Synthèse comportementale basée sur l'ordonnancement

Z. Sugar

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

THESE

pour obtenir le grade de

DOCTEUR DE L'INPG

Spécialité : Microélectronique

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

présentée et soutenue publiquement

par

Zoltán SUGÁR

le 15 Mai 2000

Titre :

Synthèse comportementale basée sur l'ordonnancement

Directeur de Thèse :
Ahmed Amine JERRAYA

JURY

M. :	Alain	GREINER	, Président
Mme. :	Anne	MIGNOTTE	, Rapporteur
M. :	Michel	AUGUIN	, Rapporteur
Mme. :	Polen	KISSION	, Examinateur
M. :	Ahmed Amine	JERRAYA	, Examinateur

à Renáta . . .

Résumé

L'objectif de cette thèse est de mettre au point une nouvelle méthodologie pour la synthèse comportementale. La synthèse comportementale traditionnelle peut être défini comme la compilation d'une spécification algorithmique en une architecture composée d'un chemin de données et d'un contrôleur. Le flux de synthèse comprend généralement l'ordonnancement, l'allocation, la génération du chemin de données et la synthèse du contrôleur. Les algorithmes de ces étapes de synthèse ont été intensivement étudiés dans la littérature alors que la recherche de l'intégration des outils de synthèse comportementale aux flux de conception existants ne fait pas partie de travaux de recherches. En dépit de grandes espérences, les outils de synthèse comportementale traditionnels ne sont jamais parvenus à être acceptés par les concepteurs.

Afin de répondre à ces problèmes, la redéfinition de la synthèse comportmenetale est donc nécessaire. Dans cette thèse, nous proposons un nouveau flux de synthèse puissant basé uniquement sur l'étape d'ordonnancement. Cet ordonnancement est étendu d'une part, d'une analyse de chemin de données pour l'ouverture vers des applications mixtes, d'autre part, d'une étape de génération du code permettant l'adaptation efficace de l'ordonnancement avec la synthèse au niveau transfert de registres. L'efficacité du nouveau flux est prouvée par deux applications complexes et industrielles, et par son intégration dans un flux de synthèse système.

Mots-clés : synthèse comportementale, synthèse au niveau transfert de registres, synthèse au niveau sysème, ordonnancement, flux de contrôle, flux de données, VHDL.

Abstract

The objective of this thesis is the elaboration of a new design methodology for the behavioral synthesis. Until now, the behavioral synthesis has been viewed as the process of synthetizing a datapath/controller from an algorithmic specification. This synthesis flows generally includes several steps: the scheduling, the allocation, the datapath generation and the controller synthesis. While all these phases have been extensively studied in the EDA domain, the integration of the behavioral synthesis tools in the existing synthesis environments has not yet got the adaquate attention. And despite a strong interest and demand, the current behavioral synthesis has never been widely adopted by the designers.

Solving this problem requires a redefinition of the behavioral synthesis. This thesis proposes a powerful behavioral synthesis flow only based on the scheduling phase. The scheduling is first extended by a datapath analysis for mixed applications, and then a generation efficiently ties the scheduled model to the RTL synthesis. This methodology has been integrated to a system level synthesis flow and its effectiveness has been evaluated on two industrial- scaled examples.

Keywords: behavioral synthesis, RTL synthesis, system level synthesis, scheduling, control flow, data flow, VHDL.

Sugár Zoltán, Mai 2000, 111 pages.

ISBN-2-913329-48-9 (Version papier)
ISBN-2-913329-47-0 (Version électronique)

Remerciement

Je tiens tout d'abord à exprimer mes remerciements à Monsieur Ahmed Amine Jerraya, Directeur de Recherche au CNRS, pour avoir dirigé mes recherches, pour sa disponibilité et pour sa participation à l'orientation de mes travaux de thèse.

Je remercie également Monsieur Bernard Courtois, Directeur du Laboratoire TIMA, pour m'avoir accueilli dans son laboratoire.

Je tiens à remercier Monsieur Alain Greiner, Professeur à l'Université Pierre et Marie Curie de Paris 6, pour m'avoir fait l'honneur de présider le jury.

Je remercie Madame Anne Mignotte, Professeur à l'Institut National des Sciences Appliquées de Lyon, Monsieur Michel Auguin, Professeur à l'Université de Nice pour m'avoir fait l'honneur d'être rapporteurs de cette thèse ainsi que Madame Polen Kission pour sa participation au jury.

Je tiens à adresser mes remerciements à mes anciens et nouveaux collègues : Maher Rahmouni pour m'avoir fait l'introduction à mon sujet de thèse ; Elisabeth Berrebi et Imed Moussa dont les travaux ont été indispensables à l'évaluation de mes résultats ; Philippe Lemarrec et Philippe Guillaume pour leur gentillesse et leur agréable compagnie ; Adel, Bernard, Claudine, Pascal, Rodolphe, Thierry G. et Thierry R. pour leur patience à corriger mon manuscrit de thèse ; Olivier D., Jérôme et Wander pour leur humeur toujours sympathique ; Olivier P. pour ses blagues inépuisables. Je remercie également Gabriela pour m'avoir beaucoup aidé pour le bon déroulement de ma soutenance.

J'adresse mes remerciements au reste du laboratoire TIMA, notamment à Sonja pour son aide parfois inestimable.

Je présente mes reconnaissances à mes parents pour leur encouragements depuis mon pays natal.

Finalement, je me dois d'adresser mes plus sincères remerciements à Renáta pour sa patience infinie, pour sa soutenance et pour tout ce que nous avons partagé pendant les trois années passées à Grenoble.

Table des matières

Chapitre 1

Présentation de la thèse

Introduction

Depuis plus de 20 ans, la recherche en synthèse comportementale est intensive. Certains outils commerciaux ont fait leur apparition sur le marché et sont déjàá disponibles. Par contre, en milieu industriel, le degré d'acceptation de ces outils n'a jamais réussi à atteindre celui de la méthodologie de la synthèse au niveau transfert de registres. Le fossé de productivité en croissance représente une nouvelle opportunité pour la synthèse comportementale puisqu'il devient de plus en plus nécessaire d'opter pour de nouvelles méthodologies. En théorie, la synthèse comportementale apporte un ensemble d'avantages importants : cycle de développement court, estimation de performance à un niveau d'abstraction plus élevé, réutilisation plus facile des conceptions existantes, indépendance de la technologie cible, etc. Par contre, certaines limitations ont empêché que la synthèse comportementale soit répandue pour les applications industrielles et complexes. En général, les résultats de la synthèse sont difficiles à comprendre et à prévoir, le modèle d'entrée est trop restreint, l'intégration de la synthèse comportementale aux flux de conception existants pose de sérieux problèmes et, dans certains cas, les outils de synthèse au niveau transfert de registres produisent des résultats plus efficaces. Il est donc nécessaire d'introduire une nouvelle génération d'outils de synthèse comportementale pour surpasser les problèmes posés par les outils classiques.

Outre les problèmes décrits ci-dessus, la synthèse comportementale n'est jamais parvenue à exploiter entièrement les capacités de la synthèse au niveau transfert de registres. Dans une approche classique, la synthèse comportementale effectue l'ordonnancement, l'allocation, l'association des unités fonctionnelles et produit une architecture composée d'un contrôleur et d'un che-

min de données. L'ordonnancement décompose la description initiale en une machine d'états finis de haut niveau dont chaque transition peut être exécutée en un cycle d'horloge. L'étape d'allocation et celle d'association des unités fonctionnelles rajoute à ce modèle les informations concernant le partage des ressources. En général, l'étape suivante décompose ce modèle en deux composants : le contrôleur sous forme d'une machine d'états finis au niveau transfert de registres et le chemin de données contenant les unités fonctionnelles capables d'exécuter les opérations de la description comportementale.

Seul l'ordonnancement est spécifique à la synthèse comportementale ; les autres étapes du flux peuvent être accomplies à l'aide d'outils de synthèse au niveau transfert de registres modernes, ce qui est illustré sur la figure 1.1. Le modèle d'entrée de la synthèse au niveau transfert de registres peut être la machine d'états finis de haut niveau produite par l'ordonnancement ou l'architecture spécifiée complètement. Les deux modèles ont une spécificité commune : la synchronisation des opérations s'effectue à l'aide d'une horloge, autrement dit, les descriptions sont définies avec exactitude du cycle d'horloge. En effet, cette précision est atteinte après l'étape d'ordonnancement. Ainsi la machine d'états finis de haut niveau produite par l'ordonnancement permet-elle une intégration de la synthèse comportementale avec la synthèse au niveau transfert de registres plus tôt dans le flux de conception. Dans ce cas, l'allocation de ressources ainsi que l'association des unités fonctionnelles sont effectuées par la synthèse au niveau transfert de registres.

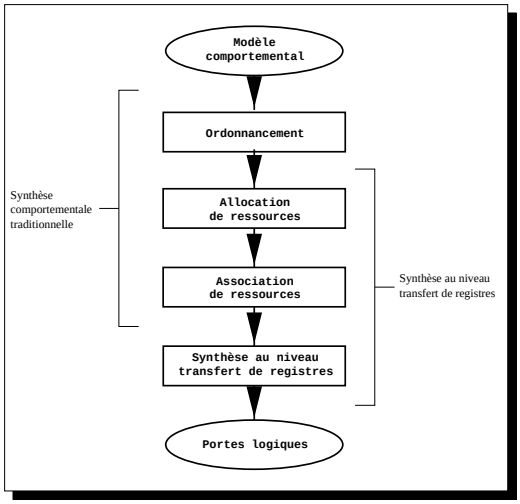


Figure 1.1: Synthèse au niveau comportemental et transfert de registres

Les recherches menées dans le cadre de cette thèse s'appuient sur cette nouvelle redéfinition de la synthèse comportementale. L'approche et les algorithmes de synthèse ont été prouvés par des applications complexes et industrielles et par l'intégration de l'outil à un flux de synthèse système qui a connu des succès commerciaux.

1.1 Objectifs

L'objectif de cette thèse est double : d'une part, étudier des méthodes d'ordonnancement utilisables par plusieurs domaines d'application, d'autre part, analyser une meilleure intégration des outils de synthèse comportementale aux flux de conception existants.

La plupart des algorithmes d'ordonnancement s'intéressent à la synthèse des circuits orientés flux de données tels que les applications de traitement du signal. D'autres approches se concentrent uniquement sur des circuits dominés par le flux de contrôle sans optimisations particulières de la partie opérative. Cependant les circuits modernes exigent des traitement plutôt mixtes permettant la combinaison d'une partie de contrôle et des blocs réalisant des calculs arithmétiques complexes. Il est donc nécessaire d'analyser la possibilité de combiner les deux approches d'ordonnancement pour parvenir à assurer des solutions pour les circuits modernes.

Pour pouvoir redéfinir certains principes de la synthèse comportementale présentés ci-dessus, il est nécessaire d'étudier de manière approfondie les raisons précises de l'échec des outils de synthèse classiques. Comme l'une de ces raisons est la mauvaise intégration de la synthèse comportementale avec la synthèse au niveau transfert de registres, l'analyse concernant les nouvelles capacités de la synthèse logique est également indispensable. Ces études nous permettent d'établir un nouveau flux de synthèse capable d'exploiter les optimisations de haut niveau de la synthèse au niveau transfert de registres.

1.2 Contributions

Les principales contributions de cette thèse sont les suivantes :

- Une étude du flux de conception général des outils utilisés pour la conception des systèmes complexes avec un accent particulier sur l'insertion des outils de synthèse comportementale aux flux existants. Cette étude montre que la plupart des outils de synthèse comportementale connus ne sont pas parvenus à établir une interface efficace avec leur environnement, ce qui entraîne généralement des défaillances en matière de performance.

- Une étude approfondie de l'interprétation des diverses structures du langage de description de matériel VHDL ainsi que la définition d'un sous-ensemble relativement étendu, principalement destiné à la synthèse comportementale.

- La mise en oeuvre d'un outil de synthèse comportementale de nouvelle génération permettant de profiter des nouvelles capacités et de l'évolution de la synthèse au niveau transfert de registres.

- L'intégration de notre outil de synthèse comportementale à un flux de conception système mixte logiciel/matériel et la définition d'un modèle d'entrée étendu pour la représentation des systèmes hiérarchiques.

1.3 Plan de la thèse

Le reste de ce chapitre décrit la structure de la thèse.

Le chapitre 2 donne une vue globale sur la synthèse comportementale en présentant les principaux algorithmes et les représentations intermédiaires utilisées par les étapes de synthèse.

Le chapitre 3 discute les raisons probables de l'échec des outils de synthèse comportementale dits traditionnels. Comme exemple, deux outils de cette catégorie, Behavioral Compiler et AMICAL, sont analysés.

Le chapitre 4 aborde le VHDL, le langage de description de matériel le plus répandu pour la synthèse comportementale. La définition du sous-ensemble accepté par la synthèse comportementale et l'interprétation de diverses structures pour la réalisation des circuits sont aussi détaillés.

Le chapitre 5 introduit une nouvelle approche de la synthèse comportementale qui est mieux adaptée aux flux de conception modernes.

Le chapitre 6 présente les principaux domaines d'application de notre outil de synthèse. L'efficacité de notre approche est prouvée par deux applications complexes et industrielles. Ce chapitre montre également la nouvelle dimension de l'utilisation des techniques de synthèse comportementale dans le cadre d'un outil de synthèse système.

Le chapitre 7 analyse les résultats et montre les perspectives et les améliorations à apporter aux travaux menés dans le cadre de cette thèse.

Chapitre 2

Introduction à la synthèse comportementale

Introduction

La synthèse comportementale, autrement dit la synthèse de haut niveau correspond à un ensemble de tâches de raffinements ou de transformations qui, à partir d'une spécification comportementale, génèrent une architecture cible décrite souvent par des descriptions au niveau transfert de registres. Les étapes de synthèse, selon leurs algorithmes, utilisent des modèles d'architecture et des représentations internes.

La figure 2.1 illustre le flux général d'un outil de synthèse comportementale. La description comportementale peut être générée automatiquement par un outil de synthèse de plus haut niveau ou décrite par le concepteur à partir d'une spécification. Elle est d'abord compilée en une représentation intermédiaire qui dépend des algorithmes utilisés dans l'outil. Les étapes de synthèse s'appliquent ensuite à cette représentation et à partir de laquelle d'autre structures de données sont générées : graphes, machines d'états finis, architectures, etc. En général, ces étapes sont pilotées par des contraintes d'utilisation qui influencent certaines caractéristiques des algorithmes ainsi que l'architecture résultante.

Le but de ce chapitre est la définition des principes de base de la synthèse comportementale : les formats intermédiaires ainsi que les algorithmes. La principale contribution des travaux menés dans le cadre de cette thèse étant l'établissement d'un nouveau flux de conception basé sur l'ordonnancement, l'accent est mis sur cette étape de synthèse.

Comme une des raisons de l'échec de la synthèse comportementale est la mauvaise intégration aux flux de conception existants, nous mettons en relief

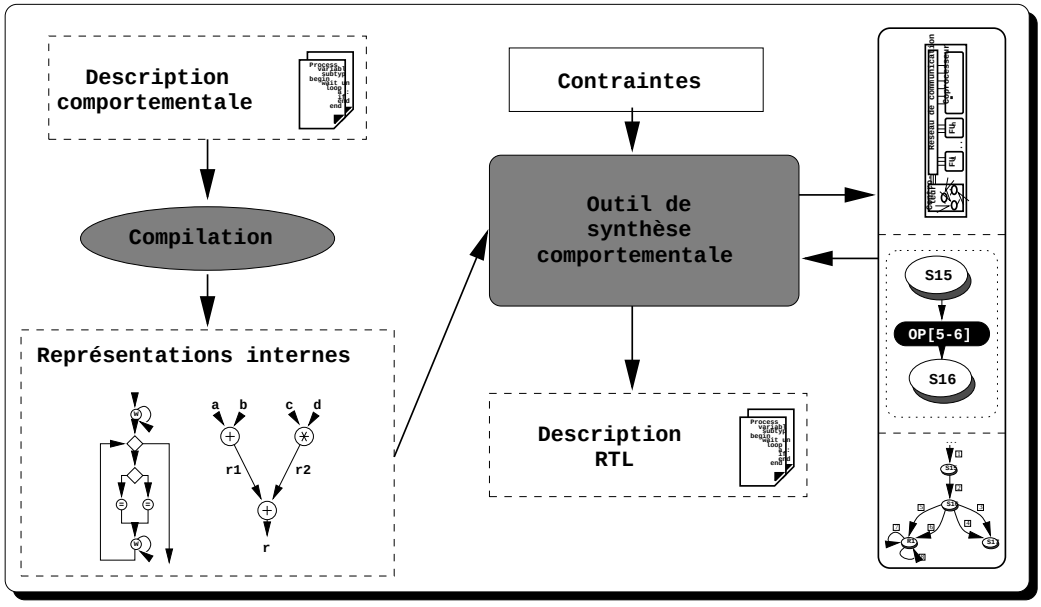


Figure 2.1: Flux de synthèse comportementale général

son lien avec d'autres outils de conception. La section 2.2 présente les principaux domaines d'application pour la synthèse comportementale ainsi que leurs représentations internes les plus appropriées. Les différentes étapes de synthèse sont ensuite discutées ; l'accent est mis sur les algorithmes d'ordonnancement dont les caractéristiques sont présentées en fonction de domaines d'application visé.

2.1 Abstraction au niveau comportemental

D'après [JDKR97], un circuit décrit au niveau comportemental est défini en termes d'étapes de calcul. La spécification est pilotée par les événements extérieurs et composée typiquement d'un ensemble de protocoles de communication pour l'échange de données avec l'environnement et de calculs internes. Les étapes de calcul sont constituées d'un ensemble d'opérations exécutées entre deux points de synchronisation successifs. L'exécution d'une étape de calcul peut durer plusieurs cycles d'horloge. La synthèse comportementale décompose ces étapes de calcul et génère une description au niveau transfert de registres. En général, les descriptions comportementales sont directement composées par le concepteur à l'aide d'un langage de description de matériel ou générées automatiquement par des outils de synthèse de

plus haut niveau. Ainsi, pour une meilleure intégration avec ces outils, est-il nécessaire d'étudier les caractéristiques principales des outils de synthèse au niveau transfert de registres. Dans le chapitre 6, l'intégration de notre outil de synthèse comportementale à un flux de synthèse au niveau système est présentée.

2.1.1 Lien avec la synthèse au niveau transfert de registres

Actuellement, la méthodologie la plus répandue pour la conception des circuits intégrés est la synthèse au niveau transfert de registres et l'optimisation logique. Ce type de synthèse permet la compilation d'une description au niveau transfert de registres en une réalisation au niveau des portes logiques définies par la technologie. Le format intermédiaire utilisé par ce type d'outils est généralement composé d'un réseau combinatoire réalisant les opérations qui doivent être exécutées pendant un cycle d'horloge et d'un ensemble de registres permettant la mémorisation des données intermédiaires. La première partie de la synthèse consiste à extraire ces structures de la description initiale. Cette architecture intermédiaire est illustrée par la figure 2.2. Les réseaux combinatoires se situent entre les blocs de registres. La deuxième phase de la synthèse optimise ces parties combinatoires à l'aide de techniques d'optimisation booléenne qui sont guidées par des contraintes telles que la surface, le délai du chemin critique, la testabilité, la routabilité, la consommation, etc. Les critères d'optimisation étant souvent contradictoires, le but est de trouver une réalisation dans l'espace de solutions qui respecte les contraintes imposées par le concepteur. Comme ces optimisations n'insèrent pas de cycles d'horloge supplémentaires, l'exécution de la description générée demande le même nombre de cycles d'horloge que la description initiale.

Le modèle d'entrée au niveau transfert de registres est très restreint par rapport au niveau comportemental. Pour chaque type d'architecture, par exemple les machines d'états finis, le style d'écriture est fixé par l'outil de synthèse. Ce style d'écriture strict assure l'efficacité de l'optimisation logique. L'éloignement du style d'écriture provoque généralement des résultats moins satisfaisants.

Les outils de synthèse comportementale n'ayant pas encore obtenu le succès escompté, les outils de synthèse logique ont été de plus en plus étendus par des techniques d'optimisation de haut niveau. La provenance de ces techniques est essentiellement le domaine de la synthèse comportementale. Par exemple, en 1999, Synopsys a développé Design Compiler Ultra [Syn99], un nouveau sur-ensemble du Design Compiler traditionnel étendu par de

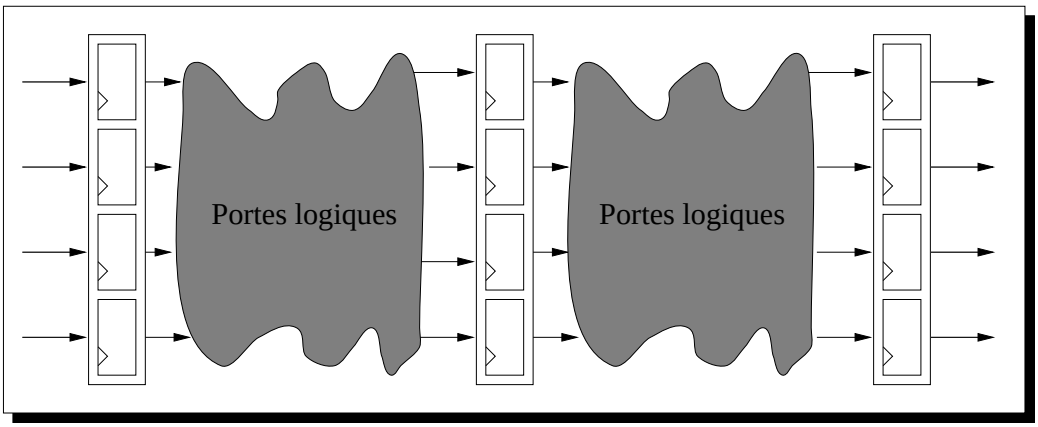


Figure 2.2: Architecture intermédiaire pour la synthèse au niveau transfert de registres

nouvelles optimisations de haut niveau. Ces optimisations comprennent des algorithmes qui font généralement partie des outils de synthèse comportementale : partage des ressources pour les opérations, transformations complexes des expressions arithmétiques avec allocation des unités fonctionnelles (BOA[1]), reordonnancement des opérations (BRT[2]) pour minimiser le chemin critique, etc. L'intégration de ces extensions avec la synthèse logique permet des optimisations beaucoup plus puissantes que ce que l'on obtient au niveau comportemental. Comme présenté dans le chapitre 3, les outils de synthèse comportementale, générant des description au niveau transfert de registres, devraient tenir compte de ces évolutions.

2.2 Domaines d'application de la synthèse comportementale

La restriction d'un outil de synthèse comportementale à un domaine d'application permet d'atteindre des résultats efficaces grâce à la réduction de la complexité du processus de synthèse. On distingue deux domaines d'application dans la littérature :

Applications dominées par le flux de données. Ces applications sont caractérisées par des flux réguliers. En d'autres termes, les entrées et les sorties de l'application forment des flux réguliers de données. Le comportement d'une telle application est généralement spécifié comme

[1]En anglais : BOA : Behavioral Optimisation of Arithmetic
[2]En anglais : BRT : Behavioral Retiming.

un ensemble périodique d'opérations. Ces opérations sont exécutées sur chaque nouvel ensemble de données. Les entrées et les sorties forment des signaux à débit fixe. Les applications les plus typiques sont les filtres numériques, les codeurs/décodeurs, les circuits réalisant les algorithmes de cryptage.

Applications dominées par le flux de contrôle. Ces applications sont pilotées par un ensemble de commandes à interpréter. Chaque commande, séquence de contrôle, donnée, peut engendrer une lecture ou écriture de structure de données complexe et l'exécution d'un algorithme spécifique à la commande. Les instructions exécutées peuvent dépendre des données, de la communication avec l'environnement, des interruptions, etc. Les applications typiques sont les contrôleurs, convertisseurs de protocoles, etc.

La conception des systèmes complexes nécessite souvent la combinaison de composants orientés flux de données et de ceux orientés flux de contrôle. Afin de pouvoir traiter ce type d'applications, le système doit être décomposé en modules orientés flux de contrôle et modules orientés flux de données. Ce découpage est effectué généralement par les concepteurs pendant la spécification du système et il permet l'utilisation des outils de synthèse optimisés aux domaines d'application spécifiques [Ber97].

Pour ces deux principaux domaines d'application, deux types de représentations internes ont été conçus : les graphes de flux de contrôle et les graphes de flux de données. Pour le problème de la synthèse des systèmes mixtes, d'autres solutions sont présentées dans la section 2.2.3 qui décrivent des représentations plus complexes, plus ou moins adaptées à tous les domaines d'application.

2.2.1 Graphe de flux de contrôle

Le modèle de flux de contrôle tient ses origines de la machine conventionnelle de Von Neumenn dans laquelle, contrôlées par un compteur ordinaire, les instructions sont sélectionnées pour une exécution séquentielle. Un graphe de flux de contrôle est un graphe $G = (V, E)$ où les sommets correspondant à ce que l'on appelle des blocs de base et les arcs décrivent le flux de contrôle. Un bloc de base est une séquence d'instructions ne contenant aucune instruction de contrôle. Dans cette section, les blocs de base ne sont plus considérés comme des sommets, mais chaque instruction représente un sommet.

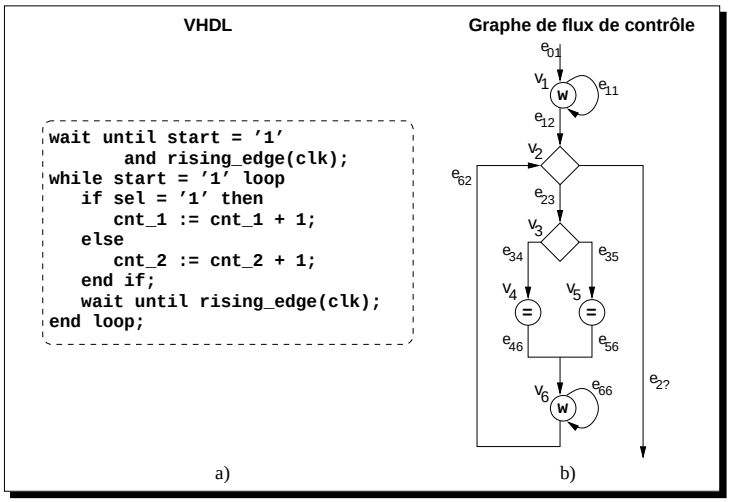


Figure 2.3: Graphe de flux de contrôle

Définition (Graphe de flux de contrôle (CFG [3])) Un graphe de flux de contrôle est un graphe orienté $G = (V, E)$, où :

- $V = \{v_1, \ldots, v_n\}$ *est un ensemble de sommets, et*
- $E \subset V \times V$ *est une relation de flux de contrôle dont les éléments sont des arcs orientés de séquencement.*

La figure 2.3(a) montre la description d'un compteur simple commandé par deux signaux : *start* et *sel* permettant sélectionner entre les deux compteurs. Le graphe de flux de contrôle est aussi illustré par la figure 2.3(b); il consiste en 6 sommets $V = \{1, \ldots, 6\}$ représentant les instructions de l'algorithme et 10 arcs $E = \{e_{01}, e_{11}, e_{12}, e_{23}, e_{2?}, e_{34}, e_{35}, e_{46}, e_{56}, e_{66}, e_{62}\}$, représentant la relation de précédence entre ces instructions.

Définition (Sommets du graphe de flux de contrôle) Les sommets sont de trois classes :

- *Les sommets d'instruction représentent les instructions simples telles que les affectations de variables, les opérations arithmétiques et logiques et les appels de procédures. Cette classe de sommets est représentée par le sous-ensemble de V, V_0.* Pour l'exemple de la figure 2.3, $V_0 = \{4, 5\}$.
- *Les sommets de branchement modélisent les instructions conditionnelles telles que IF, CASE, WHILE et sont représentées par le sous-ensemble V_b. Pour l'exemple de la figure 2.3, $V_b = \{2, 3\}$.*

[3] En anglais : Control Flow Graph.

- *Les sommets de synchronisation correspondent à une attente d'événement extérieur tel qu'un changement de valeur d'un signal. Cette classe de sommets est représentée par le sous-ensemble* V_a. *Pour l'exemple de la figure 2.3,* $V_a = \{1, 6\}$.

Les sommets d'opérations possèdent un seul successeur immédiat alors que les sommets de branchements peuvent en avoir plusieurs. Les sommets de synchronisation ont deux successeurs dont le premier est le même sommet modélisant l'attente par un rebouclage.

Définition (Arcs du graphe de flux de contrôle) : Un arc (v_i, v_j) *dans un graphe de flux de contrôle représente une relation de séquencement entre deux instructions. Un arc* (v_i, v_j) *est étiqueté par une expression booléenne* $Cond_{ij}$ *et signifie que l'instruction représentée par* v_j *sera exécutée si l'instruction représentée par* v_i *est exécutée et que l'expression* $Cond_{ij}$ *est vrai. Il est facile de remarquer que si un sommet* v_i *représente une instruction simple, en d'autres termes* $v_i \in V_o$ *et* v_j *est le successeur de* v_i *alors* $Cond_{ij} = vrai$.

Dans ce modèle de graphe de flux de contrôle, il est supposé que les expressions des arcs issues d'un sommet représentant une instruction de branchement sont exclusives deux à deux et que pour un échantillon de données quelconque, une seule des expressions est *vraie*. D'une manière plus formelle, si v_i est un sommet représentant une instruction de branchement et que $v_1, \ldots, v_n$ sont les k successeurs immédiats de v_i, alors :

- $Cond_{i,i_m} \bigwedge Cond_{i,i_h} = faux$, pour tout $m, h \in \{1, \ldots, k\}$ et $m \neq h$.
- $\sum_{n=1}^{k} Cond_{i,i_n} = vrai$.

2.2.2 Graphe de flux de données

Les graphes de flux de données sont l'une des représentations les plus utilisées dans le domaine de la synthèse comportementale. La plupart des algorithmes et des outils de synthèse reposent sur cette représentation. En effet, l'évolution des ordinateurs vers des architectures massivement parallèles a inspiré de nouveaux modèles de calcul conformes à la nature parallèle du matériel. Les sommets du graphe de flux de données représentent des *opérations* telles que les opérations arithmétiques et logiques alors que les arcs représentent des *valeurs*. Les arcs sont généralement associés aux variables mémorisant les valeurs intermédiaires dans un calcul complexe. Ainsi pouvons-nous également considérer les arcs comme une représentation des dépendances de données entre les diverses opérations qui déterminent le parallélisme et les séquences des opérations.

