur le réseau de communication

Dans cette figure, les flèches entre les blocs sont des canaux de communication et les autres sont des routes (voir Figure 60a). Nous remarquons dans cette figure la simplification de la communication entre les deux blocs **A** et **B** après leur fusion. Elle devient une

communication interne à travers d'une route (voir Figure 60b). La fusion des deux blocs **A** et **B** est très facile à faire en pratique. A part cela, il y a très peu de modifications à faire. Il s'agit de changer légèrement la définition des canaux de communications connectés du nouveau bloc **AB** au bloc **C**.

6.3.4.2 L'attribution des processeurs logiciels/matériels

Il s'agit de choisir, pour chaque nœud, le type du processeur sur lequel les processus du nœud seront exécutés. Le processeur peut être matériel (ASIC), ou un certain microprocesseur.

Comme pour chaque bloc de base nous avons calculé plusieurs délais correspondant aux plusieurs processeurs, dont nous disposons, cette étape d'attribution des processeurs est réalisée simplement par l'annotation du bloc de base par le délai correspondant à son implémentation sur le processeur choisi. Bien évidement, tous les blocs de base qui se trouve à l'intérieur d'un nœud doivent être implémentés sur le même processeur.

En effet, cette étape est complémentaire à celle présentée dans la section 6.3.3.1.

6.3.4.3 Le choix de la communication

Dans cette tâche nous décidons, pour chaque canal de communication, le protocole de communication qui sera utilisé. Ce choix est un paramètre qui influence grandement la performance d'une architecture.

Nous avons vu dans la section 6.3.3.2, comment l'annotation de la communication est faite. Le choix d'un certain protocole de communication revient à ajouter des attributs aux routes connectées aux canaux de communication et de sélectionner les bonnes procédures, de la bibliothèque d'estimation de la communication, à appeler lors des actions *input* et *output*.

Il faut noter que le choix de ces dernières procédures dépend aussi du choix des deux processeurs sur les quels les deux processus communicants vont être exécutés. Car, dans la bibliothèque d'estimation de la communication, à chaque type de protocole correspond plusieurs procédures (délais), selon le type du processeur implémentant ce protocole. La Figure 61 illustre un modèle SDL instrumenté avec des attributs de communication.

```
package RDV_PROCEDURES;                                  Bibliothèque d'estimation de la communication
...
procedure pin_sig_sst10_e;                               Délai pour recevoir un int avec le protocole
 start;                                                  rendez-vous implémenté sur ST10
  task 'delay' comment '#delay ((28+12)*p_sst10)';
  task 'delay' comment '#delay ((28+input_char_st10+14+output_char_st10)*p_sst10)';
 return;
endprocedure pin_sig_sst10_e;
...
endpackage RDV_PROCEDURES;

use RDV_PROCEDURES;
SYSTEM motor /*COMMENT '#node'*/;
...
procedure PC_iPIDPC_sig;                                 Le choix du protocole de
    start;CALL pin_sig_sst10_e;return;                   communication
  endprocedure PC_iPIDPC_sig;
...
  BLOCK PC;
    PROCESS host(1,1);
      START ;
      STATE DistCtrl;
        INPUT pid_pc_values via pc_pid.1;                Annotation de la communication
        CALL PC_iPIDPC_sig;
          TASK 'TEE - START' COMMENT '#delay (PC_delay2+0)';
          TASK bb:=65281;
          TASK ind := 0;
          TASK bb:=65297;
        loop1:
...
        JOIN loop1;
      ENDSTATE;
    ENDPROCESS;
  ENDBLOCK;
ENDSYSTEM;
```

Figure 61 – Le choix et l'annotation de la communication

6.3.5 Simulation

Il s'agit d'exécuter la description SDL obtenue à la sortie de l'étape précédente, et qui contient toutes les informations de performance et de l'architecture modélisée. Ceci est réalisé grâce au simulateur ***geodesim*** d'ObjectGEODE. Le résultat de la simulation indique la performance de l'architecture.

En effet, il faut d'abord compiler la spécification SDL avec l'outil ***gsmcompil*** d'ObjectGEODE. Le résultat de la compilation est exécuté par le simulateur geodesim. Dans un fichier d'initialisation (*system.startup*), nous pouvons tracer les variables et les E/S, nous pouvons également préciser une condition d'arrêt de la simulation. Il faut choisir la même condition d'arrêt pour toutes les architectures qu'on va simuler pour les comparer (on peut aussi mettre plusieurs points d'arrêt à l'intérieure du même système pour l'analyse détaillée des différentes phases de l'exécution de l'application). A la fin de chaque simulation (ou à l'arrivée à un point d'arrêt), ***geodesim*** affiche le temps de l'exécution (cumuls des délais partiels),qui représente la performance de l'architecture simulée.

Dans la section 6.2.3.3.2, nous avons mentionné que l'ordonnancement des processus à l'intérieure d'un nœud (un processeur), suit une distribution aléatoire uniforme. Et que la directive PRIORITY peut être associée à un processus ou à un type de processus, à l'intérieure d'un nœud pour changer ce choix aléatoire. Si le comportement de cette directive ne peut pas répondre à une certaine stratégie d'ordonnancement, nous pouvons indiquer au simulateur ***geodesim*** un *scénario* particulier à suivre lors de l'exécution. Ce *scénario,* qui est dépendant de l'application, contiendra les informations nécessaires pour modéliser notre stratégie d'ordonnancement.

La simulation avec ***geodesim*** (niveau système), prend un temps très court, de l'ordre de la seconde. C'est l'avantage principal de notre méthodologie. Nous pouvons donc parcourir la boucle d'exploration d'architectures (voir Figure 54) plusieurs fois dans un temps raisonnable.

6.4 Analyse expérimentale de la méthode et résultats

Pour valider notre méthodologie, nous l'avons appliqué sur un exemple réel : système d'asservissement de la vitesse de deux moteurs. Nous avons estimé la performance de plusieurs architectures du système. Ces mêmes architectures ont été synthétisées et cosimulées dans l'environnement de MUSIC. La comparaison et l'analyse des résultats nous ont permis de démontrer la puissance de notre approche. Aussi, nous avons relevé la capacité et les limitations de cette approche.

Dans la section 6.4.1 nous présentons l'application choisie. Les architectures choisies et l'application de notre méthode sont présentées dans la section 6.4.2. Ensuite, la synthèse et les résultats de la cosimulation de ces architectures sont faites dans la section 6.4.3. Finalement, les résultats sont analysés dans la section 6.4.4.

6.4.1 L'application : contrôleur d'un bras de robot

Il s'agit de concevoir un système d'asservissement d'un bras de robot. Dans la version complète de cet exemple, le bras du robot est constitué de 8 moteurs. Nous allons en considérer seulement deux pour améliorer la clarté de l'exemple. Ce système ajuste la variation de vitesse de chaque moteur de façon à ce que le mouvement du bras soit souple et que les contraintes physiques d'accélération et de freinage soient respectées.