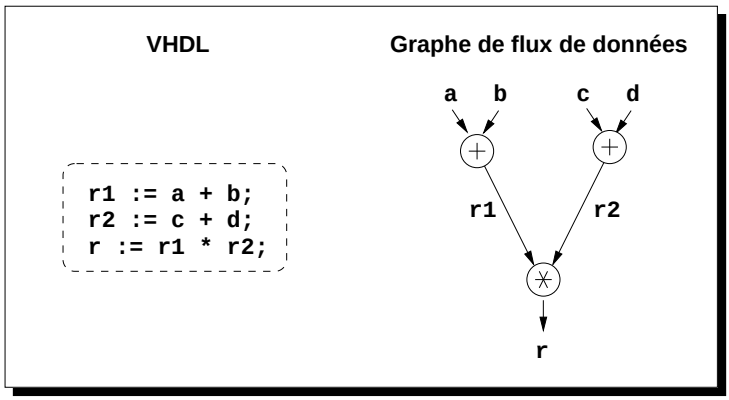


Figure 2.4: Graphe de flux de données

Définition (Graphe de flux de données (DFG[4])) Un graphe de flux de données est un graphe orienté $G = (V, E)$ *où :*

- $V = \{v_1, \ldots, v_n\}$ *est un ensemble fini de sommets, et*
- $E \subset V \times V$ *est une relation asymétrique de dépendance de données dont les éléments sont des arcs orientés.*

Les sommets correspondent aux opérations et un arc orienté e_{ij} de $v_i \in V$ à $v_j \in V$ existe, si la donnée produite par l'opération o_i (représentée par v_i) est consommée par l'opération o_j (représentée par v_j). Un exemple d'un graphe de flux de données est illustré par la figure 2.4. Il représente le calcul de l'expression $r = (a + b) * (c + d)$. Ce graphe est composé de 3 sommets et de 7 arcs nommés selon les noms des variables intermédiaires.

2.2.3 Applications mixtes

À part quelques applications spécialisées, les spécifications comportementales contiennent la combinaison de parties de contrôle décrivant des protocoles de communication et de parties dominées par des opérations de calcul. Les deux représentations pures présentées ci-dessus que ce soit le graphe de flux de contrôle ou celui de données ne permettent généralement pas une représentation efficace de ces descriptions complexes. Un graphe de flux de données n'admet aucune structure de contrôle ; un graphe de flux de contrôle ne représente pas les dépendances de données. Il exclut donc la possibilité de certaines optimisations orientées flux de données. Ainsi dans le premier cas,

[4]En anglais : Data Flow Graph.

les domaines d'application sont extrêmement restreints, dans le deuxième cas, les résultats de synthèse restent médiocres.

C'est pour cette raison que la recherche de la synthèse comportementale s'est intéressée à des représentations plus complexes et plus propices à des applications réelles et industrielles. La plupart des solutions constituent à étendre et/ou à combiner les deux approches.

2.2.3.1 Modèles de représentation pour les applications mixtes

La majorité des outils de synthèse comportementale reposent sur une représentation interne mixte nommée *graphe de flux de donnée et de contrôle* (CDFG[5]). Cette représentation étend le modèle pur du graphe de flux de données par les sommets de contrôle pour les instructions conditionnelles (if, case, while, etc.). Le modèle de CDFG est propice à des applications dominées principalement par le flux de données. Sans que la liste des outils soit exhaustive, cette approche est utilisée par plusieurs outils de synthèse tels que BC[Kna96], CATHEDRAL[MCG+90], HIS[CBH+91], HERCULES[MK88], CALLAS[BK92], etc.

Nous pouvons distinguer deux types de CDFGs. Le premier type étend le graphe de flux de données par les nouveaux sommets capables de représenter les branchements[6] et les jonctions[7]. D'après [Sto91, Jon93], les sommets de branchement sont toujours conditionnels. Leur comportement décrit le passage des données de leur port d'entrée à un de leur ports de sortie suivant l'évaluation de la condition. Le comportement des sommets de jonction est l'inverse des branchements ; ils assurent le passage des données d'un de leur ports d'entrée à leur port de sortie. Ainsi les structures de contrôle sont-elles intégrées dans le graphe de flux de données.

Le deuxième type de CDFGs est fondé sur la séparation des opérations de contrôle et des blocs de base ne contenant que des opérations de calcul. Ces blocs de base sont représentés par un graphe de flux de données et leur exécution et conditionnée par les opérations conditionnelles. Ce type de CDFG est utilisé dans [OG86b, LG88].

D'autres représentations internes existent encore qui sont spécifiques à un outil de synthèse. MIMOLA [Mar86] utilise une représentation arborescente similaire aux arbres syntaxiques des compilateurs logiciels. CATHEDRAL [Not91] de l'IMEC a conçu une représentation appelée *graphe de flux de signaux*[8] dérivée directement de langage *Silage* [Hil85], un langage spécifique

[5]En anglais : CDFG : Control-Data-Flow Graph.
[6]En anglais : fork node.
[7]En anglais : merge node.
[8]En anglais : Signal Flow Graph.

aux applications de traitement du signal.

2.3 Approche traditionnelle pour la synthèse comportementale

La synthèse comportementale est une méthodologie de conception qui accepte comme modèle d'entrée une spécification comportementale ou fonctionnelle et produit une architecture capable d'exécuter cette spécification initiale. L'architecture résultante est généralement décrite au niveau transfert de registres et composée d'un contrôleur et d'un chemin de données. La description comportementale définit la fonctionnalité qui doit être réalisée par le système synthétisé. Cette description peut avoir une représentation graphique ou textuelle. Un ensemble d'algorithmes de synthèse s'applique sur cette représentation pour transformer la structure initiale graduellement vers une architecture cible. Dans un flux de synthèse traditionnel, ces étapes comprennent la compilation de la description comportementale en une représentation interne, l'ordonnancement, l'allocation de ressources, l'association des unités fonctionnelles aux opérations et finalement, la génération de l'architecture. Le but de cette section est de développer les spécificités principales des diverses étapes de la synthèse comportementale. Dans [JDKR97, Rah97], ces algorithmes généraux sont détaillés dans le cadre de la présentation d'un outil de synthèse comportementale complet.

2.3.1 Description comportementale

La description d'entrée d'un outil de synthèse comportementale décrit la fonctionnalité d'un circuit. En général, c'est une description algorithmique du comportement du circuit qui ne contient aucune information ni sur sa structure ni sur la façon dont il sera réalisé. Pour la spécification d'entrée, les concepteurs utilisent généralement un langage de description de matériel (HDL[9]). Certains langages de description de matériel permettent la combinaison de la description algorithmique avec des éléments de structure. Cette combinaison rend les étapes de synthèse plus simples.

Parmi les langages de description de matériel utilisés dans le monde de la synthèse comportementale, peuvent être cités VHDL[VHD94], Verilog[TM91], HardwareC[KM90] et Silage[Hil85]. HardwareC se base essentiellement sur le langage C étendu avec des structures nécessaires à la description de concept matériels tels que le temps et la synchronisation des événements. Silage

[9]En anglais : HDL : Hardware Description Language.

est un langage applicatif conçu spécialement pour les applications à base de flux de données telles que les applications pour le traitement du signal. La standardisation des langages VHDL et Verilog ainsi que les supports commerciaux intensifs en ont fait les langages les plus répandus pour la description du matériel. Ils supportent aussi bien les structures séquentielles que celles concurrentes et permettent de décrire des circuits à des niveaux d'abstraction différents.

La complexité de ces langages peut engendrer des problèmes sérieux pour les outils de synthèse comportementale. La capacité des expressions et des structures des langages dépasse largement les limites des algorithmes de synthèse. Il est donc nécessaire que les outils imposent des restrictions sur les modèles d'entrée. Ces restrictions varient selon les représentations et les algorithmes internes des outils. La restriction principale est la limitation du modèle d'entrée aux structures strictement synchrones. Pour les outils de synthèse comportementale, le temps est divisé en étapes de contrôle qui correspondent aux états d'une machine d'états finis dans l'architecture résultante. Le chapitre 4 de cette thèse présente de façon détaillée les structures comportementales du langage VHDL et leurs interprétations possibles en vue de la synthèse comportementale.

2.3.2 Compilation et élaboration de la description comportementale

La première étape de la synthèse comportementale consiste généralement à compiler la description décrite dans un langage de description de matériel en une représentation interne. Ces représentations internes peuvent être un des graphes discutés dans la section 2.2. Ces représentations serviront par la suite de modèle d'entrée à l'étape d'ordonnancement. Pendant la construction de ce graphe ainsi que la compilation de la description comportementale, un ensemble de pré-optimisations peut être effectué, appelé souvent élaboration. Ces optimisations, connues d'ailleurs du monde de la compilation des langages de programmation [ASU86, Sto94], sont l'élimination du code mort, la propagation des constantes, la simplification des expressions, la pré-évaluation des conditions, la mise à plat des sous-programmes, le remplacement des paramètres génériques, etc. Cette étape d'élaboration permet de simplifier dans certains cas la description comportementale ainsi que les algorithmes d'ordonnancement.

2.3.3 Ordonnancement

L'ordonnancement est le partitionnement de la description comportementale en sous-graphes dont l'exécution peut être effectuée en un seul cycle de contrôle. Ces cycles de contrôle correspondent aux transitions de la machine d'états finis résultante qui peuvent comprendre plusieurs opérations s'exécutant en parallèle. Ce parallélisme est aussi conditionné par l'algorithme d'allocation de ressources qui doit être capable d'allouer le nombre nécessaire de ressources pour l'exécution des opérations parallèles. Cette forte interdépendance justifie certaines approches de synthèse où l'allocation et l'ordonnancement ne sont pas considérés comme des tâches séparées. Par exemple, certains outils de synthèse tels que CATHEDRAL [Not91], GAUT [MSDP93] exécutent l'allocation avant l'étape d'ordonnancement. D'autres, comme Behavioral Compiler [Kna96], AMICAL [JDKR97], CALLAS [BK92], EASY [Sto91], HAL [PKG86], HIS [CBH+91] ont choisi l'ordre inverse.

Dans la littérature, l'étape d'ordonnancement est peut-être l'une des plus étudiée dans le flux de la synthèse comportementale. Cela est dû au fait que c'est l'efficacité de cette étape qui a le plus d'influence sur la performance du circuit généré parce que le nombre de cycles de contrôle et le parallélisme des opérations y sont fixés. Le modèle d'entrée comportmenetale, autrement dit le style d'écriture influence l'ordonnancement. Vu la complexité de cette tâche, des centaines d'algorithmes ont été publiés bien que la plupart d'entre eux puissent être ramenés aux quelques algorithmes de base. Une étude approfondie se trouve dans [JDKR97, Rah97] qui donnent un classement général sur les techniques d'ordonnancement pour la synthèse comportementale.

2.3.3.1 Mode d'ordonnancement

Quel que soit le langage utilisé pour une description comportementale, une classe spéciale des instructions est définie : les instructions d'attente ou autrement dit, les instructions de synchronisation. En VHDL, un WAIT spécifie un état de contrôle de la description comportementale. Pendant la simulation d'un processus, l'exécution du processus est suspendue tant que la condition exprimée dans l'instruction d'attente a une valeur *fausse*. Pour les algorithmes d'ordonnancement, ces instructions introduisent des étapes de contrôle dans le contrôleur. Donc chaque WAIT peut être considéré comme un état implicite imposé par le concepteur. Le code exécuté entre deux WAIT successifs est appelé pas d'exécution[10]. Il en résulte qu'une description comportementale définit une machine d'états finis dont les états correspondent aux instructions d'attente et les transitions sont les pas d'exécution.

[10]En anglais : execution thread.

D'après ces définitions, la tâche principale de l'ordonnancement peut être reformulée :

- Extraire des pas d'exécution de la description comportementale.
- Ordonner ces pas d'exécution sous certaines contraintes imposées par le concepteur en les décomposant dans les étapes de contrôle.

Suivant les résultats de l'ordonnancement appliqués aux pas d'exécution, notamment le nombre de cycles nécessaires à leur exécution, nous avons établi trois catégories d'ordonnancement :

Mode en un cycle (En anglais : Cycle Mode) : Pour ce mode, chaque pas d'exécution est ordonné en un seul cycle de contrôle, les opérations doivent donc être exécutées en un seul cycle d'horloge. Dans ce cas, l'interprétation de la description est proche du modèle utilisé pour la synthèse au niveau transfert de registres. Afin que la spécification comportementale puisse être synthétisable, des restrictions strictes doivent être imposées sur le style d'écriture des pas d'exécution :

- Le pas d'exécution ne peut contenir de boucles infinies ou dépendant de l'exécution de la même boucle.
- Les opérations multi-cycle ne sont pas permises.
- La période de l'horloge doit être assez long pour permettre l'exécution du pas d'exécution le plus complexe du système.

L'avantage principal de ce mode est que le modèle initial est très proche de la description générée au niveau transfert de registres, ce qui assure une synthèse sans changements de comportement externe du modèle. Ce mode d'ordonnancement permet une vérification aisée via un processus de simulation simple avant et après la synthèse comportementale. Ce mode d'ordonnancement est propice aux applications réalisant des protocoles de communication décrites au niveau de cycle d'horloge.

Mode des macro-états (En anglais : Superstate Mode) : [Kna96] Dans ce mode, chaque pas d'exécution est décomposé en plusieurs cycles de contrôle dont le nombre est fixé durant le processus de synthèse. Par rapport à la synthèse au niveau transfert de registres, ce mode est très éloigné puisque les pas d'exécution sont décomposés afin qu'ils puissent être exécutés en plusieurs cycles d'horloge. La seule restriction sur le style d'écriture

est l'interdiction des boucles infinies ou dépendant des données résultant de l'exécution de la même boucle.

Ce mode permet l'exploration des architectures différentes à partir de la même description comportementale avec la modification des contraintes de synthèse. À cause de la décomposition des pas d'exécution en plusieurs cycles d'horloge, le comportement du circuit généré diffère de la spécification initiale, ce qui implique un processus de vérification plus complexe. Pour pouvoir utiliser le même jeu de tests, le temps absolu ne peut être employé pour la génération des stimulus. Des protocoles complexes, par exemple des poignées de main, doivent être utilisés pour les entrées et les sorties du circuit. Aussi ce mode altère-t-il considérablement la correspondance entre le modèle initial et l'architecture résultante.

Mode des états comportementaux : Dans ce mode, chaque pas d'exécution est réalisé par une machine d'états finis dont les transitions peuvent être exécutées en un seul cycle d'horloge. Ainsi le résultat de l'ordonnancement est-il généralement très complexe, l'exécution des transitions étant souvent conditionnelle. N'imposant aucune restriction sur le style d'écriture du modèle initial, ce mode est le mode d'ordonnancement le plus général. La liberté dans la description des algorithmes fonctionnels ainsi que la possibilité de l'exploration d'architectures est plus étendue. Par contre, l'implantation de ce mode nécessite des algorithmes de synthèse beaucoup plus sophistiqués. La correspondance entre les descriptions initiales et celles générées devient encore plus difficile à établir, la vérification de l'architecture au niveau transfert de registres pose des problèmes similaires au mode des macro-états présentés ci-dessus. L'avantage de ce mode se manifeste pour les applications réalisant des algorithmes arithmétiques complexes décrits par des boucles imbriquées où les techniques d'ordonnancement ont une grande liberté de disposer les opérations entre les cycles de contrôle.

La décomposition des pas d'exécution en plusieurs cycles d'horloge implique généralement des changements de comportement des entrées et des sorties du circuit. Ce changement devient encore plus important si les optimisations effectuées durant l'ordonnancement déplacent les opérations au-delà des limites marquées par les instructions de synchronisation pour avoir une plus grande liberté en matière de partage des ressources. Ces transformations nécessitent de fixer les stratégies pour les ordonnancements des opérations manipulant des entrées/sorties. Deux classes de techniques sont connues :

Entrées/sorties fixes : Ce mode préserve l'ordre des opérations des entrées/sorties entre elles et vis-à-vis des instructions de synchronisation. Dans

	Mode d'ordonnancement des pas d'exécutions		
Mode pour les entrées/sorties	**Mode en un cycle**	**Mode des macro-états**	**Mode des états comportementaux**
E/S fixes	**1. En un cycle E/S fixes**	**2. Macros E/S fixes**	**3. Comportementaux E/S fixes**
E/S flottantes	**4. En un cycle E/S flottantes**	**5. Macros E/S flottantes**	**6. Comportementaux E/S flottantes**

Figure 2.5: Modes d'ordonnancement

le mode d'ordonnancement des macro-états ou des états comportementaux pour un pas d'exécution donné, les lectures sont ordonnées dans le premier cycle et les écritures dans le dernier. Pour la réalisation de ce mode, deux solutions s'imposent : l'algorithme d'ordonnancement impose des restrictions supplémentaires sur le style d'écriture ou effectue des mémorisations sur les entrées et les sorties à l'aide de registres destinés à sauvegarder des valeurs intermédiaires. L'analyse de ce type de traitement est présentée dans le chapitre 4.

Entrées/sorties flottantes : Dans ce mode, les opérations manipulant les entrées/sorties sont ordonnées de la même manière que les autres opérations : elles sont ordonnées de façon à ce que le résultat de la synthèse satisfasse les contraintes de dépendances de données tout en minimisant le temps d'exécution. Bien que ce mode permette un espace d'exploration d'architecture plus étendu, le comportement du circuit généré devient souvent extrêmement éloigné de la description comportementale.

La figure 2.5 résume les modes d'ordonnancement ainsi que leur combinaison avec des stratégies de traitement des entrées/sorties. D'une manière générale, nous pouvons conclure que chaque mode a ses avantages et ses inconvénients qui se résument essentiellement dans la liberté du codage des descriptions comportementales et dans la correspondance faible ou forte entre les modèles initiaux et ceux générés. Encore faut-il noter que l'implantation de ces modes dépend de l'algorithme d'ordonnancement. En général, seuls les algorithmes orientés flux de contrôle peuvent réaliser tous les modes présentés. Pour les algorithmes orientés flux de données, le mode des états comportementaux pose des difficultés sérieuses.

2.3.3.2 Algorithmes d'ordonnancement

Les algorithmes d'ordonnancement dans le domaine de synthèse comportementale peuvent être classifiés en deux catégories majeures : les algorithmes orientés flux de données et les algorithmes orientés flux de contrôle. Les premiers consistent à sérialiser une description parallèle alors que les seconds réalisent la parallélisation d'une description séquentielle. Les représentations internes pour ces deux approches s'imposent : en général, un algorithme orienté flux de données utilise des graphes de flux de données alors qu'un algorithme orienté flux de contrôle repose sur un graphe de flux de contrôle.

Algorithmes orientés flux de données. L'ordonnancement d'un graphe de flux de données consiste à affecter chaque opération à une étape de contrôle de manière à ce que toutes les contraintes soient satisfaites. En général, les contraintes fixent aussi bien le nombre d'étapes de contrôle que le nombre de ressources utilisées. À part les opérations multi-cycle, aucune opération ne peut être affectée à plusieurs étapes de contrôle. Les approches orientées flux de données se concentrent sur l'exploitation du parallélisme intrinsèque de la représentation intermédiaire.

Dans ces approches, le problème d'ordonnancement définit, d'une part, la manière dont les opérations sont attribuées aux étapes de contrôle et, d'autre part, le nombre d'étapes de contrôle nécessaires pour exécuter en sa totalité le graphe de flux de données et le nombre d'unités fonctionnelles qui sont impliquées de façon optimale. Le problème d'ordonnancement pour les circuits de données est donc un problème d'optimisation à deux dimensions, définies par les contraintes : le temps et la surface. Suivant les contraintes, nous pouvons distinguer quatre catégories des algorithmes orientés flux de données :

Ordonnancement sans contraintes : L'ordonnancement au plus tôt (ASAP[11]) et l'ordonnancement au plus tard (ALAP[12]) sont des algorithmes d'ordonnancement sans contraintes. Ces algorithmes sont généralement utilisés pour déterminer les intervalles d'exécution, autrement dit la mobilité des opérations.

Ordonnancement sous contraintes de ressources : Les contraintes de ressources imposent généralement des restrictions sur le nombre d'opérations et d'unités fonctionnelles pour limiter la surface du circuit généré.

[11]En anglais : As Soon As Possible.
[12]En anglais : As Late As Possible.

L'approche la plus connue pour résoudre ce problème est l'ordonnancement par liste[13] [Hu61a].

Ordonnancement sous contraintes de temps : La contrainte de temps reflète la limitation du temps d'exécution global du graphe de flux de données, autrement dit le nombre d'étapes de contrôle. L'algorithme le plus répandu est l'ordonnancement orienté par les forces[14] [PK89].

Ordonnancement sous contraintes de temps et de ressources : Cette approche peut être considérée comme la combinaison des deux précédentes et elle donne la problématique la plus complexe à résoudre. Les méthodes de programmation en nombres entiers[15] permettent de décrire et résoudre les trois types d'ordonnancement et garantissent une solution exacte et optimale. Vu que les problèmes de programmations en nombre entiers sont NP-Complets, la résolution de ces formulations est généralement très longue, peu pratique dans le cas des outils de synthèse comportementale.

Algorithmes orientés flux de contrôle. Contrairement à une approche orientée flux de données, une approche orientée flux de contrôle repose sur un graphe de flux de contrôle comme représentation intermédiaire. Un graphe de flux de contrôle est par sa nature une représentation séquentielle de la description d'entrée. Ainsi, un algorithme orienté flux de contrôle doit extraire le maximum de parallélisme et en même temps, satisfaire les contraintes de données. Pour atteindre ces objectifs, ces approches consistent à affecter des séquences d'opérations à des étapes de contrôle, ainsi une opération peut-elle être affectée à plusieurs étapes de contrôle.

Toutes les approches orientées flux de contrôle sont basées sur la recherche des chemins d'exécution composés de séquences d'opérations. Les algorithmes considèrent tous les chemins d'exécution possibles du graphe de flux de contrôle et les ordonnent séparément. Cet ordonnancement est effectué dans le but de minimiser le nombre d'étapes de contrôles pour chaque chemin. Le résultat est un ensemble de chemins ordonnés dont les opérations peuvent être exécutées en un seul cycle de contrôle. Les chemins ordonnés sont regroupés afin de générer la machine d'états finis qui exécute le comportement défini par le graphe de flux de contrôle.

Dans la littérature, le nombre d'algorithmes orientés flux de contrôle est

[13] En anglais : List Scheduling.
[14] En anglais : Force Directed Scheduling.
[15] En anglais : Integer Linear Programming.

inférieur à celui orienté flux de données. L'ordonnancement AFAP[16] est un des algorithmes les plus connus [Cam91]. Le problème principal de cette approche se trouve dans la partie de la recherche des chemins d'exécution. L'algorithme considère en même temps tous les chemins d'exécution du graphe de flux de contrôle et, ensuite, les ordonne. Pour une application complexes composée d'un grand nombre de branchements, l'explosion du nombre de chemins d'exécution devient inévitable. Il en résulte un temps d'ordonnancement très long. En revanche, l'ordonnancement à boucles dynamiques (DLS[17]) [ORJ93] évite ce problème en combinant la recherche des chemins d'exécution avec un ordonnancement à la volée. Une fois qu'une des contraintes est transgressée, la génération du chemin actuel est arrêtée et un nouveau chemin commence. Un algorithme similaire qui résout le problème de l'explosion du nombre de chemins a été publiée dans [BDB94].

2.3.4 Allocation

L'étape d'allocation fixes le nombre de ressources matérielles nécessaire pour la réalisation du circuit décrit par la spécification comportementale. Le nombre de ressources est déterminé en fonction des résultats de l'étape d'ordonnancement. L'allocation détermine le nombre et les types des unités fonctionnelles (additionneurs, multiplieurs, co-processeurs, etc.), les unités pour les stockages de données (registres, blocs de registres, mémoires, etc.) et les unités permettant de réaliser des connexions (multiplexeurs, bus, etc.). Le nombre de ressources peut être restreint par la structure, la complexité du chemin de données, certaines contraintes imposées par le concepteur. Les algorithmes d'allocation comprennent souvent une étape permettant de déterminer les unités fonctionnelles devant exécuter chacune des opération de la description initiale. Cette étape est appelée *l'association des unités fonctionnelles*[18]. Suivant la complexité des unités de la bibliothèque des composants, pour chaque conception, plusieurs solutions peuvent exister. Par exemple, pour l'opération de la multiplication, plusieurs types de multiplieurs peuvent exister avec des des caractéristiques différentes. Le but de l'allocation est donc de décider un ensemble des composants, d'une part qui assure l'exécution de la description comportementale, d'autre part qui peut satisfaire toutes les contraintes imposées par le concepteur.

[16] En anglais : As Fast As Possible.
[17] En anglais : Dynamic Loop Scheduling.
[18] En anglais : Binding.

2.3.5 Génération d'architecture

La génération d'architecture commence par l'allocation de ressources nécessaires pour la communication des unités du chemin de données. Suivant l'architecture cible de l'outil de synthèse, cette étape génère des chemins pour le transfert de données à la base de multiplexeurs ou de bus. Le contrôleur est ensuite produit généralement sous forme d'une machine d'états finis au niveau transfert de registres. Le chemin de données ainsi que le réseaux de communication est finalement générés dans une description structurelle avec l'instantiation des composants de la bibliothèque spécifique. Ces descriptions forment le résultat de la synthèse comportementale et sont synthétisées par les outils de synthèse logique.

2.4 Conclusions

Ce chapitre est une introduction à la synthèse comportementale classique. Comme l'intégration des outils de synthèse comportementale aux flux de conception existants pose généralement des difficultés, un accent a été mis sur le lien de la synthèse comportementale avec la synthèse au niveau de transfert de registres. La problématique de cette intégration ainsi que les limitations des outils de synthèse comportementale classiques sont discutées dans le chapitre suivant.

Les étapes du flux de la synthèse comportementale classique a été également présentées. Ces étapes sont la compilation de la description comportementale, l'ordonnancement, l'allocation, l'association des unités fonctionnelles et la génération de l'architecture. Le sujet principal de cette thèse étant un flux de synthèse basé sur l'ordonnancement, les algorithmes de cette étape ont été présentés de façon plus détaillée. L'existence d'un très grand nombre d'algorithmes d'ordonnancement est justifiée par la diversité des domaines d'application.

Chapitre 3

Synthèse comportementale : état de l'art

Introduction

Un outil de synthèse comportementale est capable de transformer une description algorithmique en une architecture cible composée, dans l'approche traditionnelle, d'un chemin de données et d'un contrôleur. Cette transformation s'effectue par une série de raffinements relativement complexes qui comprennent l'ordonnancement, l'allocation, la génération du chemin de données et la synthèse du contrôleur. Cette approche traditionnelle connue depuis le début les années quatre-vingts soulève plusieurs problèmes sérieux qui ont empêché que la synthèse comportementale soit répandue et acceptée pour les applications industrielles.

La plupart des outils de synthèse essaient d'assurer une solution de synthèse pour le plus large domaine d'application possible. L'architecture cible étant en général plus ou moins fixe, peu configurable par le concepteur, les résultats de la synthèse se sont souvent avérés beaucoup moins satisfaisants que ce que l'on obtient en appliquant la synthèse de transfert de registres. De surcroît, l'efficacité d'une description pour la synthèse est difficile à prédire et la faible correspondance entre la description initiale et l'architecture générée permet difficilement de guider le concepteur sur les modifications à effectuer pour obtenir un meilleur résultat. Le dernier problème lié aux outils de synthèse provient du fait que l'approche utilisée pour leur intégration aux flux de conception existants soit inadéquate et l'architecture générée est généralement peu adaptée à la synthèse au niveau transfert de registres.

Dans ce chapitre, nous essayons d'analyser et d'étudier les causes des problèmes brièvement décrits ci-dessus. Deux outils de synthèse comporte-

mentale classique sont présentés : Behavioral Compiler de Synopsys et AMICAL. Behavioral Compiler est parmi les rares outils à être commercialement disponible. Quant à AMICAL, il est un outil de synthèse universitaire qui a connu un certain succès en matière d'applications industrielles.

3.1 État de l'art

Les outils de synthèse comportementale peuvent être classés en trois générations. La première, prématurément parue, a précédé l'acceptation générale de la synthèse au niveau transfert de registres. La deuxième a tenu en compte de la synthèse au niveau transfert de registres mais l'intégration de ces outils aux flux de conception a été peu efficace car ils n'ont pu profiter de la puissance des outils de synthèse au niveau transfert de registres. Les outils de la dernière génération s'adaptent mieux aux exigences des applications industrielles en reformulant certains principes et approches de la synthèse comportementale et en s'insérant aux flux de conception existants.

3.1.1 La première génération des outils de synthèse comportementale

Les outils de synthèse comportementale de la première génération ont été mis au point principalement par des chercheurs venant de la recherche sur l'architecture des ordinateurs [Bar73, Mar79, Tho81, GKP85, OG86a]. Le bénéfice le plus important de ces travaux est la formulation et la définition des tâches et des étapes de la méthodologie pour la synthèse comportementale. Ces outils comprenaient déjà des étapes d'ordonnancement, d'allocation, de l'association des unités fonctionnelles[1], de partitionnement et de génération de l'interface. Ces outils peuvent être considérés comme des outils à usage général visant le domaine d'application le plus large possible. À cette époque, la synthèse au niveau transfert de registres n'étant qu'à ces débuts, les outils de synthèse comportementale réalisaient aussi les étapes de la synthèse logique et celles de la réalisation physique [Sou83].

3.1.2 La deuxième génération des outils de synthèse comportementale

Contrairement aux outils présentés ci-dessus, ceux de la deuxième génération sont caractérisés par des domaines d'application restreints et spécialisés

[1] En anglais : Binding.

ainsi que par la génération d'une architecture composée d'un contrôleur et d'un chemin de données. Le flux de synthèse est réduit à un nombre plus limité d'étapes telles que l'ordonnancement, l'allocation et l'association des unités fonctionnelles ; il est lié aux outils de synthèse de transfert de registres. Plus d'une centaine d'outils faisant partie de cette génération ont été publiés et visent une large variété de domaines d'application : DSP, contrôleurs, circuits de communication, etc. [WC91]. L'interdépendance entre les étapes de synthèse étant forte, toutes les possibilités de l'ordre de leur applications ont été étudiées. Dans certaines approches, la problématique de la synthèse comportementale a été considérée comme un problème d'optimisation [LMD94, GE92]. Pour la plupart des outils de deuxième génération, le problème principal était la difficulté de leur utilisation au sein des flux de conception existants. Les recherches n'était focalisées que sur l'optimisation des ces trois étapes. En général, ces outils ont été évalués par un ensemble de *benchmarks* académiques et non pas par des applications complexes et industrielles (*"the fifth-order elliptical filter syndrome"* [PR94]). Le lien étant faible entre ces outils et la synthèse au niveau transfert de registres devenue de plus en plus puissante, ces outils n'ont pu s'adapter aux nouvelles capacités des outils de synthèse au niveau transfert de registres [Sto94].

En général, la décomposition de l'architecture générée par la synthèse en contrôleur et en chemin de données détaillé empêche d'exploiter la capacité de la synthèse au niveau transfert de registres et celle de l'optimisation logique. Par exemple, si cette décomposition fixe l'allocation de certaines opérations du chemin de données qui peut d'ailleurs être effectué au niveau transfert de registres, cela rend difficile l'exploitation des optimisations dépendant de la technologie qui sont assurées par la synthèse au niveau transfert de registres. De surcroît, nous avons observé qu'une conception dont la structure contenait trop de niveaux de hiérarchie pouvait réduire considérablement l'efficacité de la synthèse. Ce résultat peut provenir du fait que la synthèse au niveau transfert de registres ne soit pas capable d'effectuer les optimisations à travers les interfaces des unités. D'autre part, si cette synthèse s'effectue après la mise à plat complète de la conception, le modèle sera à un niveau trop bas, ne permettant plus d'optimisations de haut niveau.

De nos jours, certains outils ont essayé d'améliorer l'efficacité de cette approche traditionnelle en assurant un ordre d'exécution des étapes flexible, souvent plus adapté à un certain domaine d'application [PR94, GIC+96, GPR98, KP91, KCG+98]. Ces approches permettent d'obtenir des résultats plus efficaces pour les applications réelles. Cependant, l'architecture générée reste peu adaptée à la synthèse au niveau transfert de registres.

3.1.3 La troisième génération des outils de synthèse comportementale

La nouvelle génération des outils de synthèse essaie de viser un domaine d'application plus étendu et d'avoir une interface plus élaborée avec la synthèse au niveau transfert de registres. Cette troisième génération des outils essaie d'exploiter entièrement des capacités de la synthèse au niveau transfert de registres dont l'évolution a été accélérée par son utilisation intensive en milieu industriel. Cette nouvelle génération des outils de synthèse est marquée par les modifications appliquées sur le flux de conception des outils traditionnels. En 1994, un outil de synthèse nommé *VOTAN* a été publié [WBD+94] ; il a introduit une nouvelle approche d'ordonnancement considérant la synthèse comme une problématique de transformation de code au niveau comportemental suivi de la synthèse au niveau transfert de registres. Cette nouvelle approche peut être considérée comme le début des outils de troisième génération. Néanmoins, elle a été abandonnée avant même d'avoir été validée par des applications complexes et réelles. Dans notre outil de synthèse, nous avons suivi le même principe mais avec un algorithme d'ordonnancement plus puissant, permettant de générer du code au niveau transfert de registres plus efficace, avec des résultats plus performants. *Hiasynth* [Ber99] représente une approche radicalement différente par rapport aux outils traditionnels en intégrant la tâche de la synthèse comportementale et celle au niveau transfert de registres dans le même flux de conception. Au sein de cet outil, la synthèse au niveau transfert de registres est utilisée pour effectuer les tâches de la synthèse comportementale basée sur un nouveau modèle de conception nommé *graphe de réseau comportemental* (*Behavioral Network Graph*). L'outil *RapidPath* [TLW+90] accepte comme entrée une description comportementale avec exactitude du cycle. Le style d'écriture de ces descriptions peut être considéré comme une extension du sous-ensemble pour la synthèse au niveau transfert de registres. Il ne restreint pas l'emplacement des attentes au changement de l'horloge. Cette approche permet à l'utilisateur d'écrire plus facilement des machines d'états finis complexes sous un format plus lisible et plus facile à comprendre. Les algorithmes de synthèse étant optimisés et orientés vers les conceptions dominées par les traitements de données, le domaine d'application reste relativement restreint.

3.2 Limitations des outils de synthèse traditionnels

Dans la littérature, une multitude d'algorithmes de synthèse comportementale ont été publiés. La plupart de ces recherches ne s'intéressaient qu'à une étape de synthèse ou n'apportaient que des améliorations à des algorithmes existants. Parmi eux, il y en a très peu qui offrent un outil complet, et encore moins qui sont parvenus à atteindre le niveau de commercialisation. Pour présenter les caractéristiques et les principales limitations des outils existants, nous avons opté pour deux outils : Behavioral Compiler qui est peut-être l'outil commercial le plus connu dans le monde de la synthèse, et AMICAL, un outil universitaire qui a d'ailleurs servi de base aux travaux de cette thèse.

3.2.1 Behavioral Compiler[Kna96]

Parmi les rares outils commerciaux existants, Behavioral Compiler de Synopsys est le plus connu et le plus discuté par les concepteurs. En dépit de certaines applications qui sont plutôt des tests de l'outil, cet outil de synthèse n'est jamais parvenu à occuper une place importante pour la conception des circuits industriels et à connaître un grand succès. L'objectif du lancement de l'outil par Synopsys était d'accroître la productivité des concepteurs en fournissant un outil de synthèse comportementale qui s'appuie sur un outil de synthèse au niveau transfert de registres : Design Compiler. Une spécificité mise en relief par Synopsys est la possibilité de l'exploration de l'architecture rapide à partir d'une description comportementale avec un ensemble de contraintes de haut niveau. Le modèle d'entrée peut être décrit en VHDL aussi bien qu'en Verilog. L'architecture générée ressemble fort aux architectures spécifiques aux outils de deuxième génération ; elle est composée d'un contrôleur réalisé par une machine d'états finis, d'un chemin de données et de mémoires multi-ports.

Le domaine d'application de Behavioral Compiler est essentiellement celui des circuits orientés flux de données, dominés par les calculs arithmétiques avec des opérations complexes comme des conceptions de traitement du signal et d'image. L'algorithme d'ordonnancement est également optimisé pour le traitement des mémoires. Les mémoires spécifiques à de diverses technologies sont associées automatiquement ou à travers des contraintes imposées par le concepteur, et leurs accès sont en conséquence optimisés. En matière de contrôle, Behavioral Compiler n'est pas considéré comme la solution optimale. Pour ces applications, Synopsys a lancé un autre produit, le Protocol

Compiler [Syn98a]. Cet outil a connu encore moins de succès que Behavioral Compiler. La décomposition de systèmes complexes en plusieurs parties suivant les outils de synthèse peut poser des problèmes et d'ailleurs, n'est toujours pas évidente à effectuer. Ainsi les concepteurs doivent-ils se contenter des limitations imposées par un de ces outils de synthèse.

Behavioral Compiler est intégré étroitement au flux de conception de l'ensemble des outils de Synopsys :

- VHDL/Verilog Analyser permet de compiler les descriptions comportementales dans une base de données utilisée par les outils de synthèse de Synopsys.
- Design Compiler [Syn98b] comme outil de synthèse au niveau transfert de registres permet d'optimiser et de réaliser à l'aide d'une bibliothèque technologique l'architecture générée par Behavioral Compiler. Cette bibliothèque technologique est la même utilisée par Behavioral Compiler qui devrait assurer une estimation de performances précise au niveau comportemental pour l'étape de l'exploration d'architecture.
- DesignWare permet aux concepteurs de construire leur propres bibliothèques de composants. Ces composants réalisent généralement des opérations plus complexes que celles qui sont fournies par les bibliothèques de base de Synopsys. Ces opérations sont ordonnées, allouées et insérées dans l'architecture résultante durant le processus de synthèse.
- BCView permet de visualiser les résultats de la synthèse comportementale à travers une interface graphique. À l'aide de cet outil, nous pouvons analyser les états du contrôleur, l'allocation des opérateurs dans le chemin de données. Se basant sur cette analyse, les contraintes initiales peuvent être modifiées pour l'exploration d'autres architectures alternatives avant l'optimisation au niveau des portes.
- Un simulateur VHDL et Verilog sont fournis qui permettent la validation des descriptions comportementales ainsi que les résultats de chaque étape de synthèse.

Le flux de conception de Behavioral Compiler est illustré par la figure 3.1. Behavioral Compiler synthétise le chemin de données, les mémoires et le contrôleur pour une description comportementale. Cette synthèse est effectuée sous les contraintes imposées par le concepteur dans la première phase de synthèse. Les contraintes influencent les activités des entrées/sorties, les types de

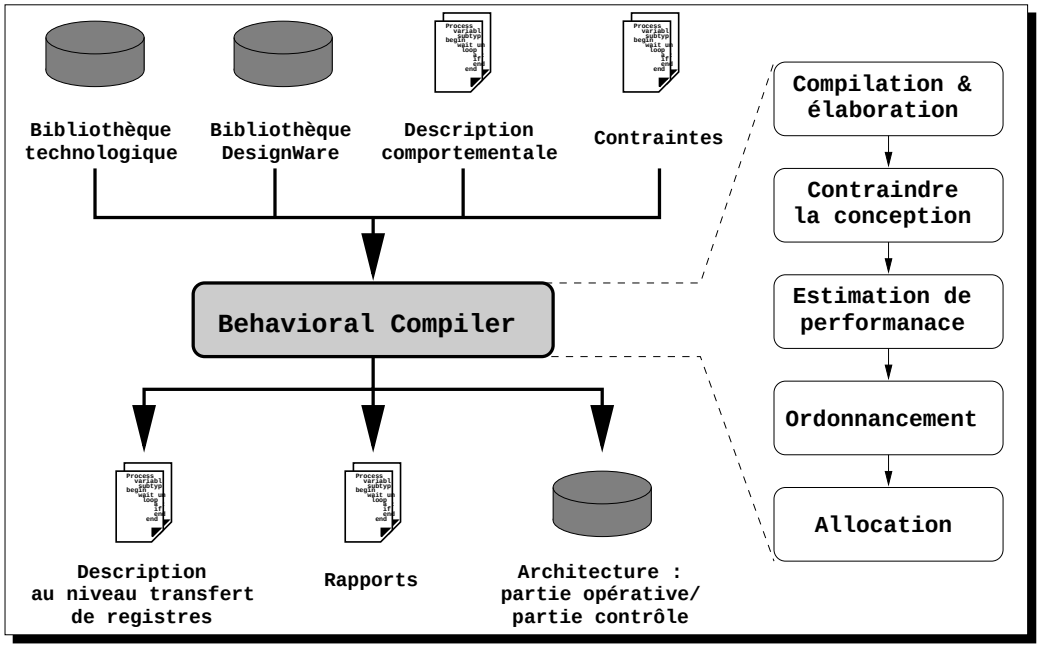


Figure 3.1: Flux de synthèse de Behavioral Compiler

ressources, le nombre de cycles de contrôle, les traitements appliqués sur les boucles, etc. Une fois la description comportementale compilée et élaborée, Behavioral Compiler utilise la bibliothèque technologique et les composants réutilisables de DesignWare pour la détermination des informations concernant les délais des opérations du chemin de données, de la machine d'états finis du contrôleur, des registres et du réseau de multiplexeurs. Ensuite, Behavioral Compiler exécute l'ordonnancement, l'allocation des ressources et génère la machine d'états finis pour le contrôleur. À l'aide de BCView, plusieurs architectures alternatives peuvent être générées. Après avoir opté pour celle qui convient le plus, le concepteur peut effectuer la simulation au niveau transfert de registres et, si le résultat est satisfaisant, la synthèse logique.

3.2.1.1 Mode d'ordonnancement de Behavioral Compiler

Behavioral Compiler offre trois principaux modes d'ordonnancement aux concepteurs. Ces modes d'ordonnancement se différencient selon les changements autorisés des comportements des entrées et sorties et/ou le nombre de cycles de contrôle insérés entre deux instructions d'attente de la description comportementale. Les modes différents d'ordonnancement sont illustrés par la figure 3.2. En général, ces modes ne conviennent qu'à des domaines

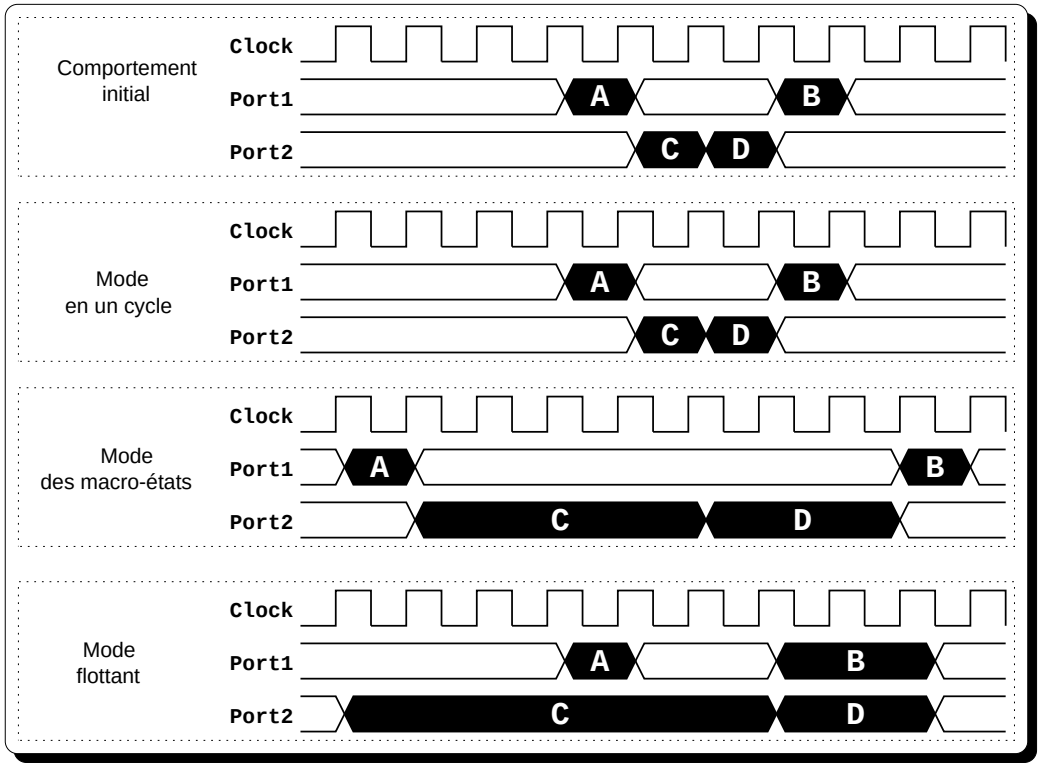


Figure 3.2: Modes d'ordonnancement de Behavioral Compiler

d'application limités :

Mode en un cycle (Cycle-Fixed) : Le comportement externe de la description au niveau transfert de registres correspond exactement à la description comportementale en ce qui concerne le nombre de cycles de l'exécution et des lectures et des écritures des entrées/sorties. Le style d'écriture est plus restreint que dans le cas des autres modes d'ordonnancement. Ce mode convient aux applications réalisant des protocoles de communication avec exactitude du cycle. Pour les calculs complexes, les cycles de contrôle insérés par le concepteur empêchent tous types d'optimisation entre les opérations arithmétiques.

Mode des macro-états (Superstate-Fixed) : Dans ce mode, les instructions d'attente désignent des états spéciaux dans le contrôleur qui sont nommés macro-états[2]. Les opérations situées entre deux macro-états peuvent être décomposées par l'ordonnancement en plusieurs cycles de con-

[2]En anglais : superstate.

	Mode d'ordonnancement des pas d'exécutions		
Mode pour les entrées/sorties	**Mode en un cycle**	**Mode des macro-états**	**Mode des états comportementaux**
E/S fixes	**1. En un cycle E/S fixes**	**2. Macros E/S fixes**	**3. Comportementaux E/S fixes**
E/S flottantes	**4. En un cycle E/S flottantes**	**5. Macros E/S flottantes**	**6. Comportementaux E/S flottantes**

Figure 3.3: Modes d'ordonnancement

trôle. L'ajout des cycles implique le changement du comportement des entrée/sorties. Les transitions des ces signaux peuvent être décalées dans le temps, mais leur ordre reste toujours inchangé. Sur notre exemple, *Port2* change de la valeur **C** à **D** avec quatre cycles de retard par rapport à la description comportementale.

Mode avec des entrées/sorties flottantes (Free-Floating) : Ce mode ressemble au mode des macro-états avec la différence que l'ordre des transitions des entrées/sorties peuvent être changées mais les dépendances de données sont tout de même sauvegardées. En général, les concepteurs évitent l'utilisation de ce mode d'ordonnancement car le comportement du circuit produit par le processus de synthèse devient très éloigné par rapport à celui de la description initiale.

La figure 3.3 illustre les modes d'ordonnancement de Behavioral Compiler à l'aide du modèle présenté dans la section 2.3.3.1. Les cases grises de ce tableau désignent les modes réalisés par Behavioral Compiler. À cause des limitations sur le style d'écriture du code comportemental imposées par les algorithmes d'ordonnancement et la représentation intermédiaire, les modes d'états comportementaux ne peuvent être acceptés.

3.2.1.2 Limitations de Behavioral Compiler

Plusieurs sociétés ont tenté d'introduire Behavioral Compiler pour la conceptions des circuits intégrés [Bla97, Ves98, JKC98]. Chez Compaq Computer, une équipe a conçu un circuit réalisant le standard *Virtual Interface Architecture* qui établit l'interface entre le bus PCI et le bus ServerNet. Dans [SB99], les auteurs expliquent la méthode employée pour cette application avec l'utilisation de Behavioral Compiler aussi bien que tous les problèmes rencontrés pendant le projet.

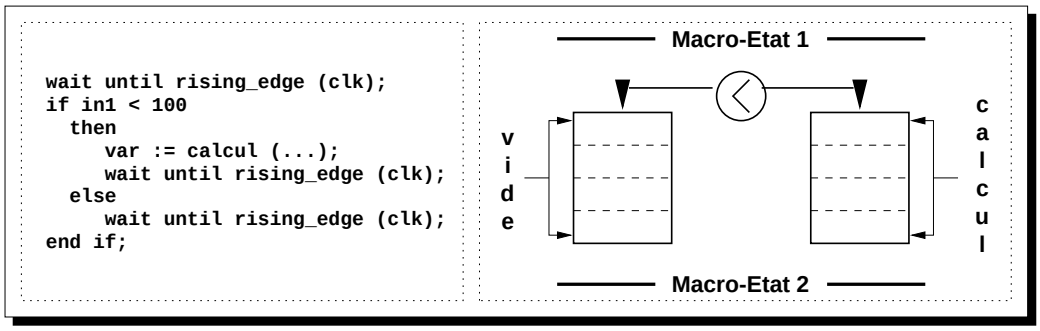


Figure 3.4: Problèmes avec les branchements conditionnels

Modes d'ordonnancement : Les trois principaux modes d'ordonnancement offerts par Behavioral Compiler conviennent à des domaines d'application différents. Une restriction importante est que le choix du mode d'ordonnancement est toujours limité à un seul processus de la conception. Ainsi la combinaison des calculs arithmétiques complexes et des protocoles de communication avec exactitude du cycle n'est-elle pas possible. Dans la plupart des cas, l'ordre du traitement des données étant la lecture des entrées, le traitement lui-même et finalement, l'écriture des résultats, cette limitation demande généralement aux concepteurs une décomposition supplémentaire de l'algorithme principal. Cette sur-décomposition justifiée uniquement par les défauts de l'algorithme d'ordonnancement de Behavioral Compiler engendre des communications additionnelles entre les processus ou les modules, ce qui augmente la complexité de la description ainsi que le temps nécessaire pour sa validation et diminue l'efficacité de l'architecture résultante.

Style d'écriture restreint : Behavioral Compiler impose un style d'écriture très restreint, ce qui est dû à la représentation interne et à l'algorithme d'ordonnancement. Ces restrictions sont fastidieuses pour les concepteurs habitués au niveau transfert de registres parce que, sans une connaissance approfondie des techniques de la synthèse comportementale, elles ne paraissent guère logiques et sont donc difficilement adaptables. Les limitations concernent essentiellement la combinaison des instructions d'attente et les structures conditionnelles. Par exemple, si une instruction d'attente, autrement dit, un macro-état se trouve dans une branche de IF, il faut qu'il y ait au moins un autre macro-état dans l'autre branche. Cette restriction est due à l'équilibrage des chemins entre deux macro-états. L'exemple est illustré par la figure 3.4. Si le mode d'ordonnancement est le mode des macro-états, les opérations entre deux macro-états peut être décomposées entre plusieurs cycles de contrôle. Par contre, le nombre de cycles est toujours fixe pour chaque

branche partant du même macro-état. La conséquence de ce fait peut être surprenante : admettons que la branche *then* contenant des calculs complexes prenne quatre cycles de contrôle. La branche *else* avec un macro-état, insérée d'ailleurs en raison de l'équilibrage des chemins, mais sans opérations, prendra également quatre cycles sans effets particuliers sauf le retard de l'exécution. Pour contourner ce type de problèmes, un macro-état doit être placé comme première instruction dans la branche *then*. Évidemment, de cette solution résulte la perte d'un cycle pour cette branche mais elle évite les quatre cycles superflus.

Comme la représentation interne de Behavioral Compiler se base sur un graphe de flux de données et de contrôle, l'ordonnancement des boucles impose un style d'écriture plus restreint. L'ordonnancement doit traiter ces boucles comme des blocs séparés. Ainsi, quel que soit le mode d'ordonnancement choisi, des instructions d'attente sont exigées avant et après chaque boucle. Cela empêche le déroulement de la première et de la dernière itération pendant l'exécution et résulte donc des pertes de cycles de contrôle. Cette restriction peut prendre une plus grande dimension, si les boucles sont imbriquées dans la description.

```
wait until rising_edge (clk);
while var1 < 100 loop
   ...
   wait until rising_edge (clk);
   ...
end loop;
wait until rising_edge (clk);
...
```

Figure 3.5: Restrictions spécifiques aux boucles

Dans les boucles imbriquées, il est courant de traiter les erreurs survenues pendant le traitement de données par les instructions EXIT qui permettent de quitter les boucles même si elles se situent en profondeur dans la hiérarchie et de sauter directement à une partie du code capable de traiter l'événement. Le style d'écriture imposé par Behavioral Compiler défend strictement ce cas de figure. Pour contourner le problème, un drapeau doit être introduit dont la valeur est vérifiée à la fin de chaque itération. Par contre, cette technique augmente le nombre de registres, rend le code moins lisible et accroît considérablement le temps nécessaire pour l'ordonnancement.

Restrictions spécifiques aux sous-programmes : Le style d'écriture est extrêmement restreint au sein des sous-programmes. Il ne se limite qu'à la description des opérations combinatoires. Ni instructions conditionnelles, ni boucles, ni instructions d'attente ne peuvent y être placées. Cette restriction pose des problèmes pour les descriptions des protocoles de communication où certains accès à une ressource (mémoires, périphériques, etc.) qui peuvent être décrits facilement à l'aide des procédures ou des fonctions puisque ces

fragments de code se répètent tout au long de l'algorithme. Le concepteur est donc obligé de répéter ce code dans le processus et chaque modification dans le protocole d'accès entraîne une chaîne de modifications.

Estimation de performances imprécise : La liaison étant étroite entre Behavioral Compiler et Design Compiler (bibliothèques technologiques communes et mêmes bases de données), on aurait pu s'attendre à une estimation de performances précise pour pouvoir réellement choisir l'architecture la plus adéquate. En réalité, la précision de l'estimation de performance n'atteint approximativement que 50 %. L'inexactitude en matière du nombre de registres est encore moins compréhensible.

Temps d'exécution de la synthèse : Une caractéristique importante de Behavioral Compiler, soulignée d'ailleurs même par Synopsys, est la génération et l'exploration des architectures alternatives à partir de la même description comportementale avec des contraintes différentes. Cette exploration exigerait un temps de synthèse relativement court. En réalité, la synthèse des descriptions complexes dure plusieurs heures, ce qui limite fortement le nombre d'architectures que le concepteur peut analyser.

3.2.2 Outil de synthèse universitaire : Amical[JDKR97]

Le deuxième outil de synthèse que nous avons choisi de présenter et qui a d'ailleurs servi de base pour les travaux de cette thèse a été développé en milieu universitaire. Bien que certains algorithmes internes du système soient uniques dans le monde de la synthèse comportementale, l'approche générale conceptuelle suit entièrement les étapes conventionnelles des outils de synthèse de deuxième génération. Le flux de conception est conçu pour la synthèse des descriptions écrites en un sous-ensemble comportemental du langage VHDL et optimisé surtout pour les applications complexes dominées par les structures de contrôle. AMICAL fournit des méthodes pour l'abstraction des modules complexes et existants pour leur réutilisation comme des boites noires dans les descriptions comportementales.

Le flux de conception d'AMICAL commence par la compilation de la description comportementale en un format intermédiaire sur lequel s'applique la première phase de l'ordonnancement. Cet étape construit une machine d'états finis dites comportementale qui est la base du contrôleur de l'architecture finale. Les opérations sont ensuite allouées et associées aux unités fonctionnelles et aux co-processeurs. Un deuxième ordonnancement, appelé micro-ordonnancement, génère la machine d'états finis au niveau transfert

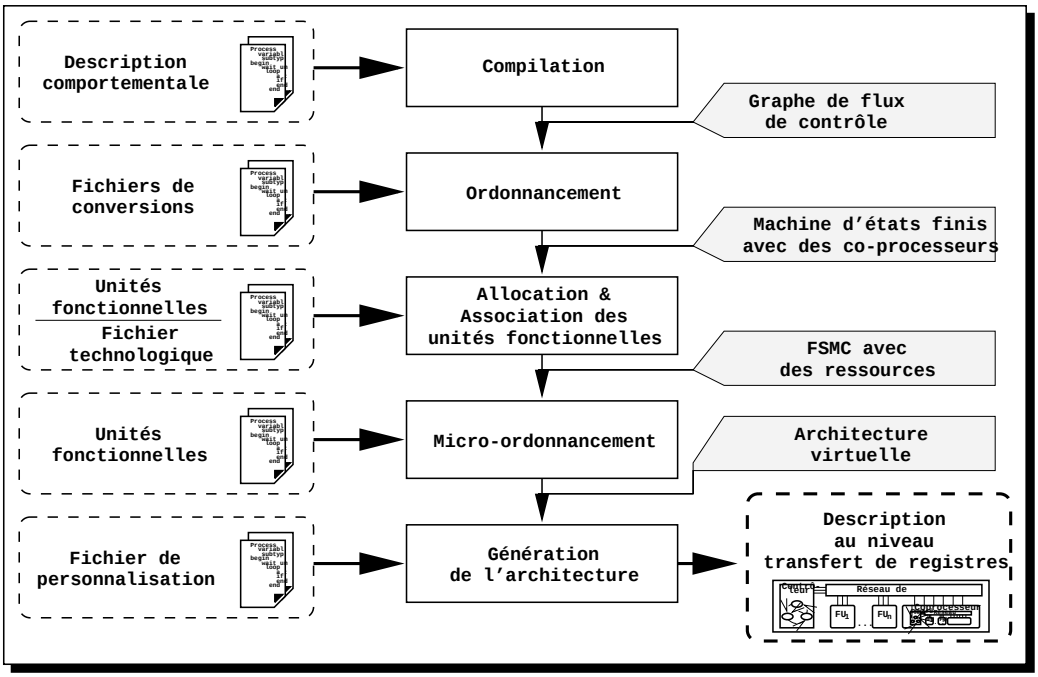


Figure 3.6: Flux de synthèse d'AMICAL

de registres. La dernière étape du flux constitue à allouer les connexions entre le contrôleur et les unités fonctionnelles et à générer les descriptions finales au niveau transfert de registres. Chaque étape de ce flux de synthèse peut être exécutée automatiquement, ou bien entièrement ou partiellement manuellement.

Limitations :

- La principale limitation provient de la complexité extrême du flux de conception d'AMICAL. Hormis la description comportementale, le système nécessite beaucoup d'informations supplémentaires pratiquement pour chaque étape de synthèse. Ces informations, placées dans plusieurs fichiers de configuration dont la cohérence ne dépend que de l'utilisateur, sont souvent redondantes, ce qui peut être une source importante d'erreurs dans l'architecture résultante. Aussi, cette complexité empêche-t-elle l'intégration de l'outil avec des outils de synthèse au niveau système. Le changement de niveau d'abstraction étant beaucoup plus considérable, le concepteur ne peut fournir ces informations supplémentaires au système qui peuvent être d'ailleurs souvent générées automatiquement.

- L'approche interactive de la synthèse a été conçue plutôt du point de

vue des développeurs que de celui des concepteurs. Son utilisation est compliquée et demande une connaissance complète des algorithmes de synthèse comportementale. Dans la vie pratique, des concepteurs ne s'en sont jamais servis. Quoique ce fait ne puisse être considéré comme une limitation puisque l'interactivité est toujours optionnelle, la plupart des utilisateurs se sentaient plutôt embarrassés par les multitudes de représentations internes et par les possibilités inutilisées.

3.2.2.1 Modèle d'entrée comportemental

Le modèle d'entrée de l'outil de synthèse AMICAL est composé de deux informations principales : une description comportementale écrite en VHDL et une bibliothèque de composants capables d'exécuter les opérations de la description algorithmique.

Description algorithmique comportementale : La synthèse accepte une description comportementale écrite en un sous-ensemble du langage VHDL. La synthèse est restreinte à un seul processus qui constitue la principale unité du flux de conception. Le sous-ensemble du VHDL accepté comprend la majorité des actions séquentielles permettant de décrire des algorithmes complexes. Toutes les actions conditionnelles et itératives sont supportées ainsi que les affectations de signaux et de variables et les instructions d'attente. Grâce à l'algorithme d'ordonnancement d'AMICAL, la combinaison de ces instructions n'est pratiquement pas limitée, ce qui assure la facilité de décrire des protocoles de communication complexes contenant souvent la combinaison des structures de contrôle imbriquées et des points de synchronisation.

Bibliothèque des composants : La bibliothèque de composants contient toutes les informations nécessaires sur les unités fonctionnelles et sur les co-processeurs du point de vue de la réalisation du circuit. AMICAL fournit une bibliothèque de base pour les composants élémentaires tels que les registres, les multiplexeurs, les opérations de base, etc. Cette bibliothèque peut être étendue par l'utilisateur pour la réalisation des opérations complexes exécutées par des co-processeurs. Ces co-processeurs peuvent être des composants déjà existants ou le résultat d'une synthèse antérieure. Dans la description comportementale, les services assurés par un co-processeur sont accessibles à l'aide d'appels de sous-programmes.

Chaque unité fonctionnelle est décrite à des niveaux d'abstraction différents :

Vue comportementale : La vue comportementale d'une opération est décrite à l'aide de sous-programmes VHDL qui ne servent qu'à la simulation de la description avant le processus de synthèse.

Vue structurelle : La vue structurelle située dans la bibliothèque des composants représente l'interface des unités fonctionnelles, le protocole de communication qui donne accès aux opérations à travers l'interface et certaines informations supplémentaires permettant une estimation de performances sur la surface du circuit, sur le temps d'exécution des opérations et sur la consommation du module.

Vue de réalisation : La vue de réalisation doit être une description au niveau transfert de registres qui est insérée dans l'architecture générée. En général, cette description est décrite directement au niveau transfert de registres pour les opérations de base telles que les additions, les multiplications, etc. Les co-processeurs sont souvent générés également par AMICAL avec la méthodologie de la synthèse comportementale.