Le système est modélisé en SDL (voir Figure 62). Il s'agit de quatre processus *Host*, *Pid*, *Moteur1* et *Moteur2*. Le processus *Host* (à l'intérieur du bloc PC) envoie les consignes de vitesse et les paramètres de contrôle au *Pid*, qui à son tour, prend la commande des deux moteurs. Le processus *Pid* (Proportionnel, Intégral, Dérivation), contrôle la vitesse des deux moteurs selon certaines équations. A chaque cycle, le processus *Pid* calcule et envoie une nouvelle valeur à un des deux moteurs, après la lecture de sa vitesse instantanée. Le contrôle des deux moteurs se fait en série. Le moteur est modélisé par son équation au dérivé.

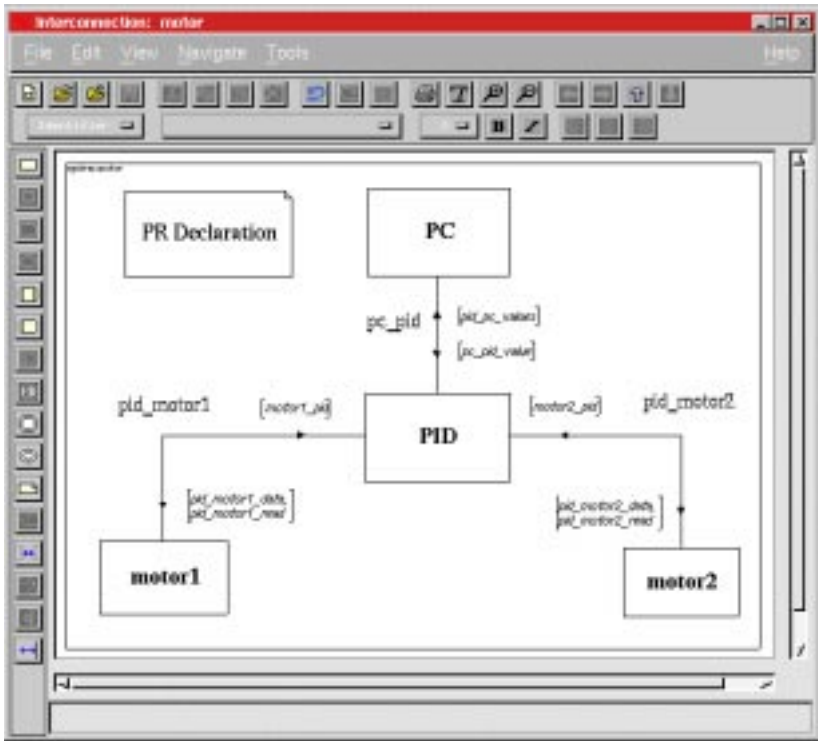


Figure 62 – La modélisation du système de contrôle en SDL

Nous avons décidé de concevoir le système entier en modules logiciels et matériels. Nous disposons de deux chaînes de développement pour deux microcontrôleurs : ST10 de **STMicroelectronics** et 8051, avec un seul compilateur pour chacun ; en plus de l'option matérielle (ASIC). Si on ne considère aucune contrainte d'implémentation, et si on ne considère pas le choix de la communication, le nombre d'architectures possibles est donné par l'équation (6-2). Dans ce cas, **p** est égal à 3 et **n** est égal à 4, donc le nombre d'architectures possibles V_4^3 est 309. Nous remarquons que c'est un nombre énorme pour un exemple aussi simple.

6.4.2 L'application de la méthode d'estimation/exploration

Nous allons appliquer les différentes étapes de notre méthode, comme elles ont été présentées à la section 6.3.

6.4.2.1 L'estimation des délais élémentaires

Les délais élémentaires sont de deux types : pour les blocs de base et pour la communication.

Nous analysons la description SDL du système et nous identifions tous les blocs de base en ajoutant des marques de début/fin de bloc. En utilisant MUSIC, nous procédons à une première implémentation complètement matérielle, sans la synthèse de la communication. Nous obtenons quatre fichiers en VHDL RTL (correspondant aux 4 processus du système). Nous retrouvons facilement les marques de blocs de base. Le temps d'exécution (en nombre de cycles), d'un bloc de base est égale au nombre de transitions entre les deux marques qui lui bornent. On calcule ce temps pour tous les blocs de base. Ensuite, nous procédons à une deuxième implémentation complètement logicielle, sans la synthèse de la communication. Nous obtenons quatre fichiers en C (correspondant aux 4 processus du système). Nous compilons (sans optimisation), ces fichiers avec le compilateur de ST10 et le compilateur de 8051. Nous retrouvons, relativement facilement, les marques des blocs de base dans les fichiers assembleurs résultants. En utilisant les spécifications (*data sheets*) des deux processeurs, nous calculons le temps d'exécution (en nombre de cycles d'horloge), de chaque bloc de base sur chaque processeur. Nous avons ainsi obtenu trois délais différents pour chaque bloc de base en SDL.

Nous avons du créer la bibliothèque d'estimation de la communication. Pour cela, nous avons décrit un système très simple en SDL qui ne contient que quelques processus qui communiquent. Pour simplifier l'exemple, nous avons considéré un seul protocole de communication de la bibliothèque de MUSIC : le rendez-vous. Donc, comme dans le cas des blocs de base, nous procédons aux deux implémentations logicielle/matérielle par MUSIC, mais en synthétisant la communication et en choisissant le protocole rendez-vous. Les fichiers C et VHDL obtenues sont traités de la même manière. Le début et la fin de la communication sont reconnues facilement due à la simplicité du système décrit en SDL. Cependant, nous avons pris beaucoup de soin lors du calcul du temps d'exécution de la communication, car comme nous l'avons expliqué dans la section 6.3.2.2, à chaque action *input* et *output* doit correspondre une procédure contenant des sous délais indivisibles (blocs de base), selon

l'implémentation. Donc pour chaque action *input* et chaque action *output* nous avons construit trois procédures (trois délais différents). Ces procédures dépendent aussi de la taille de données transmises. Ils sont regroupés pour construire une sous bibliothèque d'estimation de la communication.

6.4.2.2 L'annotation

Nous déclarons, au début de la spécification SDL, tous les délais des blocs de base que nous avons calculé dans l'étape précédente. Aussi, nous déclarons la bibliothèque d'estimation de la communication. Nous annotons aussi les blocs de base et les actions *input* et *output* par des directives DELAY et des appels de procédures (de la bibliothèque), comme expliqué dans la section 6.3.3. Ces dernières annotations seront configurées avec les bonnes valeurs lors du choix d'architecture (étape suivante).

6.4.2.3 Le choix de l'architecture

Concernant la tâche du choix de la communication, nous avons décidé de choisir un seul type de communication : le protocole rendez-vous. Donc l'attribution des bonnes valeurs aux appels de procédures, ajoutés (à la spécification SDL) dans l'étape précédente, dépend seulement des deux autres tâches de cette étape.

Le partitionnement du système et l'attribution des processeurs logiciels/matériels caractérisent l'architecture que l'on veut réaliser. Nous avons vu qu'il y a un nombre très grand d'architectures possibles. Nous avons décidé d'en choisir quelques-unes parmi les plus pertinentes. Ainsi, nous avons choisi de faire les architectures présentés dans le Tableau 11.