Limitations :

- La principale limitation du modèle d'entrée provient de la complexité des descriptions des unités fonctionnelles et des co-processeurs. La représentation de trois vues pose le plus de problèmes. La cohérence des ces vues est entièrement sous la responsabilité du concepteur. La maintenance des ces vues est difficile puisque chaque modification dans une de ces vues doit entraîner la modification des autres vues. La vue comportementale écrite uniquement pour la simulation est souvent éloignée de la vue de réalisation, ce qui peut impliquer de grandes différences au niveau de la simulation. La génération automatique de ces vues étant difficile, voire impossible dans la plupart des cas, l'intégration de cet outil de synthèse à un flux partant d'une description au niveau système est très problématique.

- Le sous-ensemble VHDL est restreint aux descriptions entièrement algorithmiques. Certaines structures ressemblant plutôt au style d'écriture au niveau transfert de registres sont exclues de ce sous-ensemble telles que les manipulations des bits complexes, les traitements des champs de bits, etc. Ces structures étant essentielles pour les protocoles de communication utilisant des trames, des entêtes complexes, ces restrictions nécessitent que ces opérations élémentaires soient réalisées par des unités fonctionnelles. L'algorithme d'ordonnancement étant incapable de chaîner ces unités fonctionnelles, le résultat de la synthèse est

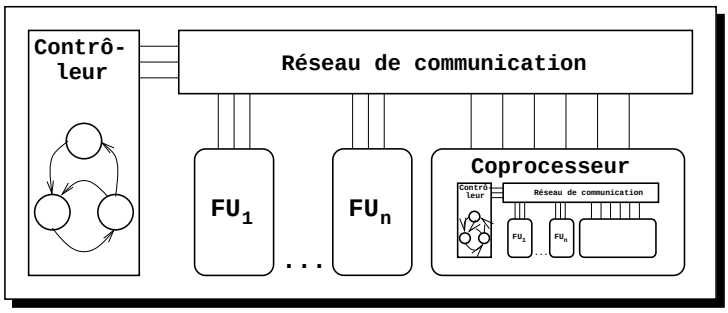


Figure 3.7: Machine d'états finis avec des co-processeurs

inefficace puisque, par exemple, chaque sélection d'un champ de bits provoque l'insertion d'un cycle d'horloge de plus alors que cet opération n'est qu'un transfert d'un registre à un autre.

3.2.2.2 Architecture cible au niveau transfert de registres

L'architecture cible de l'outil de synthèse AMICAL est basé sur le modèle de *machine d'états finis avec des co-processeurs* (FMSC[3]). Ce modèle d'architecture a été conçu pour les conceptions modulaires et la réutilisation des composants existants, même ceux qui ont été réalisés à l'aide d'un outil de synthèse comportementale. L'architecture, illustrée par la figure 3.7 est composée d'un contrôleur et d'un chemin de données qui est constitué d'un ensemble d'unités fonctionnelles et de co-processeurs et d'un réseau de communication.

Contrôleur : Le contrôleur de l'architecture commande les opérations exécutées par les unités fonctionnelles et les co-processeurs ainsi que le réseau de communication. Ce contrôleur est généré automatiquement durant le processus de synthèse. Il est représenté par une machine d'états finis au niveau transfert de registres selon le modèle de Mealy.

Unités fonctionnelles : Les unités fonctionnelles peuvent s'exécuter en parallèle. Ce parallélisme est fixé durant le processus de synthèse. Les unités fonctionnelles communiquent entre elles et avec l'environnement externe à travers le réseau de communication, sont commandées par le contrôleur. Du point de vue du contrôle, elles sont traitées comme des co-processeurs et utilisées pour l'exécution des opérations de la description comportementale. La complexité des ces unités fonctionnelles peut

[3]FMSC : Finite State Machine with Coprocessors.

varier : elles peuvent exécuter de simples opérations arithmétiques, des blocs de mémoires ou des unités d'entrées/sorties complexes.

Réseau de communication : Le réseau de communication établit les connexions entre les unités fonctionnelles et l'environnement externe et il est basé soit sur un système de bus soit sur des multiplexeurs.

Limitations :

- Bien que ce modèle d'architecture ait été conçu pour être plus configurable et plus efficace que les modèles traditionnels comme celui de FSMD[4] [GDWL92], il souffre tout de même de toutes les limitations qui caractérisent les outils de synthèse de deuxième génération. Après la synthèse, le circuit est sur-décomposé, l'allocation est fixée, les optimisations au niveau des portes ne sont plus possibles entre les unités fonctionnelles, il est difficile de trouver la correspondance entre la description initiale et l'architecture résultante, etc.

- L'exécution de chaque unité fonctionnelle dans le chemin de données est fixé en un ou plusieurs cycles d'horloge. L'architecture ne permet pas d'utiliser de structures complexes telles que les opérations chaînées, la connexion entre un champ des blocs de registres et une unité fonctionnelle, etc. Cette restriction demande plusieurs cycles de contrôle pour l'exécution des opérations entre lesquelles il existe des dépendances de données. Cela implique des résultats souvent inefficaces et des changements considérables dans le comportement pour les descriptions avant et après le processus de synthèse.

- Le modèle d'architecture aussi bien que le processus de synthèse sont peu adaptés pour les éléments structurels des systèmes. Dans la description en VHDL, l'utilisation des composants et des affectations concurantes n'est pas supportée, la combinaison des modules synthétisables et non-synthétisables est difficile. Bien qu'AMICAL assure une solution pour cette dernière à l'aide d'un fichier de personnalisation, son utilisation reste extrêmement compliquée et, souvent, donne des résultats inefficaces.

3.2.2.3 Utilisation du système AMICAL

Malgré les critiques et les limitations formulées ci-dessus, AMICAL reste l'un des outils les plus avancés parmi les outils de synthèse comportementale de deuxième génération. Il a été utilisé avec succès pour la réalisation

[4]FSMD : Finite State Machine with Data path.

de plusieurs applications. L'une de ces applications est un estimateur de mouvement d'un CODEC H261 [Ber97]. Ce circuit contient trois modules complexes orientés flux de contrôle, deux blocs de mémoires avec des protocoles d'accès et un co-processeur réalisé par l'outil de synthèse Cathedral [Not91]. La confrontation avec des concepteurs industriels lors de la réalisation de cette application a permis de définir la nouvelle génération des outils de synthèse comportementale.

3.3 Nouveaux objectifs pour la synthèse comportementale

Afin que la synthèse comportementale puisse être acceptée par les concepteurs en milieu industriel, la redéfinition des tâches et du flux de conception est absolument nécessaire et primordiale. Les travaux de cette thèse visent donc à établir un nouveau flux de conception pour la synthèse comportementale qui est fortement lié à la synthèse au niveau transfert de registres pour en exploiter toute la puissance. Pour atteindre ce but, les principaux objectifs sont les suivants :

- *Meilleure intégration de la synthèse comportementale aux flux de conception existants.*

  La plupart des systèmes complexes composés de modules inter-connectés et hiérarchiques ne peuvent être décrits au niveau d'abstraction comportemental. Ces applications se composent donc de sous-systèmes dont quelques-uns sont réalisés par l'application de la méthodologie de la synthèse comportementale. Certains d'entre eux peuvent être décrits au niveau transfert de registres, d'autres au niveau de portes (par exemple les opérations arithmétiques) ou même au niveau encore plus bas (mémoires, convertisseurs numériques/analogiques). Ces applications nécessitent que les modules générés par un outil de synthèse comportementale soient facilement connectés avec d'autres modules générés en utilisant d'autres techniques, notamment la synthèse au niveau transfert de registres. Cette intégration comprend les descriptions comportementales ainsi que les descriptions au niveau transfert de registres et au niveau des portes logiques. Le modèle initial aussi bien que la description générée doivent être validés au sein du système entier.

- *Flux de conception flexible mais facile à utiliser.*

  Comme les raffinements au cours du processus de synthèse sont complexes et impliquent une transformation profonde de la spécification

fonctionnelle en une architecture cible, il est difficile de donner un ensemble d'algorithmes propices à tous types d'applications. Il est donc nécessaire que le flux de synthèse soit flexible et qu'il puisse être guidé par le concepteur. Cela concerne par exemple le traitement des boucles et des sous-programmes, l'utilisation des unités fonctionnelles externes, le choix d'architecture, etc.

- *Génération d'un modèle de conception efficace pour la synthèse au niveau transfert de registres.*

  La synthèse comportementale doit tenir compte des exigences de la synthèse au niveau transfert de registres pour que l'on puisse obtenir des résultats efficaces au niveau du nombre de portes, de la vitesse et de la consommation.

- *Génération d'une description au niveau transfert de registres lisible et compréhensible.*

  Même si la description générée par l'outil de synthèse n'est pas nécessairement adressée à être lue par les concepteurs, un code compréhensible peut l'aider à mieux comprendre le processus de synthèse, à connaître le style d'écriture nécessaire afin d'obtenir des résultats plus efficaces ainsi qu'à trouver des erreurs éventuelles dans la conception.

- *Étendre le domaine d'application.*

  Les applications industrielles peuvent rarement être classées dans un seul domaine d'application : elles se composent de parties algorithmiques avec des calculs arithmétiques et de descriptions de protocoles de communication nécessitant un traitement plutôt orienté flux de contrôle. Pour la plupart des outils de synthèse, c'est cette combinaison qui pose le plus de problèmes bien que cet objectif soit un des plus importants. Un outil de synthèse général doit permettre la combinaison du contrôle avec les calculs arithmétiques avec la plus grande liberté possible.

- *Étendre le sous-ensemble du langage d'entrée.*

  Une des difficultés importantes est la composition de la description comportementale pour la synthèse. Comme présenté dans le chapitre 4, le langage d'entrée est restreint à un sous-ensemble spécifique à l'outil de synthèse en question. Ce sous-ensemble est déterminé principalement par l'ensemble d'algorithmes utilisé durant l'ordonnancement et la représentation interne de la description comportementale. Les outils de synthèse doivent restreindre ce sous-ensemble le moins possible.

Ces restrictions concernent principalement la complexité des opérations arithmétiques, l'utilisation des instructions d'attente et les structures plutôt spécifiques à la synthèse au niveau transfert de registres. Surtout ces deux derniers posent le plus de problèmes du point de vue des concepteurs parce que ce genre de limitations empêche l'insertion du code au niveau transfert de registres à la description comportementale, souvent nécessaire pour les traitements des trames, des entêtes d'un protocole, etc.

- *Réduire le temps de synthèse*

  Comme présenté dans le chapitre introductif, la synthèse comportementale permet l'exploration de l'architecture à un niveau beaucoup plus élevé que celle de la synthèse au niveau transfert de registres. Pour que cette exploration puisse être efficace, la durée nécessaire pour effectuer le processus de synthèse doit être raisonnablement courte.

3.4 Nouvelle approche pour la synthèse comportementale : contributions

Cette thèse présente deux principaux axes des travaux. Le premier s'intéresse à démontrer l'efficacité des outils de synthèse comportementale de troisième génération à travers un nouveau flux de conception utilisé pour plusieurs applications industrielles. Le deuxième axe décrit un ensemble d'algorithmes qui permettent un processus de synthèse efficace même pour un domaine d'application très étendu.

Les caractéristiques principales de notre flux de synthèse sont les suivantes :

- Le code généré peut exploiter entièrement les capacités de la synthèse au niveau transfert de registres. Il est représenté sous forme d'une machine d'états finis au niveau transfert de registres. Le partage des ressources est effectué par la synthèse au niveau transfert de registres. Le style d'écriture de la machine d'états finis est choisi de manière à ce que cette tâche puisse être efficacement accomplie. Le processus de synthèse comportementale est pratiquement restreint à l'ordonnancement étendu par un module permettant la transformation et la réécriture du code pour la synthèse au niveau transfert de registres.

- Le modèle d'entrée est défini par un sous-ensemble du langage VHDL permettant de combiner du code purement comportemental, algorith-

mique et des spécifications de protocole de communication avec exactitude du cycle d'horloge. Ce modèle d'entrée avec un style d'écriture très peu restreint permet d'élargir considérablement le domaine d'application pour la synthèse comportementale. Cela rend possible la combinaison des parties de contrôle décrivant souvent des protocoles de communication complexes et les algorithmes dominés par le traitement de données. De surcroît, cette technique aide l'intégration de modules comportementaux dans les sous-systèmes écrits au niveau transfert de registres. Cette spécificité est essentielle du point de vue de l'intégration de l'outil de synthèse comportementale aux flux de conception existants.

- Le modèle généré au niveau transfert de registres est relativement facile à comprendre. Une forte correspondance existe entre les descriptions initiales est celles générées. Cette spécificité est essentielle aussi pour les concepteurs parce que la correspondance des descriptions permet d'évaluer les résultats de la synthèse et aide à découvrir les points faibles de la description comportementale.

Ces caractéristiques et le flux de synthèse sont détaillés dans le chapitre 5.

3.5 Conclusions

Ce chapitre de thèse a étudié les raisons de l'acceptation faible de la synthèse comportementale en milieu industriel. Nous pouvons conclure que les outils de synthèse comportementale classique, générant une architecture cible composée d'un contrôleur et d'un chemin de données, sont généralement peu adaptée aux flux de conception existants :

- L'architecture générée est sur-décomposée ; l'allocation des unités fonctionnelles est fixé par l'outil de synthèse comportementale, ce qui restreint l'efficacité des nouvelles optimisations de haut niveau disponibles dans les outils de synthèse au niveau transfert de registres.

- La complexité du flux de conception des outils de synthèse comportementale et les limitations importantes sur le style d'écriture du modèle d'entrée rendent difficile l'intégration de ces outils aux flux de synthèse au niveau système.

Comme exemples, deux outils de synthèse comportementale classiques ont été présentés : Behavioral Compiler de Synopsys, développé principalement pour les applications dominées par le flux de données, et AMICAL, un outil universitaire, conçu pour les circuits de contrôle. L'étude menée sur ces deux

outils nous a amenés à définir un ensemble d'objectifs qui permet d'établir un nouveau flux de conception pour la synthèse comportementale basé uniquement sur l'étape de l'ordonnancement. Cette nouvelle approche est détaillée dans le chapitre 5.

Chapitre 4

Interprétation de VHDL pour la synthèse comportementale

Introduction

Du point de vue des concepteurs qui utilisent un outil de synthèse comportementale, l'une des tâches les plus difficiles est la prédiction de l'efficacité de l'architecture générée par l'outil de synthèse à partir de la description d'entrée. Cette difficulté est essentiellement due à la complexité des transformations effectuées lors du processus de synthèse. En effet, ces transformations peuvent induire une réorganisation importante du modèle initial. De surcroît, chaque outil a ses propres restrictions sur le style d'écriture respecté soit l'interdiction de certaines structures ou le respect de règles pour l'utilisation d'autres constructions.

La plupart des outils de synthèse comportementale sont sensibles au style d'écriture de la description initiale. L'efficacité de l'architecture résultante est fortement influencée par ce style d'écriture qui peut varier considérablement d'un outil à un autre.

Plusieurs langages dédiés à la modélisation de matériel sont de nos jours disponibles ; ils forment la famille des langages de description de matériel (HDL[1]) dont font partie VHDL[VHD94], Verilog[TM91], HardwareC[KM90]. Ces langages permettent la description de tout type de circuit grâce au large éventail d'instructions offertes [KJ95]. D'autres langages sont quant à eux dédiés tout spécialement à des circuits de traitement du signal, tels Silage[Hil85], Signal[GG87], Lustre[HCRP91], etc.

Malgré le nombre de langages de description de matériel élevé, ces langages se ressemblent tous et l'étude de ce chapitre, entièrement consacrée à

[1]En anglais : HDL : Hardware Description Language.

VHDL, peut être donc appliquée à d'autres langages. Comme ce langage a été initialement conçu pour la modélisation des circuits complexes, certaines structures ne sont pas adaptables pour la synthèse comportementale. Ainsi, chaque outil restreint-il son modèle d'entrée à un sous-ensemble du langage VHDL. Chaque section de ce chapitre introduira certains concepts du VHDL en se focalisant sur les aspects de synthèse comportementale. Leurs interprétations possibles seront présentées ainsi que les restrictions imposées par la majorité des outils de synthèse.

4.1 Le modèle VHDL

Un circuit décrit en VHDL est composé d'un ensemble de modules interconnectés. Chacun de ces modules est constitué d'une paire entité et architecture. L'entité définit l'interface du module vers l'environnement alors que l'architecture, ou dans certains cas plusieurs architectures, donnent la réalisation et la description algorithmique du module. Deux modèles principaux s'offrent pour l'interconnexion des modules :

- Les modules sont structurés sans hiérarchie. Ils sont instanciés et interconnectés dans une description structurelle qui, avec son interface, constitue le circuit entier. Ce modèle peut convenir aux systèmes relativement simples qui ne sont composés que de quelques unités.

- Dans la deuxième approche, les modules sont composés d'autres modules dont les descriptions ne peuvent plus être considérées comme purement comportementales : elles contiennent des constructions plutôt structurelles, telles que les affectations concurrentes, les instanciations d'autres composants, etc. Cette approche offre plusieurs avantages par rapport à la première. Elle rend possible la réutilisation de certains composantes de base. Avec le puissant mécanisme de configuration du langage VHDL, il est possible de combiner librement les niveaux d'abstraction dans la même description. En revanche, la simulation et la validation des descriptions sont plus complexes surtout, en ce qui concerne le circuit généré. Dans le cas de la plupart des outils de synthèse comportementale, la synthèse de chaque module s'effectue séparément et indépendamment des autres.

Pour la plupart des outils de synthèse comportementale, l'unité de compilation est la paire entité et architecture. L'entité définit l'interface entre le module et son environnement par l'intermédiaire de ports d'entrée et/ou de

sortie. La déclaration d'une entité décrit la vue externe du circuit en question. Ainsi, chaque module peut-il être considéré comme une boîte noire dont les ports sont spécifiés mais sans plus d'informations à propos des transformations subies par les entrées pour la génération des sorties. Une spécificité intéressante de l'entité VHDL est la possibilité de configurer la description à l'aide de paramètres génériques. Ces paramètres sont généralement utilisés pour spécifier de l'extérieur du module la taille des données, les valeurs constantes et parfois même certains détails d'exécution de l'algorithme. L'usage intensif des paramètres facilite la réutilisation des descriptions comportementales. Par contre, pour la synthèse, il nécessite des transformations et des optimisations supplémentaires, plus sophistiquées dont les techniques sont connues du monde de la compilation.

Une ou plusieurs architectures peuvent être attachées à chaque entité d'une conception. Hormis la partie déclarative, une architecture contient une liste d'actions concurrentes dont seuls les processus sont traités par la synthèse comportementale. Le rôle des autres actions concurrentes est plutôt structurel. Les instanciations de composant permet de décrire des systèmes hiérarchiques. Les affectations et les appels de procédure concurrents établissent des interconnexions entre les modules et peuvent insérer des structures combinatoires. Ces éléments structurels sont généralement ignorés par la synthèse comportementale et sont intégrés à l'architecture cible.

La possibilité de donner plusieurs architectures pour la même entité, autrement dit plusieurs réalisations, permet d'attacher plusieurs modèles du même module à la même entité. On peut notamment associer les descriptions avant et après le processus de synthèse. Lors de la simulation on peut choisir l'architecture qui est utilisée. Cela a une importance dans le cas des applications complexes où, la simulation au niveau bas étant plus lente, l'on peut économiser du temps de simulation. La seule condition qui s'impose est que les outils de synthèse ne modifient pas la déclaration de l'entité de la description. Cette approche de simulation a été employée dans le flux de vérification du circuit réalisant la gestion de trafic ATM qui est présenté dans la section 6.1.1.

4.2 Processus

Un processus VHDL se divise en deux parties. La partie déclarative contient toutes les déclarations locales au processus qui sont nécessaires à la description de comportement. La partie déclarative d'un processus peut contenir les déclarations des type, des sous-types, des constantes, des sous-programmes et des variables. Surtout, ces deux derniers ont une grande im-

portance pour la synthèse comportementale. La section 4.3 donne l'interprétation des variables pour la synthèse comportementale avec un accent sur sa comparaison sémantique avec les signaux. La section 4.4 présente les différentes possibilités de réalisation des sous-programmes.

Aussi le processus a-t-il toujours un corps associé à sa déclaration qui regroupe des instructions séquentielles. Ces instructions sont similaires à celles utilisées par les langages de programmation procéduraux tels que Ada, C ou Pascal. La tâche principale de la synthèse comportementale est la traduction de ces instructions en une architecture cible. Dans l'approche traditionnelle, cette architecture se compose d'une partie opérative représentée par un chemin de données et d'un contrôleur. D'autres approches, ne séparant pas le contrôleur et le chemin de données, génèrent directement une machine d'états finis suivant le modèle Moore ou Mealy.

Bien que les actions séquentielles soient similaires à celles des langages de programmation, elles ont des extensions spéciales destinées à spécifier les caractéristiques d'un circuit. Ces extensions permettent de manipuler le temps, les transactions des signaux et d'effectuer des opérations sur un seul bit ou sur des champs de bits. L'extension la plus importante est la possibilité de la synchronisation de la description à certains événements extérieurs. Dans un processus, deux possibilités s'offrent pour cette synchronisation :

- Une liste de sensibilité peut être attachée à chaque processus. Cette liste définit un ensemble de signaux auxquels le processus est dit *"sensitif"*, c'est-à-dire l'exécution des instructions ne commence que lorsqu'au moins un signal de cette liste change de valeur. Puisqu'un processus ayant une liste de sensibilité ne peut contenir d'autres instructions de synchronisation et qu'un processus peut être considéré comme une boucle infinie et inconditionnelle, la séquence d'instructions du corps du processus s'exécute une seule fois et le processus reste bloqué jusqu'à l'arrivée du prochain événement. Ce style d'écriture convient parfaitement aux modules contenant beaucoup d'opérations arithmétiques sans structures de contrôle et aux approches d'ordonnancement basées sur un graphe de flux de données. Dans une description n'ayant pas de synchronisations particulières, l'ordonnancement a une liberté dans la disposition des opérations dans les cycles de contrôle sous contraintes de ressources ou/et de nombre de cycles. Cette approche est aussi appropriée aux architectures pipelinées parce que le flux de données y est régulier.

- Un processus n'ayant pas de listes de sensibilités peut contenir un nombre illimité d'instructions WAIT. Cette instruction d'attente est

la clé pour forcer la synchronisation entre les processus concurrents, mais aussi avec l'environnement, par l'intermédiaire des signaux et des ports. La condition d'attente est définie par des clauses ou par des combinaisons de clauses :

- *clause de sensibilité :* La clause de sensibilité est similaire à la liste de sensibilité des processus. Sémantiquement, chaque processus ayant cette liste peut être transformé en un processus dont la première action est un instruction WAIT avec une clause de sensibilité. Par exemple :

  wait on *sig1* **until** *sig2 = '1';*

  Dans ce cas, l'exécution ne se poursuivra que si la valeur de signal *sig1* change et, en même temps, la valeur du signal *sig2* est égale à *'1'*. Comme les outils de synthèse comportementale synchronisent l'échantillonage des signaux aux fronts de l'horloge, l'interprétation et la réalisation de cette forme de WAIT posent de sérieux problèmes pour la synthèse. La plupart des outils l'excluent donc de leur sous-ensemble.

- *clause de condition :* La clause de condition spécifie une condition qui doit être respectée pour que l'exécution se poursuive. Cette clause peut être utilisée pour la synchronisation explicite entre le processus et d'autres modules. Un exemple typique est la poignée de main pour les échanges de données entre deux unités de conception. Cette clause est admise par tous les outils de synthèse comportementale. Par contre, certains d'entre eux restreignent cette clause à une attente du changement de l'horloge. Par exemple :

  wait until *clk'event* **and** *clk = '1';*

  ou

  wait until *rising_edge (clk);*

  D'un point de vue comportemental, une instruction d'attente peut être une instruction synchrone aussi bien qu'asynchrone. Les instructions synchrones sont évaluées uniquement sur des fronts montants ou descendants de l'horloge. Une instruction d'attente synchrone typique est la suivante :

  wait until *sig1 = '1'* **and** *rising_edge (clk);*

  ou, une attente sur le front descendant :

  wait until *sig1 = '1'* **and** *falling_edge (clk);*

Les instructions d'attente asynchrones sont évaluées indépendamment de l'horloge. En ce qui concerne la synthèse comportementale, les mêmes problèmes s'imposent que pour le traitement de la clause de sensibilité. En fait, dans le modèle au niveau transfert de registres produit par la synthèse comportementale, toutes les instructions sont synchronisées au changement de l'horloge. Ainsi le traitement de ces attentes devient-ils difficile à réaliser.

– *clause temporelle :* La clause temporelle provoque la suspension du processus jusqu'à l'expiration du délai fixé qui est défini en temps absolu. Certains outils de synthèse comportementale [ROJ94, NBD92] ignorent cette clause car les outils de synthèse ne peuvent garantir que le circuit réalisé respecte de telles caractéristiques temporelles. Pour surmonter ce problème, l'outil de synthèse comportementale d'IBM [Sau87] utilise un attribut pour spécifier le temps de cycle et chaque expression temporelle est convertie en étapes de contrôle. Par exemple, l'instruction d'attente

wait for *2*cycle_type;*

est traduite en deux étapes de contrôle par l'ordonnancement. Autrement dit, cette instruction correspond à deux WAIT avec une clause de condition où la condition est uniquement l'attente du changement de l'horloge. D'autres outils de synthèse, comme BC[Kna96], n'acceptent cette clause d'attente qu'au sein des boucles qui sont destinées à être pipelinées. Dans ce cas, la clause temporelle spécifie la profondeur du pipeline en terme du cycle d'horloge.

Ainsi pouvons-nous conclure que chaque instruction d'attente du langage VHDL supportée par les outils de synthèse comportementale provoque un ou plusieurs états supplémentaires dans le contrôleur. Ces états correspondent aux suspensions de l'exécution du processus lors de la simulation de la description. Toutes les instructions situées entre deux instructions d'attente consécutives sur un chemin d'exécution s'exécutent pendant le même cycle de simulation.

4.3 Affectations de signaux et de variables

Le langage VHDL offre deux possibilités pour le stockage et la transmission des valeurs et des données : les variables et les signaux. Les signaux sont

principalement destinés à établir des connexions entre des modules concurrents, donc leur rôle est plutôt structurel. Les variables sont similaires aux variables des langages de programmation : elles permettent de sauvegarder des valeurs intermédiaires dans une description algorithmique afin d'accélérer la vitesse de la simulation, d'augmenter la lisibilité du code et de permettre la réutilisation des valeurs intermédiaires. Les signaux sont déclarés dans la partie déclarative de l'architecture d'une description. Aussi, du point de vue sémantique, faut-il considérer les ports de l'entité et certains paramètres des sous-programmes comme des signaux. Les motifs de l'utilisation des signaux peuvent être les suivants :

- Dans un processus VHDL, la communication avec son environnement et avec d'autre processus concurrents se fait par l'intermédiaire de signaux et de ports [CDT91]. L'écriture des signaux s'effectue par des affectations syntaxiquement distinguées des affectations de variables.

- Dans la partie structurelle d'une description, les composants sont connectés avec d'autres composants et/ou avec des entrées/sorties à l'aide de signaux. Dans ce cas, les signaux sont considérés comme des éléments purement structurels et, du point de vue de la réalisation, ils correspondent à des fils de connexion.

- Les signaux peuvent être également utilisés dans les affectations concurrentes. Ces affectations permettent de connecter conditionnellement un signal avec un autre signal ou un bloc combinatoire. Par exemple, les multiplexeurs ainsi que d'autres opérations combinatoires peuvent être décrits. Dans ces cas, le rôle reste structurel, mais les connexions sont commandées par d'autres signaux ou des ports d'entrée souvent à partir d'un processus.

Dans une description comportementale, il est possible que plusieurs valeurs soient affectées à un seul signal dans le même cycle de simulation. Dans ce cas, un mécanisme de résolution définit la valeur courante du signal qui constitue un appel d'une fonction de résolution. Dans certains outils de synthèse comportementale, ces fonctions sont considérées comme des structures synthétisables et l'outil est capable de produire un réseaux logique permettant la réalisation des affectations multi-sources. Par exemple, cela peut comprendre un bus à trois états. En général, ces outils imposent certaines restrictions sur le style d'écriture de ces fonctions de résolution ainsi que sur les affectations.

Les clauses temporelles des affectations de signaux sont prohibées ou ignorées par la majorité d'outils de synthèse comportementale. Ces clauses permettent de retarder l'exécution de l'affectation par un temps absolu. C'est

le même problème que celui posé par la synthèse des instructions d'attente avec une clause temporelle. Les outils de synthèse ne peuvent garantir dans aucun cas le respect de ces délais. Ignorer ces clauses paraît plus pratique du point de vue des descriptions que les prohiber, parce qu'elles permettent d'augmenter la précision de la description pour la simulation.

En VHDL, les affectations de signaux sont effectifs avec en certain retard. Chaque signal de la description ne prend sa nouvelle valeur qu'à la fin du cycle de simulation, c'est-à-dire lorsque tous les processus sont bloqués par des instructions d'attente et avant que le prochain cycle de simulation ne commence. Par conséquent, si un signal est affecté deux ou plusieurs fois dans le même cycle de simulation, seule la dernière affectation sera prise en compte.

Le sémantique de la mise à jour des valeurs des signaux VHDL pose des problèmes pour la synthèse comportementale. Nous avons défini la tâche principale de l'ordonnancement comme la décomposition des séquences d'opérations entre deux instructions d'attente en plusieurs cycles. Cela peut provoquer des états supplémentaires dans le contrôleur, ce qui change le comportement de la description initiale. De sûrcroit, ces états supplémentaires décalent généralement la mise à jour des signaux et des sorties du module. Le même problème se produit en ce qui concerne la lecture des entrées. Un pas d'exécution pouvant être décomposé en plusieurs cycles de contrôle, une lecture d'une entrée après le premier cycle de contrôle transgresse la sémantique du VHDL et le comportement des entrées/sorties peut être modifié. La figure 4.1 illustre un cas où l'introduction d'un état supplémentaire change le comportement d'une description VHDL. Dans ce cas les valeurs des signaux *out1* et *out2* sont décalées par rapport au comportement initial et la lecture retardée de l'entrée *in1* provoque une affectation erronée du signal *out3*.

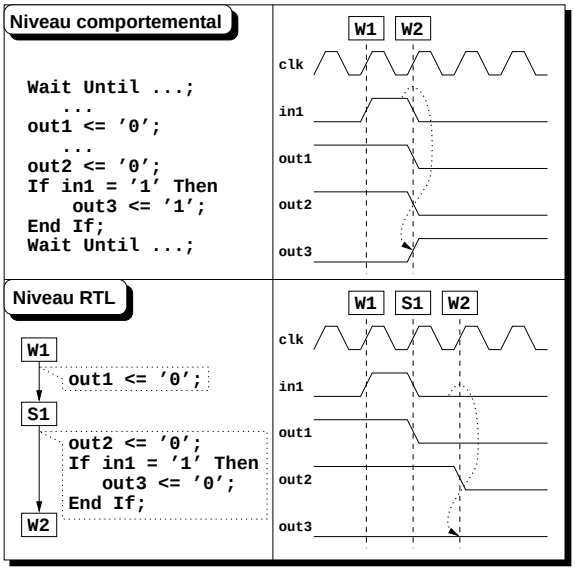


Figure 4.1: Sémantique des affectations de signaux

Pour préserver l'ordre de l'écriture et celui de la lecture, des registres supplémentaires doivent être insérés dans l'architecture du circuit pour la

mémorisation des entrées et des sorties. Entre deux instructions d'attente, il faut mémoriser les entrées dans le premier cycle, et mettre à jour des sorties dans le dernier.

Du point de vue de la réalisation, nous pouvons distinguer deux types de signaux. Les signaux ne servant qu'à connecter des unités entre elles ne nécessitent pas de mémorisation. En revanche, ceux qui établissent la communication entre les processus ainsi que les ports de l'entité doivent être mémorisés afin qu'ils gardent leurs valeurs. Le VHDL ne distingue pas syntaxiquement les deux types de signaux. C'est plutôt la façon d'affecter une valeur à un signal, qui détermine la réalisation. Si l'affectation est sensible au changement d'un signal qui est d'ailleurs considéré comme l'horloge, le signal sera mémorisé et deviendra un registre.

Le comportement et la sémantique des variables sont exactement les mêmes que ceux des variables connues dans les langages de programmation. Par rapport aux signaux, les variables prennent leurs nouvelles valeurs au moment de l'affectation et elles les sauvegardent jusqu'à la prochaine affectation. En VHDL, on distingue deux types de variables :

- Les variables déclarées dans la partie déclarative des processus qui ne sont visibles que pour les actions séquentielles du processus.

- Les variables partagées dans l'architecture, l'entité et les packages. La plupart des outils de synthèse comportementale ne supportent pas les variables partagées, parce que leur sémantique en cas d'écriture multi-source pose des problèmes de réalisation. Cette difficulté se manifeste surtout pour les variables déclarées dans les packages parce qu'elles sont partagées entre plusieurs modules du système. En général, les modules étant synthétisés indépendamment les uns des les autres, ce type de variables partagées ne peut être traité sans une étape de synthèse préalable qui doit fixer leur protocoles d'accès dans les descriptions comportementales.

Les variables dont la durée de vie dépasse un seul cycle d'horloge doivent être mémorisées dans l'architecture résultante. En revanche, les variables temporaires qui ne sont utilisées que dans une étape de contrôle peuvent être éliminées par une analyse de chemin de données durant le processus de synthèse comme présenté dans la section 5.5.4.

4.4 Sous-programmes : procédures, fonctions et opérateurs

Les sous-programmes du VHDL (procédures, fonctions et surcharges des opérateurs) sont utilisés pour simplifier le codage des parties de code répétitives ou communes. La différence principale entre les appels de procédure et ceux de fonction est que les fonctions ont toujours une valeur de retour est leurs appels font toujours partie d'une expression. Ainsi le rôle des fonctions est-il plutôt de regrouper des actions servant à réaliser des calculs ou des transformations sur des données spécifiées par les paramètres. Une spécificité intéressante du langage VHDL est la possibilité de surcharger les opérateurs arithmétiques pour un type de donnée spécifique. Dans le langage, ces opérateurs sont considérés comme des fonctions spéciales dont les noms sont les opérateurs. Ce mécanisme permet d'améliorer la lisibilité du code comportemental et de cacher la réalisation de l'unité fonctionnelle qui réalise l'opérateur. Comme ces unités fonctionnelles sont souvent décrites au niveau d'abstraction plus bas, au niveau transfert de registres ou de portes, où elles font partie d'une bibliothèque spécifique à une technologie, ces fonctions surchargées donnent un modèle d'exécution pour le processus de simulation qui, à part la réalisation, permettent de vérifier la validité des données. Un exemple typique pour cette approche est l'ensemble des packages IEEE arithmétiques acceptés par la plupart des outils de synthèse.

La souplesse d'utilisation des appels de sous-programmes est assurée par le mécanisme de passage de paramètres. Ces paramètres dont les sémantiques d'exécution peuvent être différentes (signal, constante et variable) influencent l'exécution des sous-programmes. Similairement aux entités, le concept de sous-programmes génériques existe également. Cela signifie que la taille de données traitées au cours de l'exécution peut varier selon les paramètres de l'appel courant. L'interprétation de ce type de sous-programmes dépend du traitement appliqué aux appels durant la synthèse.

L'usage de tous les éléments déclaratifs des sous-programmes sont restreints aux actions séquentielles du corps du sous-programme. Du point de vue de la synthèse, ce sont les variables qui ont une grande importance. Comme ces variables sont locales aux sous-programmes, les registres dans l'architecture générée peuvent être partagés entre les variables déclarées dans des sous-programmes différents pour minimiser les ressources nécessaires. À l'intérieur d'un processus, une hiérarchie d'appels de procédures et de fonctions peut être définie. Ceci correspond au mécanisme d'appel de sous-programmes communément utilisé par les langages de programmation modernes. Par contre, les appels récursifs, soit directs ou indirects, sont générale-

ment exclus du sous-ensemble du VHDL accepté par les outils de synthèse comportementale.

Les appels de sous-programmes peuvent être réalisés de plusieurs manières par les outils de synthèse :

- Le premier schéma consiste en une expansion en ligne de l'appel. L'appel est remplacé par le code du sous-programme correspondant. Évidemment, la mise à plat du corps du sous-programme accroît la taille de la partie de contrôle de l'architecture générée. Cependant, les opérations à l'intérieur du sous-programme peuvent partager les ressources avec les opérations du processus appelant. Bien qu'un grand nombre d'outils de synthèse comportementale supportent la mise à plat de procédures et de fonctions, beaucoup d'entre eux imposent des restrictions supplémentaires par rapport aux actions situées à l'intérieur des processus. Dans la plupart des cas, ce style d'écriture correspond plus ou moins à celui du niveau transfert de registres. Par exemple, dans l'outil Behavioral Compiler [Kna96], les boucles et les instructions d'attente sont prohibées à l'intérieur des sous-programmes. Dans notre approche, nous n'avons pas imposé de limitations particulières. Le style d'écriture supporté est le même que dans un processus.

- La procédure ou la fonction appelée est synthétisée comme une unité supplémentaire, autrement dit une unité fonctionnelle indépendante. Dans ce cas, l'appel est remplacé par un protocole de communication pour le passage de paramètres entre le module principal et l'unité fonctionnelle correspondant à la procédure ou fonction appelée. Un traitement spécial basé sur cette approche est offert par l'outil DesignWare de Synopsys. Chaque sous-programme est synthétisé indépendamment des autres parties de la conception et ces modules deviennent des composants réutilisables comme d'autres composants fournis par la bibliothèque technologique.

- L'appel est interprété comme une opération correspondant à une unité fonctionnelle existante au niveau du chemin de données. Ce modèle permet la réutilisation de composants complexes appelés coprocesseurs. Dans ce cas, quelques restrictions sont souvent imposées sur le style d'écriture telles qu'un modèle d'exécution fixe (le corps de la procédure ou de la fonction doit être ordonné en nombre fixe d'étapes de contrôle). Ce type d'interprétation nécessite que l'outil de synthèse utilisé adopte un modèle d'architecture basé sur le modèle de machine d'états finis avec des co-processeurs (FSMC[2][JDKR97].

[2]En anglais : FSMC : Finite State Machine with Co-processors.

4.5 Instructions conditionnelles

Le langage VHDL supporte des instructions séquentielles conditionnelles pour la représentation du comportement conditionnel telles que les instructions *if* et *case*.

L'interprétation des ces structures dépend fortement de l'architecture cible choisie par l'outil de synthèse. Principalement, trois types de réalisations peuvent être distingués dont deux sont liés à l'approche traditionnelle de la synthèse comportementale, c'est-à-dire à un flux de conception qui comprend aussi la génération de l'architecture composée d'une partie de contrôle et d'un chemin de données.

- Dans la première approche, l'évaluation de la condition de chaque instruction conditionnelle s'effectue dans la partie de contrôle de l'architecture résultante. Les comparateurs, les opérateurs arithmétiques et logiques font partie du contrôleur.

- L'évaluation des conditions peut être également effectuée au niveau de chemin de données. Cette approche permet d'augmenter le partage des ressources car les opérateurs des conditions peuvent être partagés avec ceux des expressions des affectations. En général, une expression conditionnelle engendre des transitions supplémentaires dans la machine d'états finis du contrôleur. Cette approche est mieux adaptée aux architectures programmables.

- La troisième approche est liée à la troisième génération des outils de synthèse comportementale. Le processus de synthèse se terminant après l'ordonnancement sans effectuer la génération de l'architecture, la réalisation des expressions conditionnelles est effectuée par l'outil de synthèse au niveau transfert de registres. La possibilité du partage des ressources dépend fortement du style d'écriture du code au niveau transfert de registres. Il est donc nécessaire d'explorer la capacité de la synthèse au niveau de transfert de registres ainsi qu'un algorithme puissant pour la réécriture du code qui adapte la hiérarchie des conditions de la description initiale au style d'écriture exigé.

4.6 Instructions itératives

Le langage VHDL, similairement à d'autres langages de programmation modernes, supporte les actions répétitives, telles que les boucles. Les types de boucles disponibles sont les boucles inconditionnelles (*loop*) et les boucles

dont le nombre d'exécution est lié à une condition exprimée dans la tête de boucle (*while* et *for*). Deux instructions supplémentaires, *exit* et *next*, permettent de sortir d'une boucle et de forcer l'exécution à la prochaine itération.

En général, les boucles nécessitent l'application d'algorithmes sophistiqués pour la synthèse comportementale car, à cause de leur nature répétitive, elles influencent davantage le temps d'exécution de la description comportementale que les autres instructions. Dans le but d'améliorer le résultat de la synthèse, des transformations complexes sont souvent appliquées. Certaines d'entre elles s'effectuent lors de la compilation VHDL ou dans une phase d'élaboration, d'autres sont plutôt liées aux algorithmes d'ordonnancement.

4.6.1 Boucles infinies

Les boucles infinies sont de la forme suivante :

infini: **loop**
    -- corps de la boucle
end loop *infini*;

Pour ce type de boucle, le nombre d'itérations est infini, ne dépendant pas d'une condition. En général, le délai d'exécution des boucles domine le temps d'exécution total du circuit. C'est pour cette raison que l'optimisation du rendement du corps de la boucle est essentielle du point de vue des performances du circuit. Le nombre d'itérations étant infini, la mise à plat ou le déroulement de la boucle n'est pas envisageable. Cependant, le pliage de boucle est préférable lorsque le débit du circuit est critique. Ce type de boucles correspond précisément à un processus n'ayant pas de liste de sensibilité.

4.6.2 Boucles avec des limites connues

En VHDL, les boucles avec des limites connues peuvent être décrites par une boucle FOR :

boucle1: **for** *limite_inférieure* bf to *limite_supérieure* **loop**
    -- corps de la boucle
end loop *boucle1*;

Dans cette instruction, la valeur des deux limites est connue au moment de la compilation ou de l'élaboration de la description. Ce type de boucles est le plus étudié à des fins de parallélisation. Le pliage ou le pipeline sont les techniques appliquées les plus répandues[GR94].

4.6.3 Boucles avec des dépendances de données

Ce type de boucle est comme suit :

boucle2: **while** *condition* **loop**
-- corps de la boucle
end loop;

Dans cette boucle, la condition peut dépendre de signaux externes ou de calculs effectués dans le corps de la boucle. Dans ce cas, ni la mise à plat ni le pliage ne sont applicables. Cependant, plusieurs techniques existent pour le recouvrement de telles boucles, dont l'ordonnancement rotatif [3] [CLS93], une approche à la base de transformations pour le pipeline de boucles. Il exécute répétitivement l'ordonnancement de manière rotative dans un sens, puis dans l'autre, de façon à obtenir un résultat plus compact selon les contraintes de ressources. Cette technique n'est applicable que pour des circuits orientés flux de données. L'ordonnancement pipeliné à base de chemins [RJ95] est une technique alternative pour le pipeline de boucles avec dépendances de données. Il considère qu'une boucle avec dépendances de données est exécutée 0, 1 ou plusieurs fois. Aussi génère-t-il des chemins d'exécution avec l'hypothèse que la boucle est exécutée autant de fois qu'il est nécessaire. Cette technique ne diffère guère du déroulement de boucle [GR94]. Deux déroulements de boucle permettent la détection de toute interdépendance existant entre différentes itérations d'une même boucle. Il faut cependant noter que l'optimisation du débit d'une boucle augmente le coût de la partie contrôle de l'architecture générée.

La combinaison des boucles, des instructions conditionnelles et de celles de synchronisation est restreinte par plusieurs outils de synthèse comportementale. Ceci est surtout vrai pour les algorithmes d'ordonnancement basés sur un graphe de flux de données car ces restrictions sont généralement imposées par la représentation intermédiaire et l'algorithme de l'ordonnancement.

4.7 Les types de données

Le langage VHDL est fortement typé. Il manipule signaux, variables et constantes ; ceux-ci peuvent être de tous les types prédéfinis par le langage lui-même ou par le concepteur. Comme tous les langages de programmation modernes, les types prédéfinis sont les types élémentaires tels que les entiers, les flottants, le booléen, etc. Ces types permettent aux concepteurs de construire des types complexes ou imposer des contraintes aux types existants. Les types VHDL incluent les types scalaires, composés, accès et fichier.

[3] En anglais : Rotation scheduling.

4.7.1 Les types scalaires

Les types scalaires ont des valeurs uniques et comprennent les entiers, les flottants et les types énumérés. La difficulté principale du traitement de ces types provient du fait que la plupart des outils de synthèse comportementale génèrent une architecture cible à partir de la description initiale. Cette architecture étant composée d'un ensemble d'unités interconnectées par des fils, et les unités étant souvent partagées entre elles, chaque type doit avoir une représentation au niveau de bits. C'est pour cette raison que la plupart des outils de synthèse comportementale limitent l'usage des types scalaires aux types avec des contraintes à partir desquelles la taille de chaque opération ou registre peut être déterminée. Pour les types énumérés, cette étape, appelée souvent synthèse de type, devient délicate. Quoique certains outils permettent au concepteur de spécifier la représentation interne pour chaque valeur énumérée par le mécanisme des attributs ou des pragmas, le choix optimal ne pouvant être fait qu'au niveau de synthèse logique. En ce qui concerne les types flottants dont le traitement est si coûteux dans un circuit, la plupart des outils les excluent de leur sous-ensemble. Pour les applications ou l'utilisation des types flottants est inévitable, leur réalisation sous forme de type virgule fixe peut être envisageable.

4.7.2 Les types composites

Les types composés incluent les structures et les tableaux. Les structures sont composées de champs de types souvent différents. Leur usage est répandu pour décrire des données structurées telles que les entêtes, les trames de communication, etc. En général, le compilateur ou l'outil de synthèse décompose les structures selon leur champs.

Les tableaux sont des structures de données où tous les éléments sont du même type. Les indices des tableaux doivent être d'un type scalaire autre que flottant. Pour les références des éléments d'un tableau, l'indexation peut être statique ou dynamique. La différence se fait selon que les valeurs des indices peuvent être déterminées dans la phase de compilation ou pendant l'étape d'élaboration de la description. À part les vecteurs de bits, les tableaux sont transformés en mémoires multi-ports ou, pour les tableaux ayant une taille relativement réduite en bloc de registres.

La déclaration d'un tableau est analogue à celle d'une mémoire car le tableau est composé de rangées de données. Plusieurs travaux [RGC94, ST95] traitent le problème des transformations de tableaux d'une description comportementale. Ces approches associent des groupes de tableaux à une même mémoire ou un tableau à plusieurs mémoires différentes. Ces associations

sont effectuées afin de réduire la surface du circuit et d'augmenter ses performances. En général, ces transformations sont insérées dans les algorithmes d'ordonnancement ou après l'étape d'ordonnancement. Suivant ces cas, deux approches se présentent :

- Le nombre et le type de mémoires sont fixés, ce qui correspond à une contrainte de ressources. L'association des tableaux aux mémoires est effectuée tout en minimisant le nombre d'étapes de contrôle et les traductions d'adresses. L'algorithme d'ordonnancement à boucles dynamiques intègre ce type de transformation [ORJ93].
- L'ordonnancement est déjà accompli, cette transformation consiste alors à trouver le nombre de modules mémoires, le nombre de ports de chaque mémoire et le groupement des tableaux dans ces mémoires. Cette approche est publiée dans [RGC94].

4.7.3 Les types accès

La plupart des langages de programmation permettent l'utilisation des types spéciaux pour référencer un autre objet par son adresse. En général, ces types sont appelés *pointeur* ou *référence*. Ces types sont également utiles pour la gestion dynamique de la mémoire. Quoique certains travaux s'intéressent à la synthèse des pointeurs [SM98b], la plupart des outils de synthèse l'excluent de leur sous-ensemble en raison des difficultés de réalisation. Ainsi les types accès restent-ils une structure du langage VHDL réservée à la simulation comportementale ou à la modélisation des systèmes.

4.7.4 Les types de fichier

À cause de la nature des types de fichier, ce ne sont pas des structures VHDL qui pourraient être synthétisables puisqu'ils référencent des fichiers externes pendant le processus de simulation. Ils peuvent cependant être utiles pour les modèles des ROMs d'un système pour décrire leur contenus. Ces contenus étant générés souvent automatiquement, par exemple par l'intermédiaire d'un compilateur, leur insertion dans la description comportementale peut s'avérer problématique. Une solution est publiée dans [OM96] qui consiste à effectuer une phase d'élaboration avant la synthèse comportementale. Cette élaboration permet de transformer les objets du type de fichier par leur valeurs décrites dans des fichiers externes.

4.8 Avenir du langage VHDL

L'évolution du langage VHDL ne s'est pas arrêtée au standard paru en 1993 [VHD94]. Plusieurs travaux s'intéressent à adapter le langage aux exigences de nos jours. Une des nouvelles approches est OO-VHDL, une extension du langage par le concept orienté objet. Dans le travail de [SMG95], le concept orienté objet est introduit pour les composants qui sont considérés comme des objets capables d'assurer certains services. Le concepteur peut appeler ces services à l'aide des structures semblables aux appels de procédures sans s'occuper spécialement de protocoles de communication avec le processeur exécutant les opérations. Cette approche rend plus compréhensible et plus lisible la description comportementale et, en même temps, élève le niveau d'abstraction en cachant certains détails d'implantation. Dans [SN95], une autre approche orientée objet est présentée. Le concept orienté objet étant assuré pour les types, il ressemble davantage aux concepts connus des langages de programmation.

Un autre ensemble d'extensions proposé par [Des99] constitue en une ouverture vers le niveau d'abstraction système. Il ajoute au langage VHDL la notion de protocole de communication abstrait entre les processus. Le concept d'*interface* permet la déclaration des canaux de communication asynchrones à travers lesquels les processus peuvent échanger des messages. Ces extensions du langage VHDL sont en cours de spécification et aucun outils de synthèse au niveau système n'est connu. La simulation des descriptions est assurée à l'aide d'un compilateur permettant les transformations des nouvelles structures en unités de compilation standardes.

4.9 Conclusion

Ce chapitre de thèse a étudié le langage VHDL en vue de la synthèse comportementale. Il avait comme but de définir un sous-ensemble comportemental du VHDL et d'analyser ses interprétations possibles pour la synthèse d'un circuit à partir d'une spécification comportementale. Nous pouvons conclure qu'il n'est pas possible de définir un sous-ensemble général et entièrement indépendant de tous les outils de synthèse. Ceci est dû, d'une part, à la complexité des transformations effectuées par les outils, et d'autre part, à la diversité des algorithmes et des approches réalisées par chacun des outils de synthèse. Le grand nombre de sous-ensembles spécifiques à chaque outils de synthèse met en question la réutilisation et la portabilité des descriptions comportementales entre différents outils.

Dans ce chapitre de thèse, nous avons présenté les diverses structures

VHDL au niveau comportemental. Le processus déclaré au sein de l'architecture a été considéré comme l'unité de compilation pour la synthèse. Nous avons présenté les réalisations envisageables des actions séquentielles du processus en mettant en relief les conséquences des instructions d'attente pour la synthèse, les transformations possibles des actions conditionnelles et, finalement, la diversité des traitements des sous-programmes offerts par les divers outils de synthèse. Les types jouant un rôle important dans le langage VHDL, leur interprétation et la réalisation des objets de divers types ont été également présentés. Dans la dernière partie du chapitre, les extensions futures du langage VHDL ont été discutées en vue de l'évolution des outils de synthèse pour satisfaire les exigences des conceptions de l'avenir.

Chapitre 5

Synthèse comportementale basée sur l'ordonnancement

Introduction

Le chapitre précédent a donné une analyse approfondie sur les limitations des outils de synthèse comportementale classiques. Les outils de synthèse au niveau transfert de registres, après une évolution considérable, permettent d'effectuer des étapes d'optimisations de haut niveau qui était considérées comme des tâches de la synthèse comportementale. L'architecture cible générée à partir d'une description comportementale est sur-décomposée et mal adaptée au style d'écriture exigé par la synthèse au niveau transfert de registres. Ce fait empêche l'utilisation de toutes les nouvelles capacités de la synthèse au niveau transfert de registres et donne des résultats moins satisfaisants que ce que l'on obtient à l'aide de méthodologies de conception traditionnelles.

D'autres faiblesses des outils de synthèse comportementale ont été également discutées : le style d'écriture restreint au niveau comportemental ne permettant pas la combinaison libre de différents styles de codage, le sous-ensemble du langage d'entrée limité, l'algorithme d'ordonnancement trop spécialisé à un domaine d'application spécifique, etc.

Le but de ce chapitre est donc d'introduire une nouvelle approche pour la synthèse comportementale en présentant un flux de synthèse basé principalement sur l'ordonnancement. Ce flux, au lieu de générer une architecture cible composée d'une partie opérative et d'une partie de contrôle, produit une machine d'états finis suivant les règles d'écriture de la synthèse au niveau transfert de registres. L'algorithme d'ordonnancement est fondé sur un graphe de flux de contrôle comme représentation intermédiaire et étendu par une

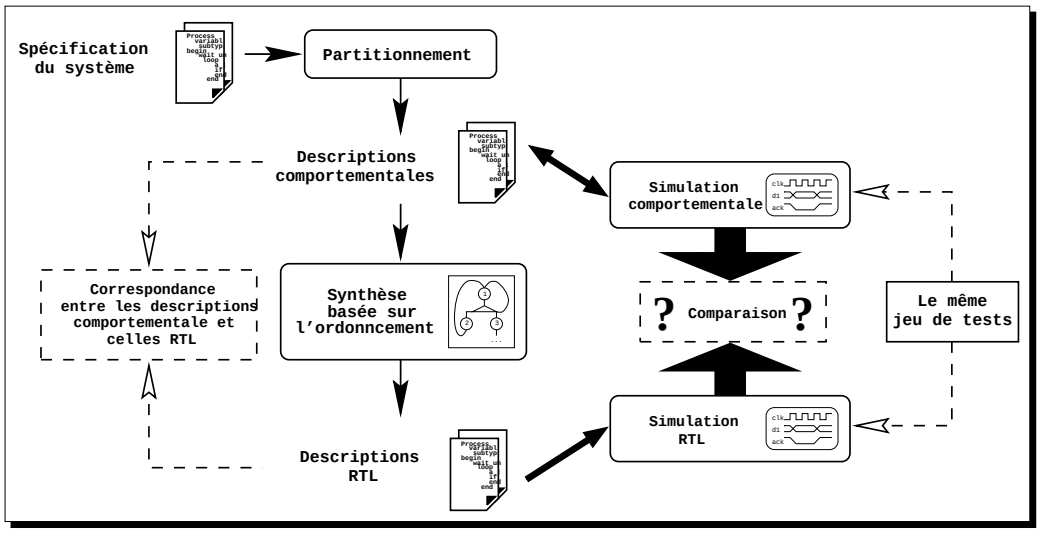


Figure 5.1: Flux de synthèse comportemental général

analyse de chemin de données pour élargir les domaines d'application vers les conceptions mixtes.

La section suivante présente la mise en contexte de notre approche de synthèse en mettant l'accent sur la problématique de la validation des résultats de la synthèse. La section 5.2 et la section 5.3 donnent un aperçu du modèle d'entrée comportemental accepté par notre outil et les spécificités du modèle de sortie au niveau transfert de registres. Les étapes de synthèse sont ensuite détaillées : la compilation et l'élaboration de la description initiale, l'algorithme d'ordonnancement, l'adaptation de la machine d'états finis à la synthèse au niveau transfert de registres et l'analyse de chemin de données.

5.1 Application de la synthèse comportementale

La conception d'un circuit intégré commence par la transformation de la spécification du système en un ensemble de descriptions synthétisables. Cette étape est souvent appelée partitionnement du système et comprend le découpage de la description en modules dont la complexité doit être raisonnablement réduite pour que ces descriptions puissent être décrites et synthétisées efficacement [SM98a]. Ce partitionnement est également nécessaire pour séparer les modules qui seront réalisés par différentes méthodologies de conception.

Les modules destinés à la synthèse comportementale sont d'abord élaborés

à l'aide de techniques de compilation, ensuite compilés dans une représentation interne. Cette représentation est un graphe de flux de contrôle étendu par la représentation des dépendances de données. Cette spécificité de la représentation interne et la technique d'ordonnancement reposant sur cette dernière peuvent être considérées comme la clé de l'efficacité de notre approche. Ce graphe peut représenter aussi bien le flux de contrôle que celui de données. L'ordonnancement, basé sur la recherche de chemins d'exécutions, génère une machine d'états finis correspondant à la description comportementale. Les transitions de cette machine d'états finis sont ensuite réécrites et optimisées pour la synthèse au niveau transfert de registres. La dernière étape de synthèse s'applique à ces transitions et consiste à effectuer une analyse de chemin de données. Ces algorithmes suivent les techniques d'ordonnancement basées sur un graphe de flux de données : chaînage des opérations, allocation et optimisation de registres, etc. À partir de la représentation interne, une description VHDL synthétisable à l'aide des outils de synthèse au niveau transfert de registres est générée. La simulation de cette description permet d'évaluer également les résultats de la synthèse comportementale.

Il est indispensable que la description comportementale aussi bien que celle au niveau transfert de registres puissent être simulées et validées. En raison de la simplicité et de la fiabilité de la simulation, il est préférable qu'elle soit effectuée à l'aide du même jeu de tests. Notre outil de synthèse le rend possible, ne transformant pas l'interface des modules. Il faut cependant noter que durant la synthèse comportementale, les changements des sorties peuvent être décalés dans le temps. Pour les jeux de tests, cela implique un style d'écriture plus sophistiqué qui évite l'usage du temps absolu pour générer les valeurs aux entrées de l'unité ou vérifier les réponses données par le module. Le fait que notre approche de synthèse ne modifie pas l'interface des modules a un second avantage : cela permet de réinsérer les modules déjà synthétisés dans le système entier et de combiner des descriptions comportementales avec du code au niveau transfert de registres. En général, la simulation au niveau d'abstraction plus bas étant considérablement plus lente, cette approche peut réduire le temps nécessaire pour la validation du système durant les premières étapes de la conception.

5.2 Modèle d'entrée comportemental

En général, les applications réelles sont constituées d'une combinaison de flux de contrôle complexe, de calculs arithmétiques et de parties décrivant des protocoles de communication avec exactitude du cycle d'horloge. La plupart des outils de synthèse traditionnels ne sont restreints qu'à un do-

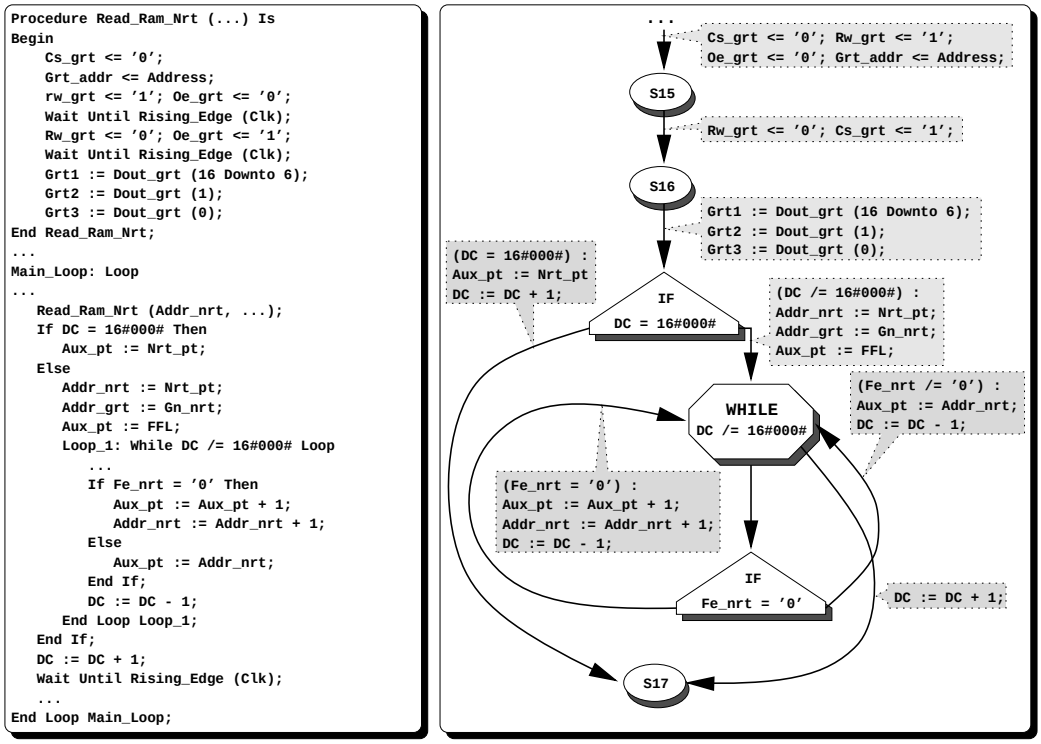
maine d'application et/ou qu'à un seul niveau d'abstraction. Ainsi pour le traitement de la plupart des applications, est-il inévitable que la spécification initiale soit sur-décomposée en modules dont le style d'écriture est pur et qui sont synthétisables à l'aide d'outils de synthèse spécialisés. Ce type de restriction est généralement dû à l'étape d'ordonnancement. La plupart des algorithmes d'ordonnancement ont été développés pour ne pouvoir accepter qu'un seul style d'écriture. Dans notre approche de synthèse, nous avons réussi à éviter un grand nombre de restrictions en proposant un flux d'ordonnancement flexible qui autorise la combinaison de plusieurs styles d'écriture afin de pouvoir traiter efficacement les applications à plusieurs critères. Cette spécificité a une grande importance puisqu'en général, il semble naturel aux concepteurs utilisant des langages de description de matériel, d'utiliser ce style d'écriture mixte.