Architecture	Host	PID	Moteur1	Moteur2
A1	8051	Matériel	ST10	
A2	ST10	Matériel	ST10	
A3	ST10	Matériel	ST10	ST10
A4	Matériel	Matériel	ST10	ST10
A5	Matériel	Matériel	Matériel	Matériel

Tableau 11 – Les architectures choisies

Nous avons choisi les fréquences d'horloges suivantes : 2MHz pour ST10, 12MHz pour 8051 et 1MHz pour la partie matérielle (ASIC).

6.4.2.4 La simulation

Pour chaque architecture, nous compilons (avec ***gsmcompil***), la spécification SDL résultante de l'étape précédente. Un fichier d'initialisation de la simulation (.*startup*), est créé. Pour les résultats présentés dans cette section, la simulation a été arrêtée quand la vitesse du moteur 1 arrive à une valeur stable (environ 10 unités), ainsi le temps de simulation correspondent à l'intervalle représenté sur la Figure 63.

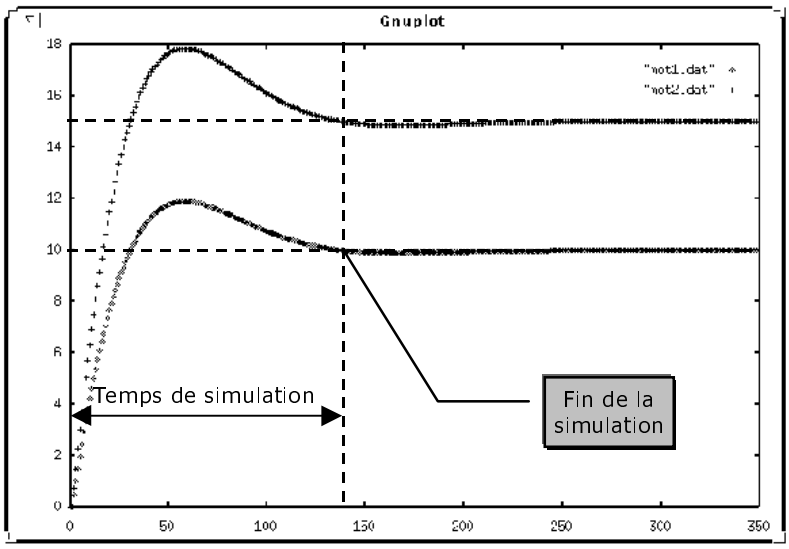


Figure 63 – Les résultats de la cosimulation et le temps de simulation

Nous avons procédé à la simulation de toutes les architectures, avec ***geodesim***. Le Tableau 12 montre les résultats obtenus. La performance est mesurer en µs.

Architecture	Estimation SDL (µs)
A1	1866
A2	1873
A3	1871
A4	1779
A5	21

Tableau 12 – La performance des architectures pour la simulation avec ***geodesim***

6.4.3 La synthèse et cosimulation avec MUSIC

Pour valider les résultats obtenus par notre méthode d'estimation, il a fallu les comparer avec les résultats réels. Grâce à l'outil de cosimulation ***MCI*** [HMVRJ98], nous pouvons obtenir des résultats de performance précis au niveau cycle. Pour les deux

microcontrôleurs ST10 et 8051, nous disposons du simulateur d'architecture ***ISS*** que nous avons adapté à ***MCI***. Pour l'ASIC (VHDL), nous utilisons le simulateur ***VSS*** de **Synopsys**, qui lui aussi était adapté à ***MCI***. A partir de la spécification SDL originale du système, nous procédons à la synthèse des architectures choisies (voir Tableau 11), en utilisant MUSIC. Les résultats de la cosimulation sont présentés dans le Tableau 13.

Architecture	Cosimulation RTL (µs)
A1	2977
A2	2986
A3	1841
A4	1747
A5	22

Tableau 13 – La performance des architectures pour la cosimulation avec ***MCI***

6.4.4 L'analyse des résultats

Dans cette section, nous allons analyser les résultats que nous avons obtenus. Ces résultats nous permettent d'effectuer des analyses non seulement qualitatives, mais aussi quantitatives. Nous démontrons la validité de notre méthodologie. Ensuite, nous présentons ses avantages et sa grande capacité. Les difficultés, ainsi que quelques limitations seront développées en dernier.

6.4.4.1 Validité de la méthode

Nous dressons un tableau récapitulatif des résultats de performance obtenus (voir Tableau 14). Nous remarquons pour les trois dernières architectures, la précision est excellente, l'erreur d'estimation est, en moyenne, de l'ordre de 3%.

Architecture	Host	PID	Moteur1	Moteur2	Estimation SDL (µs)	Cosimulation RTL (µs)	Erreur d'estimation
A1	8051	Matériel	ST10		1866	2977	-37%
A2	ST10	Matériel	ST10		1873	2986	-37%
A3	ST10	Matériel	ST10	ST10	1871	1841	2%
A4	Matériel	Matériel	ST10	ST10	1779	1747	2%
A5	Matériel	Matériel	Matériel	Matériel	21	22	-5%

Tableau 14 – Les performances des architectures et la comparaison des résultats

Pour les deux premières architectures, nous avons une erreur d'estimation assez grande (38%). Cette erreur nous a incité à regarder les traces de l'exécution en détail. Nous remarquons que cette erreur arrive lorsque *Moteur1*et *Moteur2* partagent le même processeur. L'ordonnancement de ces deux processus suit une distribution aléatoire uniforme dans le modèle d'exécution de ***geodesim***. Cependant, l'ordonnancement dans le code généré par MUSIC se matérialise par l'entrelacement des deux machines d'états des deux processus. En particulier, les protocoles de communications sont entrelacés. C'est à dire, lorsque *Pid* est entrain de communiquer avec *Moteur1*, chaque transition dans le protocole de communication du coté *Moteur1*, sera suivi par une transition du *Moteur2*. Il se trouve dans notre cas, où *Moteur1* est identique à *Moteur2*, que la transition exécutée dans le *Moteur2* est la première partie d'une communication. Il s'agit de la première transition d'une demande de communication (*handshake*). Cette transition consomme un temps non négligeable du temps de la communication (presque 20%). En plus, comme elle sera exécutée un nombre de fois égale au nombre de transitions dans le protocole de communication, elle peut augmenter le temps d'une communication de presque 30 à 50 %. Il faut se rappeler que cette analyse est faite dans le cadre d'un protocole rendez-vous.

L'utilisation d'un *scénario* prédéfini pour la simulation avec ***geodesim***, permet de modéliser l'entrelacement utilisé par MUSIC. Cependant, il ne permet pas de modéliser ce schéma précis d'entrelacement d'une communication avec une autre qui n'est pas établie. Pour résoudre ce problème, une première solution est de modifier la spécification SDL en ajoutant quelques actions pour modéliser ce *scénario*. Dans la pratique, cette solution est très difficile à mettre en œuvre d'une façon automatisé. En plus, ce n'est pas acceptable d'introduire des modifications lourdes dans la spécification SDL originale, on risque de changer sa fonctionnalité. La deuxième solution est analytique. Il s'agit de trouver des équations analytiques fiables qui corrigent cette erreur. Cette solution sera plus facile à mettre en pratique. Pour cet exemple, nous avons changé les délais des communications pour prendre en compte le résidu expliqué ci-dessus. Cette solution est spécifique pour cette application. Nous avons obtenu les résultats montrés dans le Tableau 15.