Comme présenté dans la section 2.3.3.1, une spécification comportementale est composée principalement de deux types d'opérations : les instructions de synchronisation et les instructions de calcul. Les premières correspondent aux *états systèmes* lesquels définissent les interactions du processus avec l'environnement extérieur. En VHDL, ces instructions sont des instructions d'attente (*WAITs*). Ces points de synchronisation réalisent des attentes aux changements de signal ou des insertions d'un ou de plusieurs cycles d'attente. Ce modèle d'exécution considère la description comportementale comme une machine d'états finis de haut niveau où les états systèmes sont des points de synchronisation et les transitions de système se composent de toutes les opérations qui s'exécutent en un cycle de simulation entre deux points de synchronisation.

Ces transitions de système sont aussi appelées pas d'exécution[1]. Elles peuvent contenir des instructions séquentielles formant des boucles, des branchements conditionnels ou des affectations des expressions à des variables et à des signaux. La mise à jour des sorties de la description s'effectuant aux points de synchronisation, le délai de l'exécution des transitions de système est transparent du point de vue de l'environnement externe. Par conséquent, l'ordre d'exécution des opérations des transitions de système peut être changé durant le processus d'ordonnancement tant que le comportement des entrées/sorties du système reste inchangé. Naturellement, le décalage des opérations manipulant des entrées/sorties n'est pas permis.

La figure 5.2 montre un fragment d'une description VHDL comportementale extraite d'une application réalisant une gestion de trafic ATM [MSS+99] et la machine d'états finis correspondante. Dans cet exemple, le code comportemental accède à une mémoire à l'aide d'appels de procédure. Cette

[1]En anglais : execution thread.



(a) (b)

Figure 5.2: Spécification VHDL (a), et modèle de machine d'états finis au niveau système (b)

procédure réalise le protocole de communication avec la mémoire avec exactitude du cycle d'horloge et, par conséquent, elle contient des instructions d'attente. La description principale est formée d'une boucle dont la condition dépend de l'exécution de son corps ; elle se compose d'une hiérarchie d'instructions conditionnelles sans instructions d'attente. Ce style d'écriture semble entièrement naturel du point de vue du codage de l'algorithme, néanmoins, la plupart des outils de synthèse comportementale n'acceptent pas cette liberté dans l'écriture. Au sein de la procédure *Read_Ram_Nrt*, les états systèmes **S15** et **S16** correspondent aux instructions *WAIT* qui sont nécessaires pour le protocole d'accès à la mémoire. Dans la représentation sous forme d'une machine d'états finis, les états sont explicites. En revanche, les transitions sont préservées sous leur forme algorithmique. Ce modèle combine la notation des machines d'états finis traditionnelles et celle des graphes de flux de contrôle. Plusieurs transitions peuvent débuter par le même état, mais elles peuvent partager les mêmes opérations, ce qui permet une notation très compacte. Par exemple, plusieurs transitions différentes

se situent entre les états **S16** et **S17**. L'ensemble de chemins d'exécution possibles entre deux états systèmes définit une transition de système. Naturellement, ce ne sont pas toutes les transitions qui peuvent être exécutées en un seul cycle d'horloge et elles doivent donc être décomposées en plusieurs états. Par exemple, certaines transitions partant de l'état **S16** contiennent une boucle dont le nombre d'itérations n'est pas prévisible durant la synthèse. Cette boucle doit être donc décomposée en plusieurs transitions. Les sections suivantes présentent l'algorithme à l'aide duquel cette étape de synthèse est effectuée.

Les principales spécificités du style d'écriture de notre outil de synthèse sont les suivantes :

- *Instructions d'attente et de contrôle dans les sous-programmes.*

  Les spécifications au niveau comportemental décrivant des protocoles de communication sont facilement exprimées sous forme de procédures formées de structures de contrôle complexes et d'instructions d'attente. Ce style d'écriture des spécifications peut considérablement réduire la taille et la complexité du modèle initial. La synthèse au niveau transfert de registres ne permettant pas ce style d'écriture dans les sous-programmes, chaque appel nécessite un traitement spécial qui remplace les appels par le corps du sous-programmes et substitue les paramètres par leur valeur courante.

- *Description des processus complexes.*

  Le nombre de lignes nécessaires pour décrire des processus complexes au niveau transfert de registres tend à s'accroître de plus en plus. Ainsi les concepteurs sont-ils contraints de décomposer les descriptions en processus moins complexes pouvant être traités plus facilement. En général, cette stratégie demande du code supplémentaire qui réalise la communication entre les processus. Souvent, à cause de cette communication supplémentaire exigée par la lisibilité et la maniabilité du code, l'efficacité de la synthèse est réduite. La compacité du code comportemental et la possibilité d'une transformation plus ou moins automatique au niveau transfert de registres est la principale valeur ajoutée de la synthèse comportementale pour les processus complexes.

- *Instructions d'attente comportementale.*

  Le modèle d'entrée étendu du code comportemental permet les instructions d'attente dont la condition peut exprimer une attente sur un autre signal que l'horloge. Le code produit au niveau transfert de registres

décrit des machines d'états finis qui sont aussi efficaces que le modèle d'une machine d'états finis écrit par un concepteur.

- *Combinaison de boucles, d'instructions conditionnelles et d'instructions d'attente.*

  Probablement, la possibilité de ces combinaisons est la différence la plus importante entre le style de codage au niveau comportemental et au niveau transfert de registres. La taille du modèle comportemental est considérablement réduite car il n'existe pas de restrictions sur les instructions d'attente, les structures conditionnelles et les boucles. La synthèse comportementale est capable d'effectuer certaines optimisations et transformations sur la description qui sont difficiles à faire manuellement. Telles sont, par exemple, les techniques pour le traitement des boucles et des sous-programmes. Naturellement, ces transformations rendent plus difficile à comprendre et à analyser la description générée au niveau transfert de registres.

5.3 Modèle de sortie au niveau transfert de registres

Le modèle de sortie des outil de synthèse comportementale constitue une description VHDL au niveau transfert de registres qui est directement générée à partir de la représentation interne de la machine d'états finis construite par l'algorithme d'ordonnancement. Cette description est relativement facile à comprendre, à part quelques constructions complexes dues aux boucles imbriquées combinées à des instructions d'attente. Cela est dû au fait que l'algorithme d'ordonnancement déroule la première et la dernière itération de chaque boucle et que des variables intermédiaires sont introduites afin de chaîner les opérations aux dépendances de données. Comme le but principal est la génération d'une description qui doit être parfaitement adaptée à la synthèse au niveau transfert de registres, son style d'écriture suit les règles imposées par des outils de synthèse au niveau transfert de registres. L'efficacité de ce code est assurée par plusieurs facteurs :

- *Modèle basé sur une machine d'états finis.*

  La structure principale de la description générée provient du modèle utilisé par les concepteurs pour le codage manuel au niveau transfert de registres [KB98]. Ce modèle comprend deux processus à l'intérieur de l'architecture : un pour le réseau combinatoire qui produit les valeurs de

sortie et le prochain vecteur d'état ; un autre pour réaliser les registres et la mémorisation des sorties. Le vecteur d'état est d'un type énuméré, des *latches* sont évités, il n'y a pas de boucles asynchrones, la liste de sensibilité des processus qui est générée automatiquement est complète.

- *Représentation non-hiérarchique.*

  La représentation non-hiérarchique du modèle VHDL permet à la synthèse au niveau transfert de registres d'effectuer une optimisation plus complète tout en évitant des résultats inefficaces provoqués par la hiérarchie profonde et non voulue.

- *Allocation de ressources et association des unités fonctionnelles effectuées par la synthèse au niveau transfert de registres.*

  Les outils de synthèse logique sont capables de combiner très efficacement l'optimisation logique avec l'allocation de ressources pour des unités fonctionnelles. Par exemple, une multiplication par une constante donnée nécessite environ un quart des portes logiques qu'un multiplieur général. Suivant la valeur de la constante, la surface occupée par le multiplieur spécialisé peut être considérablement réduite [DW99]. De surcroît, l'optimisation logique supplémentaire est également possible entre les modules générés et les portes logiques qui les entourent.

- *Codage des conditions.*

  La plupart des outils de synthèse comportementale réalisent les expressions conditionnelles comme des opérations du chemin de données. Cette approche peut introduire de l'inefficacité au niveau surface et délai du chemin critique. Dans notre approche, l'étape d'ordonnancement garde la structure des instructions algorithmiques dans la machine d'états finis générée. Cela permet à la synthèse au niveau transfert de registres d'appliquer des techniques d'optimisations plus efficaces tels que le partage des comparateurs, le chaînage de données et celui de contrôle. La description générée est formée de manière à ce que le partage des ressource et le chaînage des opérations puissent être efficaces. En plus, dans le cas de décomposition du modèle au niveau transfert de registres en un contrôleur et en un chemin de données, les transitions du contrôleur produites par la synthèse au niveau transfert de registres doivent être générées explicitement. Cela entraîne généralement des duplications du code et une conception inefficace.

- *Lisibilité.*

  Les structures du code généré au niveau transfert de registres ressemblent au modèle de la machine d'états finis initiale. Notre outil de synthèse utilise une stratégie de nommage qui préserve autant que possible les noms des objets donnés par le concepteur. Lorsqu'un nouvel objet est créé, par exemple un état ou une variable, son nom est toujours relatif au contexte de sa création. L'attribution de ces nouveaux noms suit des règles connues. Cette technique de nommage rend encore plus lisible aux concepteurs la description générée.

5.4 Élaboration de la description et construction du graphe de flux de contrôle

L'élaboration de la description comportementale comprend la compilation du code écrit en VHDL et son optimisation à l'aide de techniques connues du monde de la compilation des langages de programmation[ASU86]. Pour la compilation VHDL, un compilateur a été développé qui permet une vérification syntaxique et une traduction de la partie comportementale en format intermédiaire. Les optimisations effectuées sont l'élimination du code mort, la simplification des expressions arithmétiques et logiques dans les conditions ainsi que dans les affectations, le traitement des boucles FOR avec leur déroulement ou leur transformation en boucle WHILE, etc. Les paramètres génériques sont aussi éliminés dans cette phase avec leur remplacement par les valeurs courantes.

Le graphe de flux de contrôle est ensuite construit. Notre graphe est étendu par une représentation du flux de données qui permet l'analyse des dépendances de données entre les opérations pendant le processus de synthèse. C'est également au cours de cette étape de synthèse que la mise à plat des appels de sous-programmes est effectuée. Les opérations du corps des sous-programmes sont insérées dans le graphe de flux de données comme si elles étaient utilisées dans le processus principal.

5.5 Ordonnancement

Dans notre outil de synthèse, nous effectuons l'étape d'ordonnancement à l'aide de l'algorithme étendu de l'ordonnancement à boucles dynamiques (DLS[2]). Cette représentation permet de décrire à la fois le flux de contrôle

[2]DLS : Dynamic Loop Scheduling.

	Mode d'ordonnancement des pas d'exécutions		
Mode pour les entrées/sorties	**Mode en un cycle**	**Mode des macro-états**	**Mode des états comportementaux**
E/S fixes	**1. En un cycle E/S fixes**	**2. Macro E/S fixes**	**3. Comportementaux E/S fixes**
E/S flottantes	**4. En un cycle E/S flottantes**	**5. Macro E/S flottantes**	**6. Comportementaux E/S flottantes**

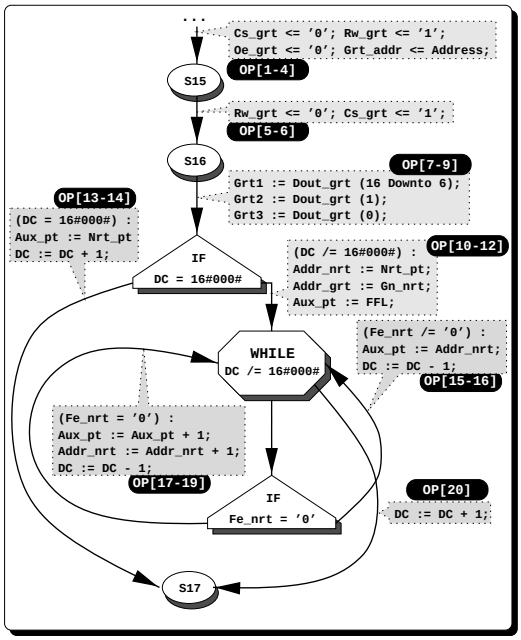
Figure 5.3: Modes d'ordonnancement

et le flux de données. L'algorithme du DLS s'exécute en deux phases : la génération des chemins d'exécution partant du graphe de flux de contrôle et la génération de la machine d'états finis. Un fragment du code de l'exemple ATM est utilisé pour illustrer ces étapes. La figure 5.2 montre le graphe de flux de contrôle correspondant à la description. Les opérations sont attachées aux arcs de ce graphe et sont numérotées pour la présentation de l'algorithme.

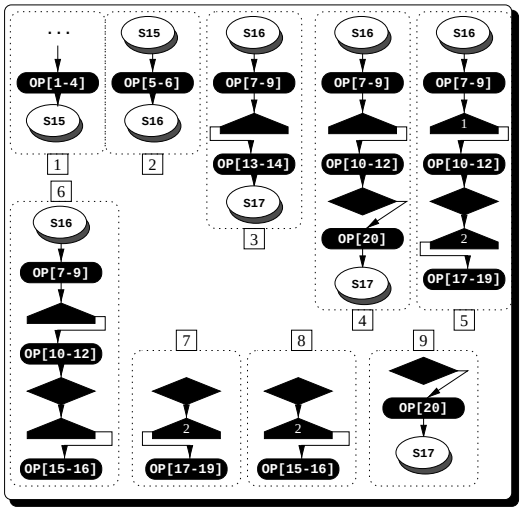
Les modes d'ordonnancement réalisés par notre algorithme sont illustrés par la figure 5.3. Les modes des entrées/sorties flottantes comprennent une limitation : les opérations ne sont jamais déplacées au-delà des limites marquées par les instructions de synchronisation. Par conséquent, le mode de cycle avec des entrées/sorties flottantes n'est pas autorisé. Ces modes d'ordonnancement peuvent être spécifiés pour un processus VHDL en sa totalité, ou spécifiés pour chaque instruction d'attente séparément avec des attributs VHDL. Cette combinaison des modes dans le même processus est une spécificité puissante de notre outil de synthèse parce qu'elle permet de combiner des styles d'écriture différents dans le même module. Ainsi la décomposition des systèmes complexes contenant des calculs et en même temps des structures de contrôle n'est-elle plus nécessaire.

5.5.1 Génération des chemins d'exécution

Les chemins d'exécution sont générés à partir du graphe de flux de contrôle. Le premier chemin débute avec le premier sommet du graphe, ensuite les successeurs sont ajoutés séquentiellement au chemin. Une spécificité importante du DLS est la génération des chemins d'exécution à la volée en effectuant une seule traversée du graphe de flux de contrôle. Cette caractéristique réduit considérablement la complexité et le temps d'exécution de l'algorithme pour les grandes applications. Pour la génération des chemins, un ensemble de contraintes sont établies. Si une de ces contraintes est transgressée, le chemin est considéré comme terminé est un nouveau chemin commence.



(a)



(b)

Figure 5.4: Génération des chemins d'exécution

L'algorithme du DLS est illustré par les figures 5.4 et 5.5. Ces figures montrent les étapes de l'ordonnancement à l'aide d'un fragment de l'exemple du circuit ATM. Cette partie du code comportemental est représentée par un graphe de flux de contrôle sur la figure 5.4(a). Les états initiaux proviennent des instructions d'attente (WAIT). Ce fragment démontre la liberté du style d'écriture permettant de combiner des constructions conditionnelles complexes et imbriquées.

L'algorithme initial du DLS a établi trois catégories de contraintes :

1. *Dépendances de données.*

   L'algorithme du DLS était intégré au sein du flux de conception d'AMICAL. L'algorithme d'ordonnancement n'étant pas étendu par une analyse de chemin de données et l'architecture générée par AMICAL ne pouvant contenir des opérations chaînées, le DLS initial décompose chaque chemin d'exécution si une dépendance de données se trouve entre le sommet courant et les sommets précédents dans le chemin. Pour ces dépendances de données, le DLS maintient deux listes, une pour les registres lus et une autre pour les registres déjà modifiés pendant le parcours du graphe. Une coupure est ainsi insérée si :

   $$Regs_{\acute{E}crits}(n.pr\acute{e}d\acute{e}cesseurs) \cup Regs_{Lus}(n) \neq 0$$

   ou

   $$Regs_{\acute{E}crits}(n.pr\acute{e}d\acute{e}cesseurs) \cup Regs_{\acute{E}crits}(n) \neq 0$$

   où $n.pr\acute{e}d\acute{e}cesseurs$ représente les prédécesseurs du sommet n dans le chemin courant.

   Cette approche a plusieurs inconvénients. Premièrement, les dépendances de données entre les instructions successives étant nombreuses dans les applications réelles, les chemins d'exécution sont trop fragmentés. Comme chaque coupure provoque un nouvel état dans la machine d'états finis, la taille de la machine d'états finis explose et le comportement du circuit change considérablement. Cet effet concerne surtout les parties des applications qui sont décrites avec exactitude du cycle d'horloge puisque les états supplémentaires altèrent le protocole précis. Le second inconvénient concerne au premier chef les calculs dans une description. La décomposition des expressions et la coupure aux dépendances de données introduisent un grande nombre de registres pour le stockage des valeurs intermédiaires.

   Dans notre approche, nous avons introduit une nouvelle étape d'ordonnancement qui sera présentée dans la section 5.5.4. Cette étape permet une analyse des chemins de données et le chaînage des opérations. Ainsi, la coupure aux dépendances de données est-elle inutile, la machine d'états finis devient plus compacte et moins de registres sont nécessaires. Par contre, les chemins d'exécution contenant plus d'opérations, le chemin critique de la machine d'états finis au

niveau transfert de registres devient plus long. C'est pour cette raison qu'un deuxième niveau d'ordonnancement sous contraintes de temps est nécessaire. En revanche, ce dernier doit être basé sur un graphe de flux de données représentant directement les dépendances de données entre les opérations, donc le résultat est plus efficace.

La coupure aux dépendances de données reste tout de même nécessaire dans les cas des accès des tableaux ou des champs de bits dont les indices ne sont pas exprimés par des expressions statiques. Cela est dû au fait que le chaînage des opérations s'effectue toujours entre des registres qui doivent être statiquement déterminés durant la synthèse.

2. *Contraintes de ressources.*

   L'algorithme initial du DLS décomposait également les chemins d'exécution sous contraintes de ressources. Pour chaque type de ressource, une fonction $Ress_{Utiliseé}(séquence, type)$ est définie. Elle permet de déterminer, pour une séquence d'opérations, le nombre de ressources utilisées de chaque type. Étant donné nb_r, le nombre de ressources disponibles de type r, une coupure est réalisée au niveau d'une opération n se servant d'une ressource de type r si et seulement si :

$$Ress_{Utilisée}(n.prédécesseurs) = nb_r$$

   Au sein d'AMICAL, seulement les accès de mémoire étaient considérés comme ressources durant la génération des chemins d'exécution. Bien que plusieurs ordonnancements basés sur un graphe de flux de contrôle coupent les chemins d'exécution sous contraintes de ressources, les coupures effectuées pendant la génération des chemins d'exécution restent une source importante de l'inefficacité pour des applications contentant des calculs. Les arcs du graphe de flux de contrôle ne représentant que l'ordre de l'exécution des opérations sans tenir compte de flux de données, la décomposition des pas d'exécution en cycles de contrôle reste généralement peu optimale. Par conséquent, les coupures sous contrainte de ressources doivent être faites dans une deuxième phase d'ordonnancement sur les transitions générées.

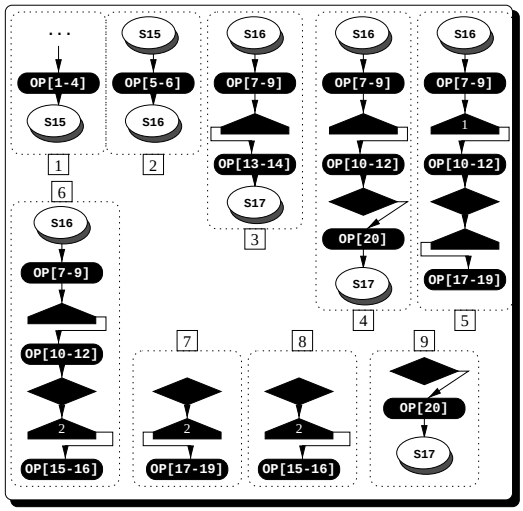
3. *Contraintes spécifiques aux boucles.*

   La plupart des algorithmes basés sur la recherche des chemins d'exécution introduisent un état supplémentaire au début de chaque instruction itérative. Ce fait est lié au traitement des boucles qui consiste à couper avant l'ordonnancement les chemins rebouclants à la tête. Cette

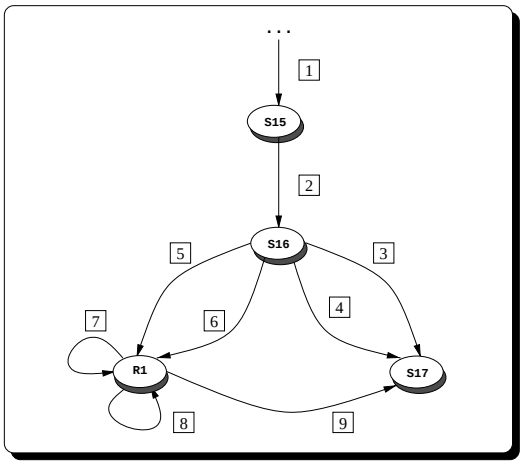
technique a comme inconvénient la perte des cycles en entrant et en sortant de chaque boucle. Cela prend une importance plus considérable lorsque plusieurs boucles sont imbriquées puisque la perte de cycles est factorisée. Coupant les chemins d'exécution à la volée, le DLS assure une approche plus efficace. Comme les chemins rebouclant à la tête de la boucle ne sont pas éliminés et le sommet de boucle n'est pas considéré comme un sommet spécial, la première et la dernière itération sont déroulées sans aucune perte de cycle de contrôle. Le style d'écriture n'imposant aucune restriction pour les actions du corps des boucles, les boucles sans instructions d'attente sont admises. Il en résulte des chemins d'exécution qui contiennent la tête de la boucle et une séquence d'instructions de l'intérieur de la boucle. En suivant les règles de la génération des chemins d'exécution, comme la condition dépend de l'exécution, le chemin d'exécution deviendrait infini. Pour éviter ce problème, une nouvelle contrainte spéciale a été introduite : si le successeur du dernier sommet du chemin d'exécution courant se trouve déjà dans le même chemin d'exécution, le chemin est considéré comme terminé.

La génération des chemins d'exécution est illustrée par la figure 5.4. Ce fragment de la description contient trois états systèmes **S15**, **S16** et **S17**, 20 opérations simples regroupées en blocs de base et trois instructions conditionnelles dont l'une est une boucle WHILE. En partant du graphe de flux de contrôle, le DLS génère 9 chemins d'exécution dont deux, avant l'état **S15** et entre l'états **S15** et **S16**, sont évidents. Les autres chemins d'exécution, partant de l'état **S16**, suivent les branchements des instructions conditionnelles. Les chemins d'exécution **7**, **8** et **9** sont introduits par la transgression de la contrainte spécifique aux boucles.

5.5.2 Génération de la machine d'états finis



(a)



(b)

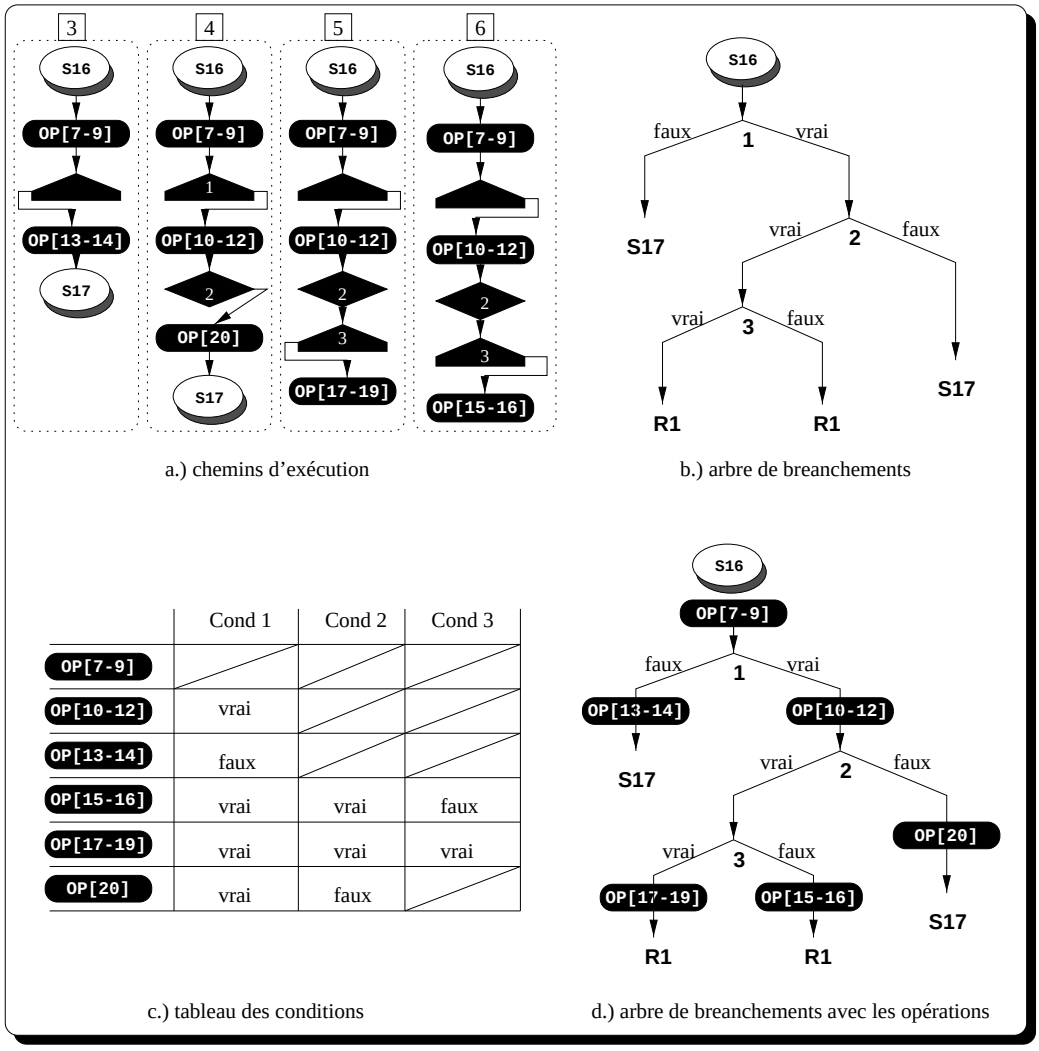
Figure 5.5: Génération de la machine d'états finis

À chaque chemin d'exécution ordonné correspond une transition dans la machine d'états finis résultante. La génération de cette machine d'états finis est relativement simple : tous les chemins d'exécution commençant par le même sommet auront le même état de départ. À chaque transition, un état suivant est associé qui est l'état désigné par le sommet succédant au dernier sommet de la transition en question. Nous pouvons ainsi conclure que le nombre d'état de la machine d'états finis résultante correspond au nombre d'instructions d'attente de la description initiale et du nombre de transgressions des contraintes.

La génération de la machine d'états finis est illustrée par la figure 5.5(b) qui reprend l'exemple présenté dans la section précédente. Les états initiaux dus aux instructions WAIT sont marqués de la même manière. Un état supplémentaire **R1** a été inséré qui réalise le bouclage de l'instruction WHILE. Les transitions **5** et **6** sont le déroulement de la première itération de la boucle alors que **3** et **4** évitent l'exécution de la boucle suivant les conditions et assurent un passage direct entre les états **S16** et **S17**. Le corps de la boucle est exécuté par les transition **7** et **8** qui rebouclent toujours au même état (**R1**) jusqu'à ce que la condition de la boucle WHILE devienne fausse et l'exécution se poursuit donc à travers la transition **9**.

5.5.3 Réécriture de code

Les transitions de la machine d'états finis construite à partir des chemins d'exécution ne contiennent que des opérations simples : affectations, instructions de branchement conditionnel (IF et CASE). Suivant la nature d'une machine d'états finis, il est assuré que chaque transition mène à un autre état. Les sommets complexes du graphe de flux de contrôle tels que les instructions d'attente et les boucles existent encore dans ces transitions, mais ils sont transformés de façon à ce que pendant l'étape de réécriture de code, ils puissent être remplacés par des opérations de branchement conditionnel simples.



	Cond 1	Cond 2	Cond 3
OP[7-9]			
OP[10-12]	vrai		
OP[13-14]	faux		
OP[15-16]	vrai	vrai	faux
OP[17-19]	vrai	vrai	vrai
OP[20]	vrai	faux	

Figure 5.6: Réécriture du code pour les transitions

Dans l'approche traditionnelle, les instructions conditionnelles sont séparées des opérations simples et extraites au début de chaque transition. En partant du même état, toutes les combinaisons de ces conditions doivent apparaître. Le résultat de cette transformation est une séquence des *if-elsif-...-else*. Pour les outils de synthèse comportementale de deuxième génération, cette approche est nécessaire pour l'étape de génération de l'architecture. En général, les conditions sont exécutées dans le contrôleur alors que les autres opérations sont réalisées par le chemin de données. Par contre une telle séparation convient peu à la nouvelle approche de la synthèse comportementale :

- Le style d'écriture de la séquence des *if-elsif-...-else* ne convient pas à la synthèse au niveau transfert de registres. Pour que toutes les combinaison de chaque condition soient exprimées, il faut dupliquer les opérations de comparaison. La synthèse logique n'étant pas capable de détecter ces duplications, le partage des comparateurs ne devient pas possible, ce qui augmente considérablement la surface du circuit généré.

- Cette séparation des conditions de la partie opérative engendre la coupure aux dépendances de données entre une condition et une opération précédant la condition, ce qui augmente le nombre d'étapes de contrôle et change le comportement du circuit par rapport à la description comportementale.

Pour éviter ces problèmes, un nouveau style d'écriture a été développé pour les transitions de la machine d'états finis : l'ensemble des transitions partant du même état est décrit à l'aide d'un arbre d'opérations ou les branchements correspondent aux instructions d'attente, aux têtes des boucles et aux instructions conditionnelles, les branches contiennent les blocs de base du graphe de flux de contrôle et les feuilles conduisent à l'état suivant. Le fait que par définition, chaque transition commence par le même sommet et que l'ordre des opérations suive celui de la description initiale rend possible cette transformation. L'avantage de cette approche est double : d'une part, les descriptions générées au niveau transfert de registres sont mieux adaptées au style d'écriture de la synthèse logique, d'autre part cette réécriture assure la correspondance forte entre les descriptions initiales et celles générées. L'algorithme est illustré par la figure 5.6 qui reprend une partie de l'exemple utilisé pour la présentation de l'ordonnancement dans les sections précédentes. L'exemple montre les quatre transitions partant de l'état **S16** : deux d'entre elle aboutissent à l'état **S17** ; deux autres à l'état **R1** introduit par l'ordonnancement pour réaliser la boucle WHILE. Les transitions sont composées d'opérations conditionnelles (une boucles et deux IF)

et de blocs de base contenant des affectations. La transformation s'effectue en deux étapes :

- Construction des branchements de l'arbre d'opérations. Ces branchements correspondent aux instructions conditionnelles et celles d'attente du graphe de flux de contrôle :

  boucles : Les retours des boucles étant coupés durant l'ordonnancement, cette opération peut être remplacée par un simple IF. La branche vraie correspond au corps de la boucle alors que la branche fausse mène à l'instruction suivante. Une spécificité de cet IF est que les deux branches ne peuvent plus jamais se rejoindre dans la même transition.

  instructions d'attente : Si l'état courant correspond à une instruction d'attente de la description initiale, le premier sommet est toujours une instruction WAIT. Ce sommet est transformé en un IF dont la branche vraie sort de l'état d'attente en exécutant les opérations qui se succèdent alors que la branche fausse reboucle au même état.

  instructions conditionnelles : Les instructions conditionnelles sont gardées dans leur propre forme. Il faut noter que les branches de ces instructions peuvent encore se rejoindre.

  Dans notre exemple (figure 5.6.(b)), il existe trois branchements dont le premier et le dernier correspondent aux IFs, le deuxième provient de la boucle WHILE.

- Les branches de l'arbre d'opérations sont ensuite remplies par des opérations en suivant l'ordre de leur exécution et en analysant leurs conditions d'exécution. Cette étape de transformation s'effectue à l'aide d'un tableau qui associe à chaque bloc de base les conditions d'exécution. Cette étape est illustrée sur la figure 5.6.(c). Les cases contenant des valeurs booléennes (*vrai* et *faux*) désignent les branches correspondantes de l'instruction IF. Les cases rayées indiquent que le bloc de base en question n'appartient pas à l'instruction conditionnelle. Dans notre exemple, le résultat de la réécriture du code est présenté sur la figure 5.6.(d) où les blocs de base sont placés sur les branches.

Les arbres d'opérations générés de cette façon reflètent précisement l'ordre d'exécution des opérations de la description initiale. À partir de cette représentation, la génération du code pour les transitions de la machine d'états finis

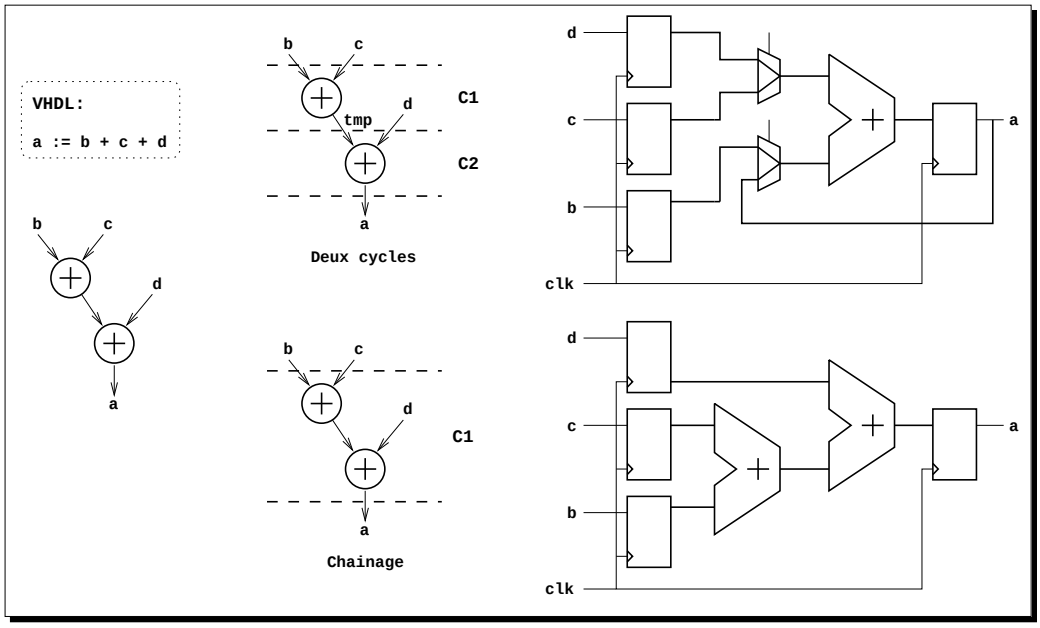


Figure 5.7: Chaînage des opérations en cas de dépendance de données

finale peut être effectuée facilement et le style d'écriture rend la description au niveau transfert de registres efficace du point de vue du partage des ressources.

5.5.4 Analyse de chemin de données : chaînage

Durant la simulation, les opérations de la machine d'états finis au niveau transfert de registres sont exécutées séquentiellement. Les variables déclarées dans le processus de la description initiale peuvent nécessiter une mémorisation, ce qui s'effectue par leur transformation en paires de signaux. Les affectations de variables deviennent donc des affectations de signaux. Ainsi la mise à jour des nouvelles valeurs ne s'effectue-elle qu'à la fin de chaque cycle de simulation. Ce fait provoque des problèmes en cas de dépendances de données parce que pendant l'exécution, les valeurs intermédiaires ne sont pas prises en considération.

Il est donc nécessaire de séquentialiser les opérations aux dépendances de données. Cette technique est appelée le chaînage des opérations qui est un traitement typiquement orienté flux de données. Nous pouvons distinguer deux types de chaînage : chaînage de données (entre deux opérations), illustré par la figure 5.7 et celui de contrôle (entre une opération et une condition),

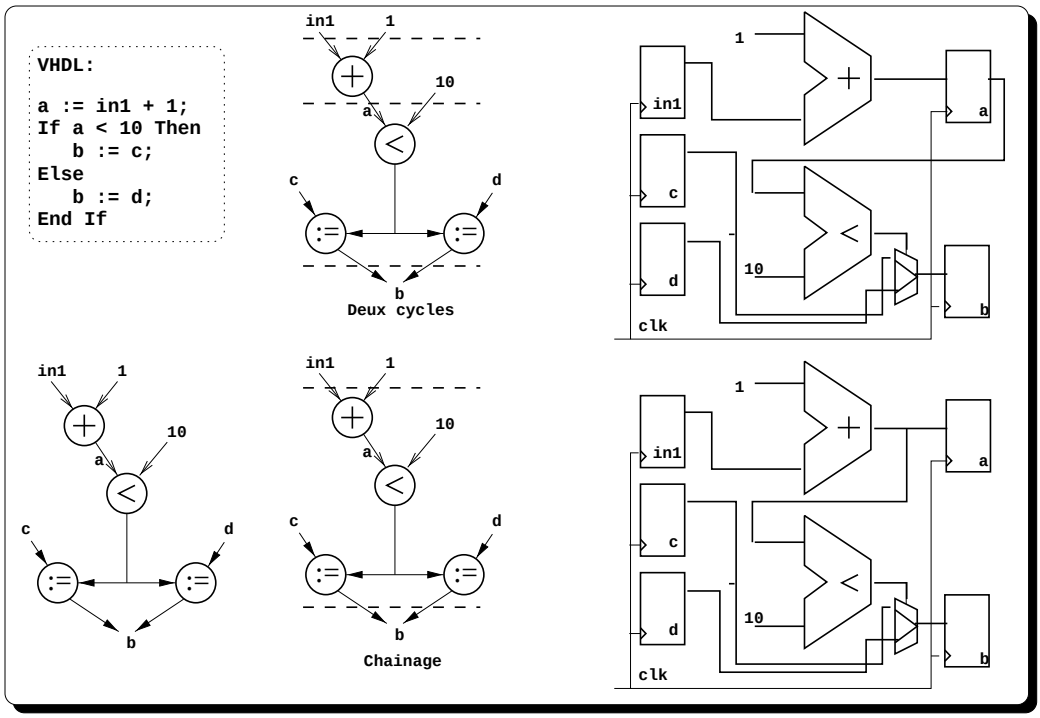
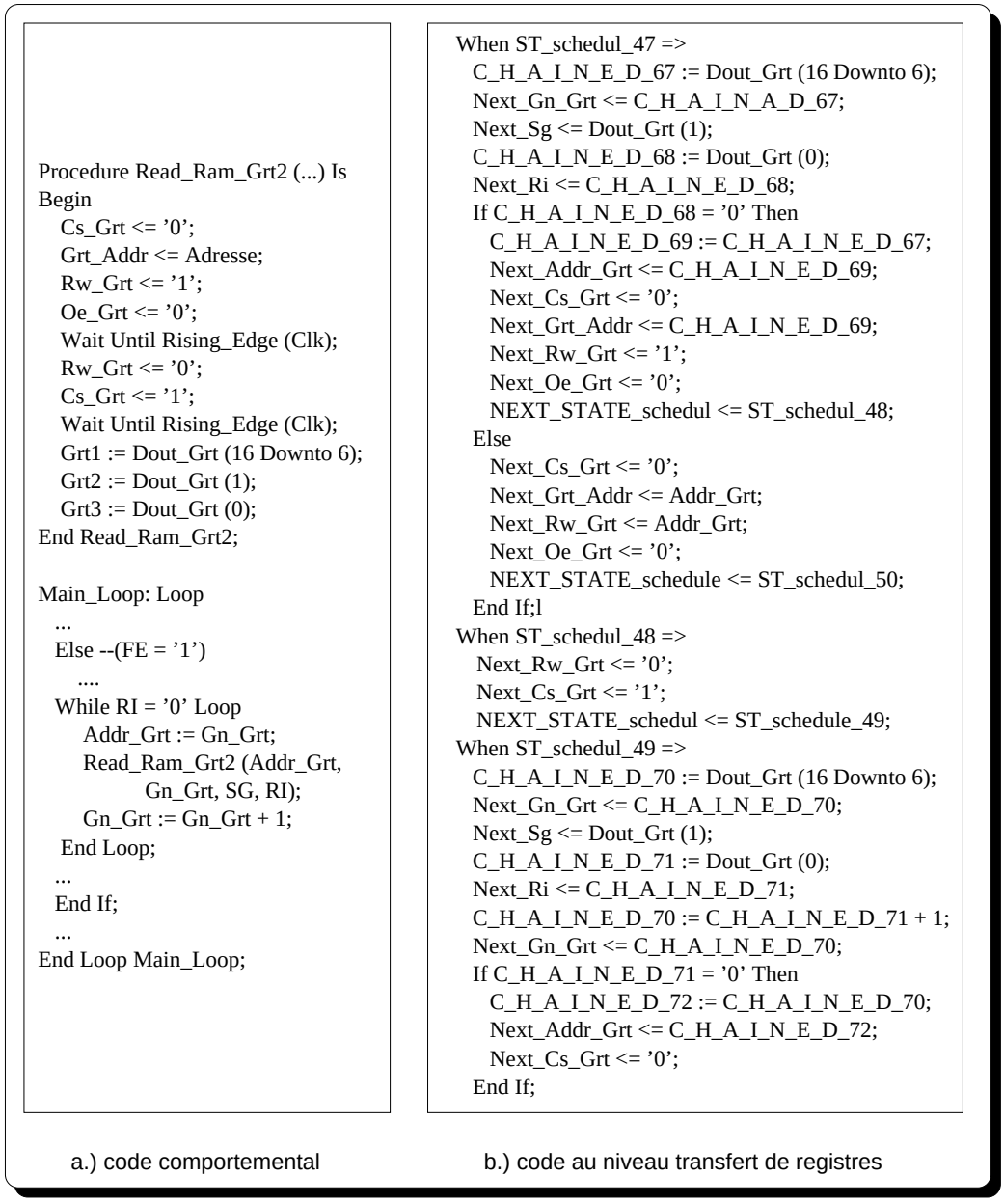


Figure 5.8: Chaînage des opérations en cas de dépendance de contrôle

montré sur la figure 5.8. En général, la réalisation de ce dernier est une tâche plus complexes car dans l'architecture générée, les opérations se trouvent dans la partie opérative alors que l'évaluation des conditions s'effectue dans la partie de contrôle.

Pour les algorithmes d'ordonnancement reposant sur un graphe de flux de contrôle comme représentation intermédiaire, la réalisation du chaînage est plus problématique. Ce fait est dû à la caractéristique principale de tels graphes : les opérations sont représentées en fonction de l'ordre d'exécution sans informations sur les dépendances de données. En général, les chemins d'exécution sont coupés à chaque dépendance de données, ce qui augmente le nombre d'étapes de contrôle dans la machine d'états finis résultante. Dans [BRNT98], une technique d'optimisation est proposée. Elle consiste à changer l'ordre des opérations dans les chemins d'exécution afin de diminuer le nombre de dépendances de données. Quoique cette technique puisse alléger le problème, en réalité son efficacité reste limitée.

Pour pouvoir effectuer l'analyse de dépendances de données, nous avons étendu le concept du graphe de flux de contrôle par la représentation des dépendances de données. Notre algorithme de chaînage s'applique aux tran-



```
Procedure Read_Ram_Grt2 (...) Is
Begin
   Cs_Grt <= '0';
   Grt_Addr <= Adresse;
   Rw_Grt <= '1';
   Oe_Grt <= '0';
   Wait Until Rising_Edge (Clk);
   Rw_Grt <= '0';
   Cs_Grt <= '1';
   Wait Until Rising_Edge (Clk);
   Grt1 := Dout_Grt (16 Downto 6);
   Grt2 := Dout_Grt (1);
   Grt3 := Dout_Grt (0);
End Read_Ram_Grt2;

Main_Loop: Loop
  ...
  Else --(FE = '1')
     ....
  While RI = '0' Loop
      Addr_Grt := Gn_Grt;
      Read_Ram_Grt2 (Addr_Grt,
            Gn_Grt, SG, RI);
      Gn_Grt := Gn_Grt + 1;
   End Loop;
  ...
  End If;
  ...
End Loop Main_Loop;
```

a.) code comportemental

```
When ST_schedul_47 =>
  C_H_A_I_N_E_D_67 := Dout_Grt (16 Downto 6);
  Next_Gn_Grt <= C_H_A_I_N_A_D_67;
  Next_Sg <= Dout_Grt (1);
  C_H_A_I_N_E_D_68 := Dout_Grt (0);
  Next_Ri <= C_H_A_I_N_E_D_68;
  If C_H_A_I_N_E_D_68 = '0' Then
    C_H_A_I_N_E_D_69 := C_H_A_I_N_E_D_67;
    Next_Addr_Grt <= C_H_A_I_N_E_D_69;
    Next_Cs_Grt <= '0';
    Next_Grt_Addr <= C_H_A_I_N_E_D_69;
    Next_Rw_Grt <= '1';
    Next_Oe_Grt <= '0';
    NEXT_STATE_schedul <= ST_schedul_48;
  Else
    Next_Cs_Grt <= '0';
    Next_Grt_Addr <= Addr_Grt;
    Next_Rw_Grt <= Addr_Grt;
    Next_Oe_Grt <= '0';
    NEXT_STATE_schedule <= ST_schedul_50;
  End If;l
When ST_schedul_48 =>
   Next_Rw_Grt <= '0';
   Next_Cs_Grt <= '1';
   NEXT_STATE_schedul <= ST_schedule_49;
When ST_schedul_49 =>
  C_H_A_I_N_E_D_70 := Dout_Grt (16 Downto 6);
  Next_Gn_Grt <= C_H_A_I_N_E_D_70;
  Next_Sg <= Dout_Grt (1);
  C_H_A_I_N_E_D_71 := Dout_Grt (0);
  Next_Ri <= C_H_A_I_N_E_D_71;
  C_H_A_I_N_E_D_70 := C_H_A_I_N_E_D_71 + 1;
  Next_Gn_Grt <= C_H_A_I_N_E_D_70;
  If C_H_A_I_N_E_D_71 = '0' Then
    C_H_A_I_N_E_D_72 := C_H_A_I_N_E_D_70;
    Next_Addr_Grt <= C_H_A_I_N_E_D_72;
    Next_Cs_Grt <= '0';
  End If;
```

b.) code au niveau transfert de registres

Figure 5.9: Exemple pour le chaînage

sitions de la machine d'états finis générées par l'étape de réécriture du code. Comme la description générée au niveau transfert de registres ne sépare pas le chemin de données et le contrôleur, la différence entre le chaînage de contrôle et celui de données n'est que théorique. Pour notre style d'écriture, le chaînage est représenté par des variables qui permettent de sauvegarder les valeurs intermédiaires et de séquentialiser les opérations en question. Ces variables n'étant utilisées que dans une seule transition, la synthèse au niveau transfert de registres les élimine et les réalise avec des fils de connexion.

Une différence importante existe entre la technique de chaînage traditionnelle et notre approche : pour les ordonnancements orientés flux de données, le chaînage est intégré à l'algorithme d'ordonnancement sous contraintes de temps alors que notre approche sépare les deux algorithmes en effectuant un chaînage maximal, autrement dit en supposant un cycle d'horloge infini. Cette séparation laisse la possibilité pour qu'un deuxième niveau d'ordonnancement sous contraintes de temps puisse être effectué.

Le résultat du chaînage est illustré par la figure 5.9. Le modèle comportemental et la partie correspondante de la machine d'états finis sont montrés. La description comportementale contient une boucle conditionnelle qui fait un appel à une procédure réalisant un protocole d'accès à une mémoire. Cet appel de procédure est mis à plat durant l'ordonnancement. Dans la description au niveau transfert de registres, les variables intermédiaires nommées *C_H_A_I_N_E_D_XX* représentent les dépendances de données et effectuent le chaînage. Par exemple, la variable *C_H_A_I_N_E_D_67* correspond à la variable comportementale *Next_Gn_Grt*. Dans notre exemple, cette variable réalise un chaînage de données. Par contre, la variable *C_H_A_I_N_E_D_68* correspondant à *Next_Ri* est un exemple du chaînage de contrôle car la valeur intermédiaire est utilisée dans la condition de l'instruction IF. Ces variables intermédiaires sont générées de façon à ce que la synthèse au niveau transfert de registres puisse les éliminer.

5.6 Conclusions

Ce chapitre décrit notre nouvelle approche pour la synthèse comportementale qui ne s'appuie que sur l'étape d'ordonnancement. La description au niveau transfert de registres est générée à partir de la machine d'états finis produite par l'ordonnancement. Le style d'écriture de cette description respecte entièrement les règles de codage exigées par les outils de synthèse au niveau transfert de registres.

Le modèle d'entrée a été présenté dans ce chapitre qui assure un style d'écriture au niveau comportemental beaucoup plus libre que ce que la plu-

part des outils de synthèse comportementale classiques acceptent. Le modèle d'entrée permet la description des applications complexes et hiérarchiques avec la combinaison des structures au niveau comportemental et au niveau transfert de registres.

Les étapes de synthèse de notre approche ont été également détaillées. Une phase d'élaboration complexe précède l'ordonnancement, ce qui permet la synthèse des descriptions hiérarchiques et paramètrées. L'ordonnancement, basé sur l'algorithme *ordonnancement à boucles dynamiques*, utilise un graphe de flux de contrôle comme représentation intermédiaire. Afin d'étendre les domaines d'application vers les conceptions mixtes (orientées flux de contrôle et flux de données), une analyse de chemin de données a été intégrée à l'approche d'ordonnancement initiale. Cette étape permet le chaînage de données et de contrôle en cas de dépendances de données, ce qui donne des résultats plus efficaces pour la plupart des applications. Une phase de réécriture du code a été développée qui adapte la machine d'états finis générée par l'ordonnancement au style d'écriture de la synthèse au niveau transfert de registres.

L'efficacité de la nouvelle approche est prouvée par plusieurs applications industrielles et par l'intégration de notre flux de synthèse comportementale à un flux de synthèse système. La présentation de ces résultats fait partie du chapitre suivant.

Chapitre 6

Application de la synthèse comportementale

Introduction

Ce chapitre décrit l'application du flux de la synthèse comportementale basé sur l'ordonnancement. Deux aspects sont abordés : le premier concerne l'application du flux de synthèse sur des conceptions complexes. Le second présente l'intégration du nouvel outil de synthèse à un flux de conception système.

La section 6.1.1 présente une application de 2 millions de transistors dont la synthèse a été effectuée à l'aide de notre outil de synthèse. Le fait que le même circuit ait été conçu de trois façons (à l'aide de deux outils différents de synthèse comportementale et à partir de descriptions manuellement décrites au niveau transfert de registres) donne de l'importance à cette application. Les résultats sont présentés comparativement, ce qui permet une évaluation plus approfondie de notre nouvelle approche. Une autre application industrielle, bien que moins importante que la précédente, est présentée dans la section 6.1.2. Ce circuit souligne l'importance de la combinaison des structures au niveau transfert de registres avec un style d'écriture purement comportemental.

La dernière section de ce chapitre donne un aperçu de l'intégration des outils de synthèse comportementale à un flux de synthèse système.

6.1 Exemples d'application

Les deux applications présentées dans cette thèse ont été conçues avec la collaboration des partenaires industriels. La première application provient

du domaine des télécommunications et donne un exemple de l'utilisation de la synthèse comportementale pour la conception d'un circuit très complexe. La deuxième application appartient au domaine du traitement d'image avec un style d'écriture mixte : comportemental et transfert de registres.

6.1.1 La conception d'un contrôle trafic ATM

Les circuits ATM (mode de transfert asynchrone[1]) ont une période de vie très courte sur le marché. Ceci est dû à l'évolution constante des standards d'ATM. En outre, la complexité de ces circuits s'accroît exponentiellement en raison de diverses demandes d'intégration de cette nouvelle technologie. Le succès de cette technologie dépend de la possibilité de satisfaire la contrainte de temps stricte de mise sur la marché (TTM[2]). Ainsi l'introduction de nouvelles méthodologies de conception plus productives est-elle primordiale. La méthodologie de conception basée sur la synthèse au niveau transfert de registres étant lente, un plus haut niveau d'abstraction de la conception est donc nécessaire. Par conséquent, l'usage de la synthèse comportementale pour la génération automatique des descriptions au niveau transfert de registres s'avère une solution efficace. L'avantage principal de cette approche est l'augmentation de la qualité de conception et la réduction des cycles de conception.

À l'aide de notre outil de synthèse, un circuit réalisant une gestion de trafic ATM a été développé, ce qui a permis l'évaluation de la méthodologie présentée tout au long de cette thèse. La complexité de cet exemple et le fait que le même circuit ait été conçu à partir d'une spécification au niveau transfert de registres rendent cette application encore plus précieuse pour étudier le flux de conception de la synthèse comportementale.

6.1.1.1 Architecture du circuit d'ATM

Le contrôleur de trafic ATM est un composant essentiel pour satisfaire la qualité de service et pour empêcher les congestions en même temps. La principale fonctionnalité du circuit est d'assurer un flux équilibré pour le transfert. Les transferts de blocs sont mémorisés et retardés dans le temps. Le trafic est transmis par priorités suivant le mode de trafic et avec insertion des périodes de silence. La méthode réduit les exigences des bandes de fréquences pour les sources aussi bien que la probabilité des conditions de congestion dans les commutateurs qui sont dues aux flux de blocs. Le transfert de données de l'utilisateur vers le réseau est contrôlé par le mécanisme de régulation

[1]En anglais : ATM : Asynchronous Transfert Mode
[2]En anglais : TTM : time-to-market.

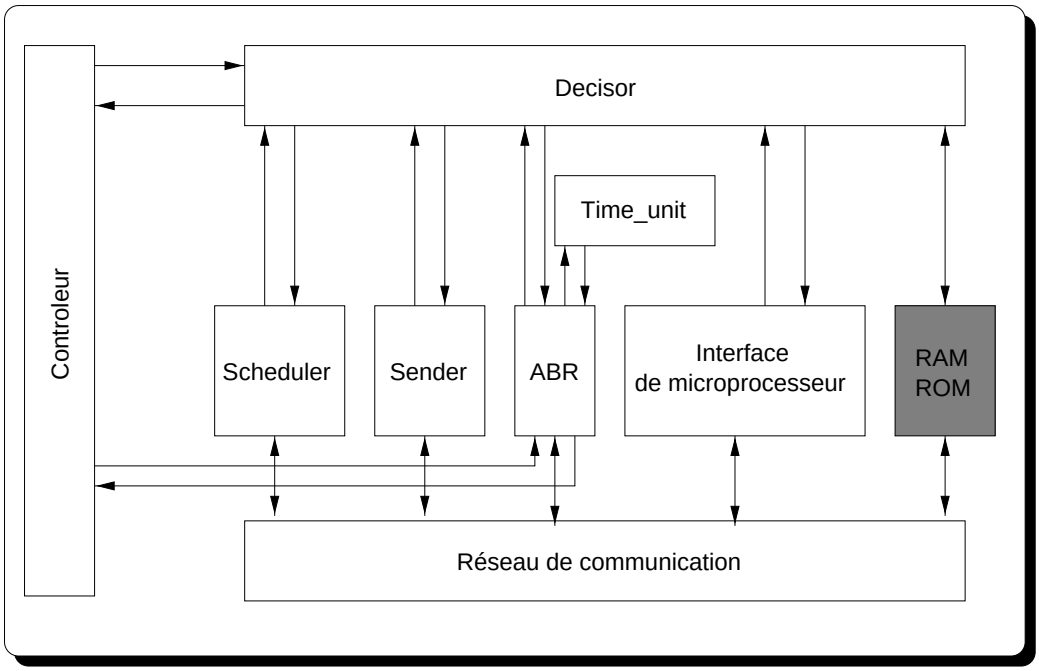


Figure 6.1: Architecture du circuit réalisant une gestion de trafic ATM

implanté dans l'adaptateur de terminal. Ce mécanisme influence le trafic suivant les caractéristiques du flux approprié. Les caractéristiques principales de la gestion de trafic sont les suivantes : les bandes de fréquences sont de 155 Mbits/sec permettant tout type de trafic tels que *Variable Bit Rate* (VBR), *Constant Bit Rate* (CBR), *Unspecified Bit Rate* (UBR) et *Available Bit Rate* (ABR) [Com96] et 4K de connexions peuvent être gérées simultanément. La principale contrainte temporaire impose que le traitement d'une cellule ATM ne peut excéder 106 cycles d'horloge à la fréquence d'opération de 40 MHz. La fonctionnalité et la structure de l'application sont détaillées dans [MSS+99].

Pour la réalisation d'une spécification à l'aide d'un outil de synthèse comportementale, un système complexe doit être partitionné en des sous-systèmes ou en des modules afin de pouvoir en réduire sa complexité. Ainsi chaque module peut-il être décrit et synthétisé efficacement [SM98a].

Le partitionnement du système s'est effectué à partir de la fonctionnalité de l'architecture de la gestion de trafic et des contraintes temporaires présentées ci-dessus. Pour satisfaire toutes ces contraintes, il a été nécessaire d'introduire du parallélisme dans l'architecture. Cette architecture est illustrée par la figure 6.1. Elle est composée de 8 modules principaux dont seul le

bloc de mémoire n'a pas été synthétisé par notre outil. Du point de vue des contraintes temporaires, la fonction ABR était la plus délicate à réaliser. Afin de respecter les délais, elle a été décomposée en deux sous-blocs : Decisor et ABR comme présenté sur la figure. Cette décomposition permet l'exécution concurrente des sous-fonctions. En outre, ABR peut être exécuté en parallèle avec Scheduler dans certaines configurations. La concurrence entre ABR et Descisor est gérée par le contrôleur alors que le parallélisme entre ABR et Scheduler est contrôlé par Descior.

6.1.1.2 Style d'écriture des descriptions comportementales

Chaque module du système est décrit par un seul processus VHDL. La nature de l'application étant principalement orientée flux de contrôle, ces processus sont composés d'un grand nombre d'instructions conditionnelles et d'opérations pour l'écriture et la lecture des entrées/sorties. La difficulté principale était la description des proccesus combinant des protocoles de communication (poignées de main) des algorithmes de contrôle (instructions conditionnelles et boucles) et des calculs nécessaires pour certaines fonctionnalités. La spécification complète du circuit contient 3200 lignes de code comportemental en VHDL.

```
Procedure Read_Ram_Nrt (...) Is
Begin
    Cs_grt <= '0';
    Grt_addr <= Address;
    rw_grt <= '1'; Oe_grt <= '0';
    Wait Until Rising_Edge (Clk);
    Rw_grt <= '0'; Oe_grt <= '1';
    Wait Until Rising_Edge (Clk);
    Grt1 := Dout_grt (16 Downto 6);
    Grt2 := Dout_grt (1);
    Grt3 := Dout_grt (0);
End Read_Ram_Nrt;
...
Main_Loop: Loop
...
   Read_Ram_Nrt (Addr_nrt, ...);
   If DC = 16#000# Then
      Aux_pt := Nrt_pt;
   Else
      Addr_nrt := Nrt_pt;
      Addr_grt := Gn_nrt;
      Aux_pt := FFL;
      Loop_1: While DC /= 16#000# Loop
         ...
         If Fe_nrt = '0' Then
            Aux_pt := Aux_pt + 1;
            Addr_nrt := Addr_nrt + 1;
         Else
            Aux_pt := Addr_nrt;
         End If;
         DC := DC - 1;
      End Loop Loop_1;
   End If;
   DC := DC + 1;
   Wait Until Rising_Edge (Clk);
   ...
End Loop Main_Loop;
```

Figure 6.2: Style d'écriture du code comportemental

Le style d'écriture utilisé exploite entièrement la liberté assurée par notre outil de synthèse. La figure 6.2 illustre ce style de codage. La combinaison des appels de sous-programmes, des instructions d'attente et des instructions conditionnelles permet une description compacte de protocoles précis nécessaires pour l'échange de données entre les modules et la mémoires. Ce style d'écriture est détaillé dans le chapitre 5.