Architecture	Host	PID	Moteur1	Moteur2	Estimation SDL (µs)	Cosimulation RTL (µs)	Erreur d'estimation
A1	8051	Matériel	ST10		3149	2977	6%
A2	ST10	Matériel	ST10		3156	2986	6%

Tableau 15 – Les performances après la correction des résultats de la simulation SDL

Nous remarquons que nous obtenons un excellent résultat. Cependant, si la solution apportée était faite plus soigneusement, nous arrivons à une erreur d'estimation encore plus petite. Maintenant nous pouvons assurer la validité de notre méthodologie. Cette nouvelle méthode d'estimation/exploration donne des résultats parfaitement fiables et très précis.

6.4.4.2 Capacité de la méthode

Notre méthode d'estimation/exploration réalise de très bons taux de précision et de rapidité. Nous avons vu que l'erreur d'estimation est de l'ordre de 4%. L'analyse des résultats nous permet de conclure que cette méthode est parfaitement fiable. En plus, le temps d'exploration d'architectures par cette méthode est incomparable avec celui nécessaire à l'implémentation et la cosimulation du même nombre d'architectures (surtout si ce dernier est grand). En fait, la cosimulation au niveau cycle, par MCI, prend actuellement un temps énorme. Le Tableau 16, montre la différence de temps entre la cosimulation et la simulation SDL.

Paramètres	Simulation SDL	Cosimulation RTL
Pas de simulation	Transition	Cycle d'horloge
Nombre de pas simulés	3300	2 451 000
Nombre de pas / sec	330	23
Temps de simulation	10 s	30h 22min 39s

Tableau 16 – La comparaison entre les temps de simulation

Nous pouvons facilement noter dans cette figure, la capacité de notre méthode. Nous avons vu aussi que le nombre d'architectures à explorer augmente exponentiellement avec la complexité de l'application et la diversité des éléments d'architectures dont nous disposons. Ce qui rend l'utilisation de notre méthodologie d'estimation/exploration encore plus avantageuse. La durée totale du flot d'estimation/exploration de notre méthode est très adaptée au besoin de rapidité demandé par la conception conjointe matérielle/logicielle.

Aussi il faut noter que notre modèle d'estimation de la communication est très général, et peut être appliqué sur tout type de communication. En plus, l'idée de construire une bibliothèque d'estimation de la communication est très avantageuse. Cette bibliothèque peut être réutilisée pour toutes les autres applications, ce qui facilite beaucoup l'application de notre méthode. En fin, la plus part des étapes du flot d'estimation/exploration peuvent être automatisées facilement.

6.4.4.3 Les limitations

Comme toute nouvelle approche, il y a toujours quelques difficultés et quelques limitations. Nous allons en citer celles que nous avons pu soulever :

1. Il faut disposer pour chaque microprocesseur une chaîne de développement contenant particulièrement un compilateur C qui génère du code assembleur. Ce type de compilateur n'est pas toujours disponible, surtout pour des microprocesseurs spécifiques.
2. Pour des microprocesseurs complexes (pipeline, parallélisme interne, etc), il est difficile de calculer, avec précision, le temps d'exécution statiquement, i.e. à partir du code assembleur.
3. L'identification des blocs de base dans le code assembleur, très facilité avec l'ajout des marques de début/fin, reste parfois difficile due à l'introduction, par le compilateur, d'appels de procédures internes de sa bibliothèque.
4. Pour pouvoir identifier les blocs de base dans le code assembleur, nous sommes obligés de ne pas utiliser des options d'optimisation qui risquent de brouiller le code. En particulier, les marques de début/fin de blocs de base risquent de disparaître. Cette contrainte ne nous permet pas d'estimer le code optimal.
5. Le modèle architectural des parties matérielles est très simple. Nous disposons actuellement d'un seul modèle d'exécution pour le matériel, chaque transition dans la machine d'états finis s'exécute en un seul cycle.
6. La difficulté de modéliser en SDL quelques schémas spéciaux d'ordonnancement de processus. Cette difficulté a été illustrée par un exemple dans la section 6.4.4.1.
7. Il faut ajouter aussi les limitations du simulateur d'ObjectGEODE (peu de modèles de communication et quelques *bugs*).

Toutes ces difficultés sont superficielles, elles ne constituent pas une défaillance dans notre méthodologie.

6.5 Conclusion

En ce chapitre, une nouvelle méthodologie d'estimation de performance au niveau d'abstraction du système est présenté. Cette méthodologie permet l'exploration efficace de

l'espace des solutions avant même de faire la conception conjointe logicielle/matérielle. Nous avons développé et validé cette méthodologie dans l'environnement de codesign MUSIC pour des spécifications en SDL. Cette méthodologie est un atout important pour l'exploration d'architectures dans un environnement de codesign de systèmes complexes. La combinaison d'un tel outil avec l'outil de codesign et l'outil de cosimulation constituera un environnement parfait pour la conception conjointe matérielle/logicielle de systèmes complexes.

Le fossé d'abstraction entre la description au niveau système et au niveau RT est considérable, par conséquent l'estimation de performance des architectures au niveau système est un problème très complexe. Il est particulièrement difficile d'estimer les temps d'exécution du logiciel sur les processeurs complexes. De toute façon, pendant l'exploration de l'espace de conception au niveau système, la précision d'estimation n'est pas si importante que la fidélité. En d'autres termes, c'est la valeur relative des estimations comparées à d'autres estimations qui seront employées pour décider si une architecture est meilleure que d'autres, et non pas la valeur absolue des estimations.

L'avantage d'employer l'exploration de l'espace des solutions au niveau système a été mis en évidence par le fait que seulement quelques minutes sont nécessaires pour tester de nouvelles architectures avec notre méthodologie, tandis qu'il fallait plusieurs jours pour tester les mêmes architectures au niveau RT.

Dans ce chapitre, nous avons présenté une nouvelle méthode d'estimation du temps d'exécution de systèmes complexes. Cependant, cette méthodologie peut être appliquée pour l'estimation d'autres paramètres comme la surface et la consommation d'énergie.

Chapitre 7

Conclusion

Enfin, pour conclusion de cette morale, je m'avisai de faire une revue sur les diverses occupations qu'ont les hommes en cette vie [...]

Descartes (René), Discours de la méthode.

Ce chapitre présente le bilan de ce travail et quelques perspectives pour les développements futurs.

Le travail effectué durant cette thèse a permis une meilleure compréhension du processus de synthèse au niveau comportemental, et de définir une nouvelle méthodologie pour l'estimation de performance au niveau système. La méthodologie de synthèse architecturale flexible permet au concepteur d'adapter le processus de conception aux besoins de chaque application. Un autre aspect important est l'utilisation des estimateurs de haute fidélité qui permettent la prévision des résultats de la synthèse. Ainsi, l'exploration de l'espace des solutions peut être faite à un niveau d'abstraction plus élevé. En faisant cela, nous pouvons accélérer l'exécution complète du processus de conception puisque les interactions les plus lentes, à des niveaux d'abstraction plus bas, peuvent être évitées.

La méthodologie de synthèse architecturale flexible a été appliquée dans MUSIC à plusieurs niveaux d'abstraction. Cette méthodologie remet le travail de création dans les mains du concepteur et fournit les outils pour effectuer le travail monotone et fastidieux. Elle donne également aux concepteurs plus de pouvoir pour contrôler et prévoir les résultats du processus de synthèse.

Le deuxième chapitre a présenté les méthodologies de conception utilisées aux niveaux d'abstraction plus élevés : au niveau système et au niveau comportemental. Il contient une introduction aux différentes techniques de conception qui mettent en œuvre les méthodologies présentées. Une plénitude de techniques peut être employée pour mettre en application des méthodologies de conception au niveau comportemental et au niveau système. Nous avons présenté celles mises en application dans MUSIC. Au niveau système nous avons montré l'importance de l'outil d'évaluation architectural pour guider le partitionnement logiciel/matériel et la synthèse de la communication. En outre, nous avons montré comment les problèmes de synthèse comportementale (l'ordonnancement, l'allocation et l'affectation) sont interdépendants et qu'il y n'a aucun ordre idéal pour les résoudre. Ainsi, nous avons montré la nécessité de différents flots de conception pour différents styles d'applications.

Le troisième chapitre a présenté le chevauchement de fonctionnalité qui existe entre les outils comportementaux et les outils de synthèse RTL en ce qui concerne les problèmes d'allocation et d'affectation de ressources. Nous avons montré une manière d'explorer ce chevauchement en produisant un flot flexible de synthèse comportementale et nous avons montré comment il est mis en application dans MUSIC. L'interprétation et la synthèse des résultats intermédiaires permettent de faciliter le lien entre la synthèse comportementale et la synthèse RTL. Un flot flexible de synthèse peut être employé pour adapter le processus de synthèse à différents styles d'application, évitant ainsi les étapes inutiles ou non-optimales.

Nos résultats ont prouvé clairement que le choix du meilleur flot de synthèse dépend du style de l'application et des priorités d'optimisation pour chaque paramètre de qualité (la surface, le délai, la puissance, etc.).

Nous n'avons pas développé tous les flots de synthèse possibles dans MUSIC. Il reste à étudier la génération d'un modèle VHDL synthétisable après l'allocation et l'affectation des unités fonctionnelles et l'application des algorithmes sur une partie de la description choisie. Pour ce dernier cas, il pourrait être intéressant de faire l'allocation et l'affectation des unités fonctionnelles pour des types d'opérations choisis (par exemple, la division) et de laisser les autres opérations à la synthèse RTL. En outre, appliquer le ré-ordonnancement seulement à un ensemble de transitions choisies (par exemple, celles de grands graphes de flot de données) et laisser les autres transitions intactes. Le mécanisme de corrélation peut être étendu pour montrer la relation entre les descriptions comportementale/intermédiaire et les éléments du contrôleur/chemin de données de l'architecture synthétisée.

Pour démontrer l'effet que les nouvelles technologies peuvent avoir sur les techniques de conception même au niveau comportemental, le quatrième chapitre compare les différents algorithmes d'allocation et d'affectation des interconnexions en termes de la surface du circuit synthétisée. Cette comparaison a été rendue possible par le schéma flexible d'interconnexion utilisé dans l'architecture cible de MUSIC. Les schémas d'interconnexion basés sur des bus et des multiplexeurs peuvent être employés. Nous avons aussi montré comment les directives d'optimisation pour réduire le nombre de bus peuvent avoir comme conséquence une pénalité de surface d'environ 20% par rapport à notre solution d'optimisation du nombre de cellules.

Il peut être intéressant de généraliser les directives de minimisation de cellules aux autres problèmes d'allocation et d'affectation. Une étape d'optimisation globale des cellules peut également être ajoutée au flot de synthèse. Par exemple, dans certains cas il peut être intéressant de dupliquer les unités fonctionnelles pour réduire le nombre des multiplexeurs.

Le cinquième chapitre décrit les outils flexibles de synthèse mis en œuvre autour du modèle SOLAR. Les outils développées utilisent des techniques de conception bien connues, elles ont été adaptées pour pouvoir fonctionner dans un flot flexible de conception. La plupart des techniques de synthèse comportementales sont spécifiques à un domaine d'application donné (flot de contrôle ou flot de données). Avec notre approche flexible nous pouvons combiner différentes techniques de conception dans un même outil, le rendant capable de traiter des applications mixtes. L'évolution de la technologie impose une révision de certaines

techniques de conception. De nouvelles heuristiques ont été développées pour résoudre le problème de l'allocation et de l'affectation des interconnexions en utilisant des directives d'optimisation des cellules.

La méthodologie d'estimation de haute fidélité au niveau comportemental est toujours un problème en grande partie non résolu. Des outils d'estimation qui peuvent fonctionner à n'importe quelle étape de notre flot de synthèse sont nécessaires. De tels estimateurs doivent fonctionner avec les résultats intermédiaires de notre flot flexible, ce qui rend le problème encore plus complexe.

Une nouvelle méthodologie d'estimation de performance au niveau système a été présentée dans le sixième chapitre. Cette méthodologie a été appliquée au langage SDL pour le simulateur *Geodesim* et l'outil de conception conjointe MUSIC. La méthodologie peut être appliquée à d'autres modèles d'exécution au niveau système et d'autres outils de conception conjointe, tant que les restrictions présentées sont respectées. Des modèles de performance ont été développés pour des délais d'exécution de logiciel et de matériel, mais la méthodologie peut être étendue à d'autres métriques de qualité. Il est vrai que la qualité de l'évaluation de performance obtenue de façon dynamique par la simulation dépendra fortement du *testbench* utilisé. Notre méthodologie permet d'employer le même *testbench* défini pour la validation fonctionnelle. Alors, les différentes solutions architecturales peuvent être évaluées dans les conditions de fonctionnement les plus importantes. L'utilisation des évaluations de haute fidélité est fondamentale, elle permet de prévoir le résultat futur du processus de synthèse à un niveau d'abstraction élevé.

L'application de la méthodologie à SDL et MUSIC doit être automatisée afin d'essayer le concept dans un grand nombre d'exemples de conception. MUSIC peut être modifiée pour permettre l'exécution d'un mécanisme de corrélation plus propre. Un mécanisme de corrélation bien défini peut être également utile pour suivre l'évolution du processus de synthèse et pour faciliter la débogage. Le modèle de performance peut être étendu à d'autres paramètres de qualité tels que la surface et la puissance. La méthodologie peut être appliquée à d'autres modèles exécutables et d'autres outils de conception conjointe.

La conclusion de cette étude est qu'il est aussi important de penser à l'intégration des outils dans l'environnement de synthèse que la découverte des nouvelles heuristiques optimisées à un domaine d'application très spécifique. Dans ces dernières années la synthèse RTL a évoluée de façon à devenir le « langage assembleur » de la conception de circuits

intégrés. A l'avenir les outils de synthèse comportementale et de conception conjointe logicielle/matérielle remplaceront les outils de synthèse RTL dans la partie frontale des environnements de conception utilisés par l'industrie. Il faut donc profiter de l'expertise et des outils très puissants existants dans le domaine de la synthèse RTL, dans les environnements de conception à plus haut niveau d'abstraction.

Annexe A

La Bibliothèque des Unités Fonctionnelles

Cet annexe présente la syntaxe de la bibliothèque d'interfaces des unités fonctionnelles utilisée par l'outil d'allocation du système MUSIC. Cette bibliothèque a la forme suivante :

```
FunctionalUnitLibrary : (SOLAR <name> FunctionalUnit)

FunctionalUnit : (FUNCTIONUNIT <name> FUProperty View)

FUProperty :  MTProperty MAProperty

MTProperty :  (PROPERTY max_time <value>)

MAProperty :  (PROPERTY area <value>)

View :  (VIEW protocol ViewType Interface)

ViewType :  (VIEWTYPE "fu")

Interface : (INTERFACE Parameter Port Method)

Parameter : (PARAMETER <name> Direction (BIT)) ↻

Direction : (DIRECTION <dir>)

Port : (PORT (ARRAY <name> <value>) Direction (BIT) PProperty) ↻

PProperty : (PROPERTY PortType <ptype>)

Method :  (METHOD <name> MTProperty StateTable) ↻

StateTable :  (STATETABLE p1 State)

State :  (STATE <name> MTProperty Assign) ↻

Assign :  (ASSIGN <name> <name>) ↻

<name> : 0-9 a-z A-Z _ ↻
<value> : 0-9.0-9 ↻
<dir> : in out
<ptype> : control data
```

Bibliographie

[Anc86] F. Anceau, "**The Architecture of Microprocessors"**, Addison-Wesley Publishing Company, 1986.

[Bacq97] P. Bacquet, **"Translating SDL into OPNET: From formal validation to performance evaluation"**, dans Proc. of OPNETWORK Conference, Mil3, mai 1997.

[Berg95] R.A. Bergamaschi, **"Productivity Issues in High-Level Design: Are Tools Solving the Real Problems?"**, Proc. 32rd Design Automation Conference, 1995.

[Bray87] R. Brayton et al., **"MIS: a Multiple Level Logic Optimization System"**, IEEE Trans. CAD, vol. CAD-6, pp. 1062-1081, novembre 1987.

[Cam91] R. Camposano, **"Path-based Scheduling for Synthesis"**, IEEE Trans. On CAD, 10(1): 85-93, janvier, 1991.

[Cam96] R. Camposano, **"Behavioral Synthesis"**, Proc. 33rd Design Automation Conference, juin 1996.

[Camp87] R. Camposano, **"Structurel Synthesis in the Yorktown Silicon Compiler"**, VLSI, 1987.

[CGR93] V. Chaiyakul, D.D. Gajski et L. Ramachandran, **"High-Level Transformations for Minimizing Syntactic Variances"**, Proc. 30th Design Automation Conference, Los Alamitos, California, 1993.

[DaKa96] A. Dasgupta et R. Karri, **"Electromigration Reliability Enhancement via Bus Activity Distribution"**, Proc. 33rd Design Automation Conference, Las Vegas, Nevada, juin 1996.

[DaVe97] C. Dangelo et P. Verhofstadt, **"Report of the Ad Hoc Working Group on Design & Test"**, Published on the Internet: http://marco.fcrp.org/reports/designtest.htm, avril 1997.

[Dev87] S. Devadas et al., **"MUSTANG: State Assignment for Finite State Machines for Multi-Level Logic Implementations"**, Proc. ICCAD, pp. 16-19, 1987.

[EHGR96] H.-J. Eikerling, W. Hardt, J. Gerlach, et W. Rosenstiel, **"A Methodology for Rapid Analysis and Optimization of Embedded Systems**", In Engineering of Computer Based Systems, Friedrichshafen, Germany, mars 1996.

[Ewer90] C. Ewering, **" Automatic High Level Synthesis of Partitioned Busses "**, Proceedings of the IEEE International Conference on Computer-Aided Design, pp. 304-307, novembre 1990.

[FaeO94] O. Faergemand, A. Olsen, **"Introduction to SDL-92"**, Computer Networks and ISDN Systems, vol. 26, pp. 1143-1167, 1994.

[Fis81] J.A. Fisher, **"Trace Scheduling: A Technique for Global Microcode Compaction"**, IEEE Trans. on Computers, C-30(7): 478-490, juillet 1981.

[FreTa87] M.L. Fredman, et R.E. Tarjan, **"Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms"**, Journal of the ACM, Vol. 34, 596-615, 1987.

[Gaj92] D. Gajski, N. Dutt, A. Wu, et Y. Lin, **"High-Level Synthesis: Introduction to Chip and System Design"**, Kluwer Academic Publishers, Boston, Massachusetts, 1992.

[GaVa94] D.D. Gajski, F. Vahid, S. Narayan, et J. Gong, **"Specification and Design of Embedded Systems"**, Prentice Hall, Englewood Cliffs, 1994.

[GICH96] D. Gajski, T. Ishii, V. Chaiyakul, H. Juan et T. Hadley**, "Interactive Behavioral synthesis"**, SASIMI, 1996.

[GiKn84] E.F. Girczyc & J.P. Knight, **"An ADA to standard cell hardware compiler based on graph grammars and scheduling"**, Proc. of ICCD, 1984.

[GPRab98] L. Guerra, M. Potkonjak et J. Rabaey, **"A Methodology for Guided Behavioral-Level Optimization"**, Design Automation Conference, p.309-314, 1998.

[GuMi90] R. K. Gupta et G. De Micheli, **"Partitioning of Functional Models of Synchronous Digital Systems"**, IEEE Int. Conference on Computer Aided Design (ICCAD), novembre 1990.

[GVNG98] D. D. Gajski, F. Vahid, S. Narayan, J. Gong, **"System-Level Exploration with SpecSyn,"** In Proceedings of Design Automation Conference, pp. 812, 1998.

[HeEr95] J. Henkel et R. Ernst, **"A Path-Based Estimation Technique for Estimating Hardware Runtime in HW/SW-Cosynthesis"**, In Proceedings 8th IEEE International Symposium on System Level Synthesis, p. 116-121, Cannes, France, 1995.

[HMVRJ98] F. Hessel, Ph. Le Marrec, C. A. Valderrama, M. Romdhani, A. A. Jerraya, **"MCI – Multilanguage Distributed Co-simulation Tool**", DIPES 98, Paderborn, Germany, octobre 1998.

[Hsu90] C.Y. Huang, Y.S. Chen, Y.L. Lin et Y.C. Hsu, **"Data Path Allocation Based on Bipartite Weighted Matching"**, Proc. 27th Design Automation Conference, 1990.

[Hu61] T.C. Hu, **"Parallel Sequencing and Assembly Line Problems"**, Operations Research, pp. 841-848, novembre, 1961.

[ITU93] ITU-T Z.100, **"Functional Specification and Description Language"**, Recommendation Z.100 - Z.104, mars 1993.

[JDKR97] A.A. Jerraya, H. Ding, P. Kission, et M. Rahmouni, **"Behavioral Synthesis and Component Reuse with VHDL"**, Kluwer Academic Publishers, 1997.

[JeOB94] A.A. Jerraya et K. O'Brien, **"SOLAR: An Intermediate Format for System-Level Modeling and Synthesis"**, Computer Aided Software/Hardware Engineering, IEEE Press 1994.

[KCGPT98] K. Küçükçakar, C.T. Chen, J. Gong, W. Philipsen et T.E. Tkacik, **"Matisse: An Architectural Design Tool for Commodity ICs"**, IEEE Design & Test of Computers, avril/juin, 1998.

[Keu94] K. Keutzer, **"The impact of CAD on the Design of Low Power Digital Circuits**", IEEE Symposium on Low Power Electronics, Oct. 10-12, san Diego, 1994.

[Kiss96] P. Kission, **«Exploitation de la Hiérarchie et de la Réutilisation de blocs par la Synthèse de Haut Niveau»**, Thèse de doctorat, INPG, Grenoble, 1996.

[KJRD93] F. J. Kurdahi, P. Jha, C. Ramachandran, et N. Dutt, **"Towards More Realistic Physical Design Models for High Level Synthesis"**, Workshop on Synthesis And System Integration of Mixed Technologies, 1993.

[Kna96] D. Knapp, **"Behavioral Synthesis"**, Prentice Hall, 1996.

[KnPar91] D. Knapp, A. Parker, **"The ADAM design planning engine"**, IEEE Trans. on Computer-Aided Design, vol. 10, no. 7, 829-46, 1991.

[Kroli98] S. Krolikoski, **"Considerations on the Usability of Behavioral Synthesis"**, Design, Automation and Test in Europe, 1998.

[KuDu97] N. Dutt, S. Ohm et F. Kurdahi, **"A Unified Lower Bound Estimation Technique for High-Level Synthesis"**, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems , mai 1997.

[KuGaj93] F.J. Kurdahi, D.D. Gajski, C. Ramachandran et V. Chaiyakul, **"Linking Register-Transfer and Physical Levels of Design"**, IEICE Transactions on Information and Systems, septembre 1993.

[KuMic92] D. Ku et G. De Micheli, **"High-Level Synthesis of ASICs under Timing and Synchronization Constraints"**, Kluwer Academic Publishers, 1992.

[KuPa90] K. Küçükçakar et A.C. Parker, **"Data Path Tradeoffs using MABAL"**, Proceedings of the 28th Design Automation Conference, pp. 511-516, 1990.

[LaPot98] A. Srinivasan, G.D. Huber et D.P. LaPotin, **"Accurate Area and Delay Estimation from RTL Descriptions"**, IEEE Trans. on VLSI systems, vol. 6, no. 1, pp. 168-172, mars 1998.

[LEG90] T.A. Ly, W.L. Elwood, et E.F. Girczyc, **"A generalized interconnect model for data path synthesis"**, Proc. 27th Design Automation Conference, pp. 168-173, juin 1990.

[LHLin89] J.H. Lee, Y.C. Hsu, Y.L. Lin, **"A new Integer Linear Programming Formulation for the Scheduling Problem in Data Path Synthesis"**, Proc. of ICCAD, pp. 20-23, 1989.

[LiGup96] J. Li et R.K. Gupta, **"HDL Optimizations using Timed Decision Tables"**, Proc. of 33rd Design Automation Conference, Las Vegas, Nevada, juin 1996.

[LKMM95] T. Ly, D. Knapp, R. Miller et D. MacMillen, **"Scheduling using Behavioral Templates"**, Design Automation Conference, p.101-106, 1995.

[LMD94] B. Landwehr, P. Marwedel, et R. Dömer, **"OSCAR: Optimum Simultaneous Scheduling, Allocation and Resource Binding Based on Integer Programming"**, Proc. of European Conference on Design Automation, 1994.

[LMWV91] P. Lippens, J. van Meerbergen, A. van der Werf, W. Verhaegh, B. McSweeney, J. Huisken, O. McArdle, **"PHIDEO: a Silicon Compiler for High Speed Algorithms"**, EuroDAC, 1991

[Mar98] G. Marchioro, **«Découpage Transformationnel pour la Conception des Systèmes Mixtes Logiciel/Matériel»**, Thèse de doctorat, INPG, Grenoble, 1998.

[McFA90] M.C. McFarland, A.C. Parker. et R. Camposano, **"The High-Level Synthesis of Digital Systems"**, Proceedings of the IEEE, Vol. 78, No. 2, pp. 301-317, février, 1990.

[McFar86] M. McFarland, **"Using bottom-up design techniques in the synthesis of digital hardware from abstract behavioral specifications"**, in Proc. 23rd Design Automation Conference, pp. 474-480, IEEE/ACM, 1986.

[MDJ97] G.F. Marchioro, J.M. Daveau, A.A. Jerraya, **"Transformational Partitioning for Co-design of Multiprocessor Systems"**, IEEE/ACM International Conference on Computer Aided Design, ICCAD'97. San Jose, CA. pp. 508-515. 9-13 novembre, 1997.

[MehNä99] K. Mehlhorn et St. Näher, **"The LEDA Platform of Combinatorial and Geometric Computing"**, Cambridge University Press, 1999.

[MGKP97] J. Madsen, J. Grode, P. V. Knudsen, M. E. Petersen, A. Haxthausen, **"LYCOS: the Lyngby Co-Synthesis System"**, Kluwer Journal for Design Automation for Embedded Systems, pp. 195-235, vol. 2, n° 2, mars 1997.

[MoBr92] C. Monahan, F. Brewer, **"Communication Driven Interconnect synthesis"**, Proc. 6th Int. Workshop on High-Level Synthesis, 1992.

[MoBr96] C. Monahan, F. Brewer, **"Concurrent Analysis Techniques for Data Path Timing Optimization"**, Proc. 33rd Design Automation Conference, juin 1996.

[Najm98] M. Nemani et F.N. Najm, **"Delay Estimation of VLSI Circuits from a High-Level View"**, Proc. of the 35th Design Automation Conference, juin 1998.

[ORJ93] K. O'Brien, M. Rahmouni et A.A. Jerraya, **"DLS: A scheduling algorithm for high-level synthesis in VHDL"**, Proc. of the European Conference on Design Automation, Paris, France, février 1993.

[OtBr98] R.H.J.M. Otten et R.K. Brayton, **"Planning for Performance"**, Proc. 35th Design Automation Conference, 1998.

[PaDu95] P. Panda et N. Dutt, **"1995 High Level Synthesis Design Repository"**, International Symposium on System Synthesis, 1995.

[PaKn89] P.G. Paulin, J.P. Knight, **"Force-Directed Scheduling for the Behavioral Scheduling of ASICs"**, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 8, n° 6, juin 1989.

[Park79] A. Parker et al., **"The CMU Design Automation System: An Example of Automated Data Path Design"**, Proc. of the 16th Design Automation Conference, San Diego, Ca., USA, 1979.

[Park92] I. Park, **"AMICAL : Un Assistant pour la Synthèse et l'Exploration Architecturale des Circuits de Commande"**, Thèse de Doctorat, TIM3 – Institut National Polytechnique de Grenoble, 1992.

[RaKur94] C. Ramachandran et F.J. Kurdahi, **"Incorporating the Controller effects during Register-Transfer Level Synthesis"**, Proc. of the European Design Automation Conference, 1994.

[RaKur94b] C. Ramachandran et F.J. Kurdahi, **"Combined Topological and Functionality-Based Delay Estimation Using a Layout-Driven Approach for High-Level Applications"**, IEEE Trans. On CAD of Integrated Circuits and Systems, vol. 13, no. 12, pp. 1450-1460, décembre 1994.

[RaMan87] J. Rabaey, H. De Man, J. Vanhoof, G. Goossens and F. Catthoor, **"CATHEDRAL-II : A Synthesis System for Multiprocessor DSP Systems**", dans Silicon Compilation, Addison-Wesley, décembre 1987.

[ROA94] M. Rahmouni, K. O'Brien, et A.A. Jerraya, **"A Loop-based Scheduling Algorithm for Hardware Description Languages"**, Parallel Processing Letters, 4(3): 351-364, 1994.

[Rumb91] Rumbaugh James, et al., **"Object-Oriented Modeling and Design"**, Prentice Hall, 1991.

[SIA97] Semiconductor Industry Association**, The National Technology Roadmap for Semiconductors**, 1997.

[Some92] Q. Ji, Y. Oh, M. Lightner et F. Somenzi, **"Technology Independent Estimation of Area in Logic Synthesis"**, Proc. of Synthesis and Simulation Meeting and International Interchange (SASIMI), pp. 171-180, avril 1992.

[SSV96] K. Suzuki, A. Sangiovanni-Vincentelli, **"Efficient Software Performance Estimation Methods for Hardware/Software Codesign"**, In Proceeding of Design Automation Conference, juin 1996.

[ToWi77] H.C. Torng et N.C. Wilhelm, **"The optimal interconnection of circuit modules in microprocessor and digital system design"**, IEEE Trans. on Computers, 26 (5) pp. 450-45, 1977.

[TsHsu92] F. Tsai et Y. Hsu, **"STAR: a system for hardware allocation in data path synthesis"**, IEEE Trans. CAD, p 1053-1064, septembre 1992.

[TsSi86] C.J. Tseng abd D.P. Siewiorek, **"Automated Synthesis of Data Path in Digital Systems"**, IEEE Trans. on Computer-Aided Design, Vol. 5, N°. 3, pp. 379-95, juillet 1986.

[VaGaj95] F. Vahid et D.D. Gajski, **"Incremental Hardware Estimation during Hardware/Software Functional Partitioning"**, IEEE Trans. on VLSI Systems, vol. 3, no. 3, pp. 459-461, septembre 1995.

[Vah95] F. Vahid, **"Procedure Exlining: A Transformation for Improved System and Behavioral Synthesis"**, International Symposium on System Synthesis, 1995.

[VHDLT98] **"IEEE 1076.6 VHDL Synthesis Standard Passes Balloting"**, VHDL Times, vol. 7, n° 3, 1998.

[VRBG93] J. Vanhoof, K.V. Rompaey, I. Bolsens, G. Goossens, et H. De Man, **"High-Level Synthesis for Real-time Digital Signal Processing"**, Kluwer Academic Publishers, 1993.

[WalC91] R. A. Walker et R. Camposano, **"A Survey of High-Level Synthesis Systems"**, Kluwer Academic Publishers, Boston, Ma, 1991.

[WOC97] W.O. Cesário, P. Kission, P. Guillaume and A.A. Jerraya, **"Unified Evaluation Model for Interconnection Schemes Used in Behavioral Synthesis"**, International Workshop on Logic and Architectural Synthesis, Grenoble, France, 1997.

[WOC99] W.O. Cesário, Z. Sugar, R. Suescun and A.A. Jerraya, **"Overlap and Frontiers Between Behavioral and RTL Synthesis"**, Design, Automation and Test in Europe, Munich, Allemagne, mars, 1999.

[WOC99b] W.O. Cesário, **"Environnement de Synthèse Flexible"**, Colloque CAO de Circuits Intégrés et Systèmes, Aix en Provence, France, mai, 1999.

[YeEr95] W. Ye et R. Ernst, **"Worst Case Timing Estimation Based on Symbolic Execution"**, COBRA report '95, Institute of Computer Engineering, Technical University of Braunschweig, Germany, octobre 1995.

[Ywolf95] T.Y. Yen et W. Wolf, **"Communication Synthesis for Distributed Embedded Systems,"** dans Proceedings of 1995 IEEE International Conference on Computer-Aided Design.

Index Bibliographique

Index

RESUME

Le sujet de cette thèse est le développement d'une nouvelle méthodologie de synthèse basée sur une approche interactive et flexible conçue pour l'exploration de l'espace des solutions. C'est le concepteur qui est au centre du processus de création, il a la possibilité d'adapter les techniques et les méthodes de conception à l'application et il est guidé par des estimations de haute fidélité pour prendre des décisions pendant la phase de synthèse. La flexibilité concerne l'architecture cible et le flot de conception.

La flexibilité de l'architecture cible offre la possibilité de choisir le style des interconnexions selon le facteur de partage des ressources de chaque destination de données. De nouvelles techniques pour l'allocation et l'affectation des interconnexions ont été employées. Elles utilisent des directives d'optimisation visant à réduire le nombre de cellules.

L'existence d'un chevauchement de fonctionnalité entre la synthèse comportementale et la synthèse RTL a permis de nouvelles formes d'intégration entre les deux étapes. Un flot de synthèse qui profite de ce chevauchement est présenté. Les concepteurs peuvent ainsi éviter des étapes de synthèse inutiles ou non-optimales en fonction de l'application et les critères d'optimisation.

Une nouvelle méthodologie pour l'évaluation de performance à partir d'une description de niveau système est aussi présentée. Cette méthodologie est basée sur un modèle de performance exécutable décrit dans un langage de spécification au niveau système. Le modèle tient compte du partitionnement logiciel/matériel, des affectations des multiprocesseurs et du choix des protocoles de communication. Les résultats préliminaires montrent que ce modèle peut atteindre une bonne précision et une bonne fidélité. Toutes ces méthodologies et techniques de conception ont été développés dans le cadre de l'environnement de conception conjointe appelé MUSIC.

Mots-clés : Synthèse de haut niveau et synthèse au niveau système, estimation de performance, exploration de l'espace des solutions, outils et modèles de conception, modèles exécutables pour la spécification de systèmes.

ABSTRACT

This thesis develops a new synthesis methodology based on an interactive and flexible approach to design space exploration. The foundations of this methodology are: the creative process is done by the designer, design techniques and models can be adapted to the application under synthesis and high fidelity estimations guide the decision process during synthesis. This flexibility concerns the target architecture and the design flow.

Firstly, the flexibility of the target architecture is presented: it is the possibility of choosing interconnection schemes according to the resource-sharing factor of each data destination. New design techniques for interconnect allocation/binding with cell minimizing optimization directives were used.

New forms of integrating behavioral and RTL synthesis were made possible by the presence of a functionality overlap in some synthesis environments. A synthesis flow that profits from this overlap is presented. Designers can avoid useless or non-optimal synthesis steps according to the application and the optimization priorities.

A new methodology for performance estimation at the system level of abstraction is introduced. This methodology is based on an executable performance model described in a system-level specification language. The model takes into account hardware/software partitioning, multi-processor bindings and the selection of communication protocols. Preliminary results show that this model can achieve good precision and fidelity. All these design methodologies and techniques were implemented in the co-design environment called MUSIC.

Keywords: High-level and system-level synthesis, performance estimation, design space exploration, design models and tools, executable system-specification models.

ISBN 2-913329-33-0 broché
ISBN 2-913329-34-9 version électronique