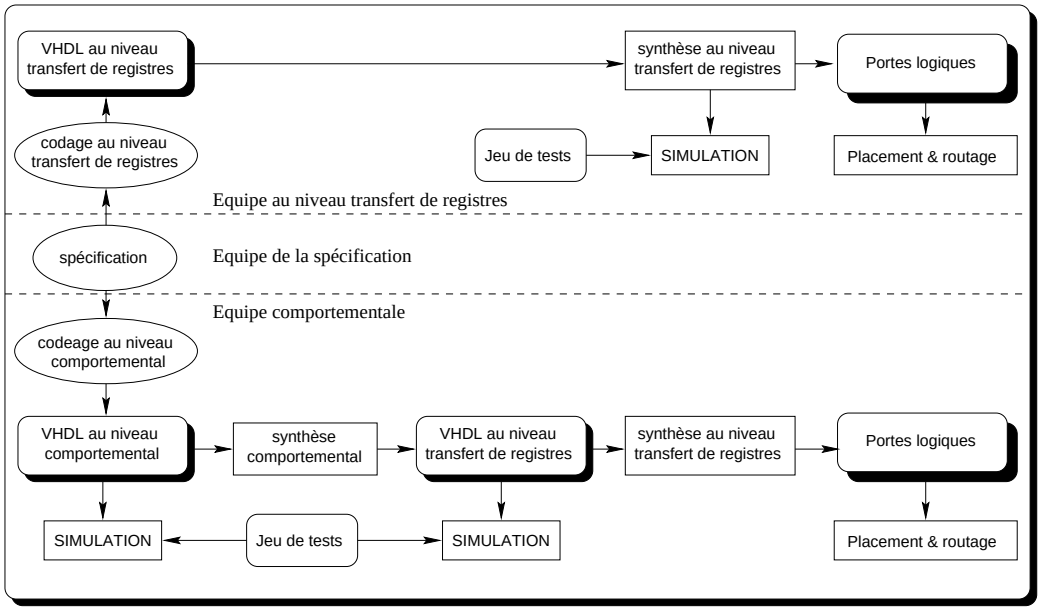


Figure 6.3: Flux de conception utilisé pour le circuit ATM

6.1.1.3 Résultats de la synthèse

La réalisation de l'application a impliqué la collaboration de trois équipes. L'équipe de conception système a été chargée de réaliser la spécification du circuit. La seconde équipe a suit le flux classique de synthèse : codage de la spécification directement au niveau transfert de registres et ensuite l'utilisation d'un outil de synthèse pour générer l'architecture au niveau des portes. La dernière équipe a utilisé notre outil de synthèse comportementale : codage au niveau comportemental, l'utilisation de l'outil de synthèse comportementale et finalement, le flux de synthèse classique. La figure 6.3 illustre les flux de conception utilisés par chaque équipe.

Cette collaboration a permis de comparer la méthodologie de la synthèse comportementale avec celle de la synthèse au niveau transfert de registres. De ce point de vue, les résultats sont illustrées par le tableau 6.4. Cette comparaison considère le nombre de lignes de code VHDL, le nombre de portes logiques et le durée du cycle de conception. La taille de la description comportementale est de 50 % de moins que celle au niveau transfert de registres, ce qui permet une gestion plus facile du code et des modifications éventuellement nécessaires au niveau comportemental. Le résultat le plus surprenant provient de la taille du code généré au niveau transfert de registres qui est moins grand que le code décrit manuellement par l'équipe suivant la méthodologie classique. L'analyse démontre que ce résultat est dû au parti-

Description	Conception comportementale	Conception au niveau transfert de registres
VHDL comportemental	3200 lignes	
VHDL transfert de registres	9311 lignes	10250
Nombre de portes logique	28500 portes	27000 portes
Nombre de modules	7 modules	15 modules
Effort de conception (personnes/mois)	8	25

Figure 6.4: Comparaison des méthodologies de conception

tionnement du système. Les concepteurs utilisant uniquement la synthèse au niveau transfert de registres ont décomposé l'application en un grand nombre de modules (15 modules). L'autre équipe n'a utilisé que 7 blocs. Dans tous les deux cas le partitionnement a été influencé par la complexité des blocs à réaliser. En réalité, le traitement d'un processus VHDL devient très difficile s'il contient plus de quelques centaines de lignes de code. Cette taille correspond à une machine d'états finis avec quelques dizaines d'états. Par exemple, le module Decisor est décrit par un seul processus VHDL composé de 850 lignes de code comportemental. Le même bloc est décrit par plusieurs modules au niveau transfert de registres. L'analyse démontre également que le code généré est aussi efficace que celui qui est produit manuellement au niveau transfert de registres par les concepteurs de l'autre équipe.

En matière de nombre de portes logiques, la conception au niveau transfert de registres ne diffère guère de celle produite par l'outil de synthèse comportementale. Le tableau montre également que l'effort nécessaire pour la réalisation du circuit à l'aide de la méthodologie de la synthèse comportementale est 3 fois inférieur à ce que l'on obtient en n'utilisant que la synthèse au niveau transfert de registres. Cette caractéristique a une grande importance puisque la rapidité est la clé pour le marché des applications des télécommunications.

La figure 6.5 montre le flux de conception et de validation utilisé pour l'application ainsi que la boucle de conception. À partir de la spécification initiale, le circuit est manuellement partitionné et décrit au niveau comportemental. Le modèle est simulé (validation B) pour vérifier la fonctionnalité globale. La boucle B1 permet de modifier les descriptions comportementales en cas d'erreur dans le codage.

À partir du niveau comportemental, une architecture au niveau transfert

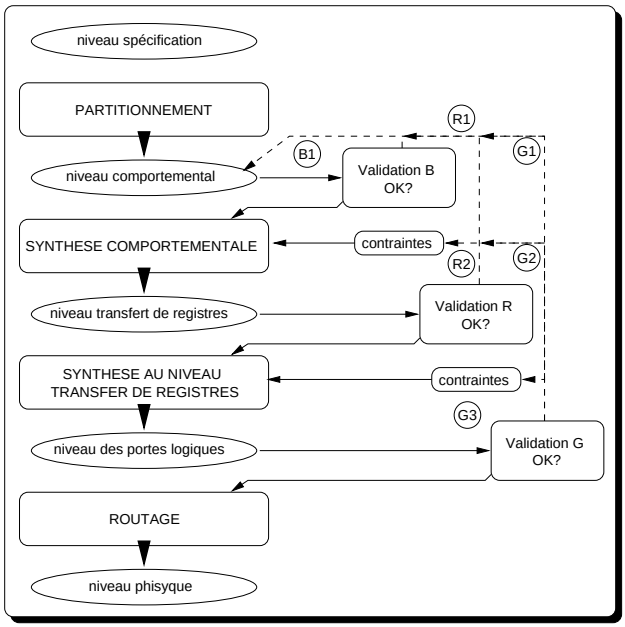


Figure 6.5: Cycle de validation pour le circuit ATM

de registres est générée automatiquement par notre outil de synthèse comportementale. Cette description est simulée (validation R) pour valider le comportement au niveau de cycle d'horloge. La boucle R1 est utilisée pour vérifier la description comportementale du point de vue du sous-ensemble synthétisable du langage et du style d'écriture accepté par l'outil de synthèse. La boucle de validation R2 est utilisée pour mettre des contraintes à l'application pour la synthèse. Dans le cas de conception manuelle, cette boucle sert à valider la description au niveau transfert de registres.

À partir de la description au niveau transfert de registres, l'architecture du circuit est générée au niveau des portes logiques et optimisée à l'aide d'un outil de synthèse au niveau transfert de registres et logique avec des contraintes de ressources et temporaires. Cette architecture est simulée (validation G) pour vérifier l'architecture du point de vue des délais. La boucle G1 permet de détecter si dans la description comportementale, des variables et/ou des signaux ne sont pas initialisés, ce qui n'est pas détectable durant la validation R.

La complexité des descriptions comportementales étant moins élevée, la simulation à ce niveau d'abstraction (validation B) peut être effectuée qu'au niveau transfert de registres (validation R), ce qui prend encore moins de temps que la simulation au niveau des portes logiques. L'avantage de la synthèse comportementale se manifeste donc durant la validation des systèmes

Module	Lignes de code comportemental	Synthèse comportementale traditionnelle			Nouvelle synthèse comportementale		
		Lignes de code au niveau transfert de registres	Nombre de portes logique	Chemin critique (ns)	Lignes de code au niveau transfert de registres	Nombre de portes logique	Chemin critique (ns)
Scheduler	781	4176	5060	19.57	1476	4350	13.82
Sender	621	3267	3930	18.31	1289	3840	16.46
ABR_SA	425	3765	3220	18.76	952	3440	14.35
Time unit	115	985	730	16.14	209	800	13.25

Figure 6.6: Comparaison des outils de synthèse comportementale (ATM)

aussi en permettant la simulation au niveau plus élevé, ce qui réduit le nombre d'itérations à l'aide de l'outil de synthèse au niveau transfert de registres. Pendant la première phase du projet, la spécification du circuit changeait continuellement, ce qui posait plus de problèmes à l'équipe des concepteurs n'utilisant que la synthèse au niveau transfert de registres puisque le modèle comportemental peut être plus facilement adapté aux modifications.

Afin de comparer les résultats de la nouvelle approche avec les outils traditionnels, certains modules du circuits ont été aussi synthétisés à l'aide d'un outil de synthèse de deuxième génération. Les résultats de comparaison sont détaillés dans le tableau 6.6. L'outil de synthèse traditionnel dont nous nous sommes servis était AMICAL. Quoique ces résultats aient été obtenus à l'aide d'un outil spécifique, nous pouvons supposer que d'autres outils auraient donné des résultats semblables. La taille des modèles comportementaux est la même pour les deux outils alors que le nombre de lignes de code généré est de 70 % de moins que ce que nous avons obtenu en utilisant l'outil traditionnel. Ce résultat est dû à la hiérarchie de l'architecture composée du chemin de données et du contrôleur. En matière de nombre de portes logiques, la nouvelle approche a produit 10 % de moins en moyenne alors que le chemin critique est diminué de 20 %. Le code généré au niveau transfert de registres par l'outil traditionnel est sur-structuré et sur-décomposé, ce qui empêche une optimisation efficace des outils de synthèse logique.

6.1.2 La conception de l'estimateur de mouvement

L'estimation de mouvement pour des vidéo encodeurs et décodeurs (H261 motion estimator, HME) est un circuit qui détermine les vecteurs de mouvement de la partie en mouvement de l'image suivant le standard de vidéoconférence H261 [Wis89]. L'algorithme réalisé divise l'image en 99 sous-fenêtres, ce qui permet l'exécution parallèle de diverses opérations. La fenêtre de recherche pour le vecteur de mouvement est limitée à 256 combinaisons

Module	Lignes de code comportemental	Synthèse comportementale traditionnelle			Nouvelle synthèse comportementale		
		Lignes de code au niveau transfert de registres	Nombre de portes logique	Chemin critique (ns)	Lignes de code au niveau transfert de registres	Nombre de portes logique	Chemin critique (ns)
HME	290	6021	7850	11.19	538	4200	8.49

Figure 6.7: Comparaison des outils de synthèse comportementale (HME)

possibles de 16 vecteurs horizontaux et de 16 vecteur verticaux de l'image précédente. L'algorithme calcule la distance ou la distorsion entre la sous-fenêtre courante et la sous-fenêtres cible pour toutes les 256 vecteurs de mouvement possibles. Le vecteur de mouvement est celui qui a la distorsion la plus basse ou la première s'il en existe plusieurs qui ont la même distorsion. Pour pouvoir traiter 15 images à la seconde, la puissance de calcul doit être environ 97.3 Mops.

Une comparaison avec la synthèse traditionnelle a été aussi effectuée pour cette application. Les résultats sont résumés sur le tableau 6.7. En matière de surface et de nombre de lignes de code, une très forte diminution s'est avérée, ce qui est dû à une limitation d'AMICAL : les opérations sur les bits et les champs de bits n'étant pas acceptées, des unités fonctionnelles sont nécessaires pour leur réalisation. Comme le traitement des vecteurs de mouvement nécessite l'utilisation excessive des ces opérations, la description au niveau transfert de registres devient encore plus sur-structurée et contient du contrôle superflu.

Le traitement des vecteurs de mouvement est décrit par plusieurs boucles FOR imbriquées. En raison des contraintes temporaires du circuit et pour que la description au niveau transfert de registres puisse être synthétisée (champs de bits avec des indices statiques), le déroulement de certaines de ces boucles est absolument nécessaire. Les boucles contiennent des protocoles de communication avec les mémoires, ce qui nécessite l'insertion des instructions d'attente et des instructions conditionnelles. La plupart des outils de synthèse comportementale n'acceptent pas ce style d'écriture. À l'aide de notre approche basée sur un graphe de flux de contrôle et d'une phase d'élaboration puissante, la description est synthétisée tout en respectant le sous-ensemble de l'outil de synthèse logique et les contraintes temporaires.

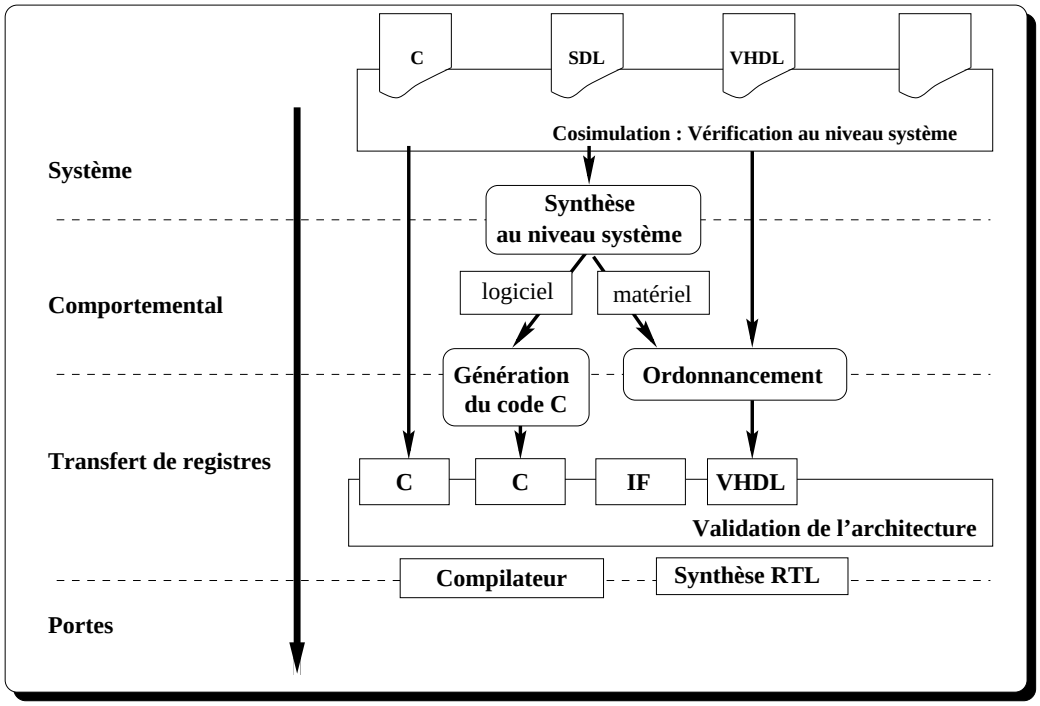


Figure 6.8: Flux de conception de l'outil de synthèse système

6.2 Lien avec la synthèse système

Nos techniques d'ordonnancement ont été intégrées dans le flux de conception d'un outil de synthèse système appelé COSMOS [IAJ94]. Cet outil de synthèse accepte une spécification en SDL [ITU93] en entrée, et produit une architecture composée de modules logiciels et matériels. L'environnement de conception s'appuie sur le concept multi-langage : les spécifications d'entrée au niveau système sont décrites en SDL ou au niveau comportemental en VHDL [VHD94]. Pour modéliser l'environnement du système, d'autres langages (COSSAP, Matlab, etc.) peuvent être utilisés. La vérification du système entier est assurée par un outil de cosimulation [Mar98]. Le modèle de sortie de l'outil est composé de trois éléments : code C pour la partie logicielle, code VHDL ou Verilog pour la partie matérielle et l'interface qui permet la communication entre les modules synthétisés. La validation de cet ensemble de descriptions est assurée par le même outil de cosimulation.

La synthèse pour les spécifications décrites en VHDL correspond exactement aux techniques présentées dans le chapitre 5 de cette thèse. La synthèse pour SDL commence par la traduction de la spécification d'entrée. Les structures SDL utilisables pour la synthèse sont restreintes à un sous-ensemble.

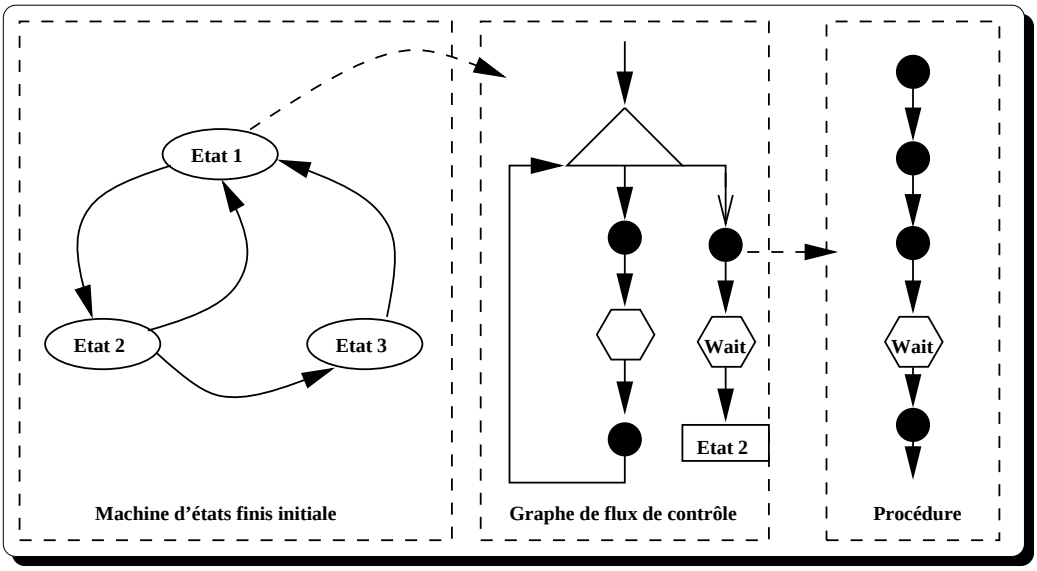


Figure 6.9: Modèle d'entrée comportemental pour SDL

Une spécification SDL est composée d'un ensemble de blocs contenant des machine d'états finis étendues. Ces blocs s'exécutent en parallèle et communiquent entre eux par des messages appelés signaux. Cette communication abstraite s'effectue à travers des canaux de communication. La notion de temps n'existe pas en SDL, les machines d'états finis sont véhiculées par des transactions. Une étude approfondie du langage SDL est présentée dans [KPSG99].

Le flux de conception de l'outil de synthèse est illustré par la figure 6.8. La structure et la hiérarchie de la spécification d'entrée peuvent être modifiées par le regroupement et le découpage des blocs, la mise à plat des machines d'états finis, etc. Une fois la structure de l'architecture fixée, l'étape de synthèse de communication remplace les canaux de communication abstraits par leur réalisation au niveau comportemental. Les accès à ces canaux sont également synthétisés à l'aide de protocoles décrits par des sous-programmes. Les modules du système sont ensuite partitionnés suivant s'ils sont réalisés en logicielle ou en matérielle. Le résultat de ce partitionnement est le niveau comportemental étendu, présenté dans la section suivante, qui constitue l'entrée de la synthèse comportementale. Cette étape construit des machines d'états finis au niveau transfert de registres.

6.2.1 Modèle d'entrée comportementale étendu

Par rapport aux descriptions comportementales décrites en VHDL, le modèle d'entrée généré par l'outil de synthèse système est plus complexe et par conséquent, sa synthèse nécessite des algorithmes d'ordonnancement et d'élaboration plus sophistiqués. Autrement dit, le niveau d'abstraction de ce modèle d'entrée est plus élevé. Ce modèle d'entrée est illustré par la figure 6.9. Du point de vue du format intermédiaire, un processus VHDL peut être considéré comme un cas extrême par rapport aux descriptions provenant du SDL. Les principales spécificités de ce modèle d'entrée sont les suivantes :

- Les tâches comportementales sont composées d'une machine d'états finis dont chaque état a le sémantique d'un processus VHDL. L'exécution de ces tâches sont mutuellement exclusives et le passage d'un état à un autre est assuré par une instruction spéciale. La synthèse s'effectue séparément pour chaque état et produit une sous-machine d'états finis dont le premier est l'entrée et certaines transitions sortent de cette machine d'états finis et déclenchent l'exécution de la suivante. Le style d'écriture exigé au niveau transfert de registres étant une machine d'états finis non-hiérarchique ces fragments sont assemblés suivant les transitions quittant la sous-machine d'états finis en question. Ainsi le modèle de sortie de l'outil est-t-il équivalent à celui qui est présenté dans le chapitre 5.

- Les protocoles de communication entre les tâches concurrentes du systèmes sont décrites sous forme de sous-programmes. Leurs appels, introduits par la synthèse de communication, donnent la réalisation des primitives de communication abstraite du langage SDL. Deux types de styles d'écriture sont acceptés suivant la nature de la communication :

  - Un style d'écriture correspondant à celui du VHDL comportemental présenté dans le chapitre 5. Cela permettra de développer les bibliothèques de communication en VHDL.
  - Les protocoles sont décrits sous forme d'une machine d'états finis, ce qui introduit le concept de machines d'états finis hiérarchiques du point de vue de l'ordonnancement. Dans ce cas, un appel de sous-programme suspend l'exécution de la machine d'états finis principale tant que le protocole de communication est actif. Comme présenté dans le chapitre 5, la mise à plat des appels de sous-programme s'effectuent au cours de la construction du graphe de flux de contrôle. Les instructions spéciales permettant de changer d'états sont remplacées par des instructions d'attente

et la relation de successeur/prédécesseur sont par conséquent appliquée. Suivant la sémantique des appels de sous-programme, le premier et le dernier état sont déroulés dans la machine d'états finis principale, autrement dit, le lieu d'appel et le point de retour ne provoquent pas de cycles d'horloge supplémentaires au niveau transfert de registres.

- Le langage SDL étant un langage de spécification, ses types sont plus abstraits que ceux du VHDL ; il lui manque des détails de réalisation. Pour pouvoir générer des descriptions au niveau transfert de registres destinées à la synthèse, il est nécessaire d'effectuer une élaboration plus sophistiquée que nous appelons la synthèse de types. Cette phase de synthèse consiste à faire correspondre les types SDL aux types définis par les packages IEEE et à modifier en conséquence les expressions de la description comportementale.

6.3 Conclusions

Ce chapitre décrit deux exemples d'application de la synthèse comportementale basés sur l'ordonnancement :

- L'outil de synthèse a été appliqué sur deux exemples complexes et industriels : un circuit ATM de 2 millions de transistors et un estimateur de mouvement H261. Ces exemples ont prouvé l'efficacité de notre nouveau flux de synthèse basé sur l'étape d'ordonnancement par rapport aux outils de synthèse comportementale classique : en moyenne, le délai du chemin critique a diminué de 10 % grâce à la suppression des interconnexions complexes d'une architecture traditionnelle ; le nombre de portes logiques est de 20 % inférieur, ce qui est dû au style d'écriture adapté à la synthèse au niveau transfert de registres permettant les optimisations plus puissantes au niveau des portes logiques.

  Le processus de conception du circuit ATM nous a permis de faire également la comparaison entre la méthodologie classique basée sur l'utilisation de la synthèse au niveau transfert de registres et notre approche pour la synthèse comportementale. Les résultats obtenus avec les deux flux de conceptions sont approximativement équivalents. En revanche l'effort de conception a été divisé par 3, ce qui prouve l'augmentation escomptée en matière de productivité.

- Le flux de synthèse comportementale a été intégré à un outil de synthèse système commercial. Cette intégration a nécessité l'extension du

modèle d'entrée comportemental avec la représentation des machine d'états finis hiérarchiques. L'algorithme d'ordonnancement a été également étendu pour la synthèse de ces types de systèmes complexes et hiérarchiques. L'approche d'ordonnancement, basée sur un graphe de flux de contrôle, s'est avérée bien adaptée aux descriptions générées automatiquement par l'outil de synthèse système parce qu'elle n'impose guère de restrictions sur le style d'écriture et, du point de vue des résultats de synthèse, est peu sensible au modèle d'entrée.

Chapitre 7

Conclusions et perspectives

7.1 Conclusions

L'objectif principal de cette thèse est une analyse de la synthèse comportementale classique qui n'a jamais réussi à atteindre le degré d'acceptation escompté. En s'appuyant sur cette étude, nous proposons une nouvelle approche de synthèse comportementale : un flux de synthèse basé uniquement sur l'étape d'ordonnancement.

Le chapitre 2 donne un aperçu du contexte de la synthèse comportementale. Cette méthodologie de conception est définie comme un ensemble de raffinements permettant la compilation des descriptions algorithmiques à un niveau d'abstraction plus bas : au niveau transfert de registres. Le lien entre la synthèse comportementale et d'autres outils de conception est également étudié. D'une part, l'architecture résultante est synthétisée par des outils de synthèse logique, d'autre part, les descriptions comportementales peuvent être générées automatiquement par des outils de synthèse système. Il est donc évident que la recherche scientifique en synthèse comportementale doit tenir compte de l'évolution des ces outils. Les principes de base de la synthèse comportementale sont aussi présentés dans cette section, tels que les domaines d'application, les formats intermédiaires et les étapes de synthèse pour l'approche classique. Vu que les recherches de cette thèse se focalisent sur l'étape d'ordonnancement, un accent particulier est mis sur les algorithmes d'ordonnancement les plus connus dans le monde de la synthèse comportementale.

Dans le chapitre 3, une étude discutant les limitations des outils de synthèse comportementale classiques est présentée. Nous pouvons conclure que le principal problème de ces outils est leur mauvaise intégration aux flux de conception existants. L'architecture générée, composée d'un contrôleur et

d'un chemin de données avec l'allocation fixe des unités fonctionnelles, empêche l'exploitation des nouvelles capacités de la synthèse au niveau transfert de registres. L'étude démontre également que deux étapes de la synthèse comportementale, l'allocation et l'association des unités foncitonnelles peuvent être accomplies à l'aide des outils de synthèse au niveau transfert de registres. Ces outils s'appuyant sur de puissantes optimisations logiques, les résultats sont plus efficaces que ce que l'on peut obtenir avec l'approche classique. Deux outils de synthèse sont aussi présentés dans ce chapitre : Behavioral Compiler et AMICAL. L'analyse de ces outils montre d'autres défaillances des outils de synthèse comportementale : outils trop spécialisés à un certain domaine d'application, style d'écriture restreint, flux de conception très complexe, etc. Ces restrictions empêchent essentiellement l'intégration de la synthèse comportementale à un flux de synthèse système.

Les concepteurs des circuits intégrés décrivent généralement la fonctionnalité de l'algorithme à réaliser à l'aide de langages de description de matériel. Le langage le plus répandu pour la synthèse comportementale, VHDL est présenté dans le chapitre 4. Une étude des diverses structures du langage, leurs interprétations suivant les outils de synthèse et leur utilisation pour la synthèse comportementale sont également discutées. L'étude a démontré qu'il était difficile de définir un sous-ensemble général pour la synthèse comportementale, ce qui est essentiellement dû à la complexité des transformations effectuées durant le processus de synthèse. Chaque outil impose certaines restrictions sur le modèle d'entrée et exige un style d'écriture afin d'obtenir les résultats escomptés. Cette diversité de styles d'écriture pose des problèmes sérieux pour l'intégration de la synthèse comportemental à un flux de synthèse système. Le sous-ensemble défini par notre approche de synthèse ne restreint guère le style de codage au niveau comportemental, ce qui nous a permis une interface efficace avec un outil de synthèse système.

Une nouvelle approche de la synthèse comportementale qui restreint la synthèse comportementale à l'étape d'ordonnancement est présentée dans le chapitre 5. Un outil de synthèse donne une réalisation de cette nouvelle approche qui utilise un algorithme d'ordonnancement basé sur un graphe de flux de contrôle étendu par la représentation des dépendances de données. Cette représentation double est une ouverture pour la synthèse des applications mixtes, orientées à la fois flux de données et flux de contrôle. Les étapes de synthèse sont détaillées ainsi que le modèle d'entrée et la technique de la génération des descriptions au niveau transfert de registres. L'ensemble d'algorithmes permet d'éviter toutes les défaillances présentées ci-dessus.

Pour évaluer l'efficacité de notre approche, nous avons utilisé des applications complexes et industrielles. Des comparaisons entre la méthodologie de la synthèse comportementale et celle au niveau transfert de registres ont

été effectuées. Les résultats prouvent que la qualité des descriptions générées automatiquement est comparable à des conceptions décrites manuellement au niveau transfert de registres. Notre outil de synthèse a été aussi comparé avec un outil de synthèse comportementale classique. Cette comparaison montre une amélioration considérable en matière de surface et de chemin critique par rapport aux approches traditionnelles. Le chapitre 6 montre également l'intégration de notre outil de synthèse dans un flux de synthèse système. La flexibilité de notre algorithme d'ordonnancement a permis facilement l'extension du modèle comportemental pour la représentation des systèmes complexes et hiérarchiques.

7.2 Perspectives

La nouvelle approche présentée dans le chapitre 5 a été évaluée sur deux applications complexes. Ces applications, tout en contenant certaines opérations arithmétiques, sont orientées principalement flux de contrôle. Notre algorithme de chaînage basé sur un graphe de flux de contrôle ne limite nullement le nombre d'opérations et le chemin critique pour les transitions de la machine d'états finis au niveau transfert de registres. Une description orientée purement flux de données (calculs complexes entre deux points de synchronisation) engendre pour le circuit généré une fréquence d'horloge trop basse et un nombre de ressources trop élevé. Il est donc nécessaire d'introduire un deuxième niveau d'ordonnancement sous contraintes de temps et/ou de ressources permettant de décomposer ces transitions critiques en plusieurs étapes de contrôle. La complexité de ce problème provient du fait qu'il soit très difficile de prévoir au niveau comportemental l'architecture générée par la synthèse logique. L'estimation de performance est donc une tâche très complexe même en ayant des informations sur les portes logiques de la bibliothèque technologique. En revanche, l'évolution de la synthèse au niveau transfert de registres permet certaines optimisations de haut niveau telle que l'équlibrage des délais entre les blocs de registres (*Behavioral Retiming*). Ainsi l'outil de synthèse comportementale doit-il introduire des cycles d'horloge supplémentaires dont le nombre peut être déterminé par une pré-évaluation à l'aide de l'outil de synthèse logique. Le deuxième niveau d'ordonnancement peut être donc effectué par la synthèse au niveau transfert de registres, ce qui permet d'obtenir des résultats plus précis.

L'intégration des techniques de synthèse comportementale avec le flux de synthèse système n'a pas été évaluée et étudiée de manière approfondie. Un lien plus fort entre l'ordonnancement et la synthèse de la communication doit être analysé [GABP98]. Une boucle d'exploration d'architecture

peut être établie entre ces outils, ce qui permet à la fois l'ordonnancement et l'optimisation des opérations de la spécification et des protocoles de communication.

Liste des figures

Bibliographie

[ASU86] A. Aho, R. Sethi, and J. Ullmann. *Compilers, Principles, Techniques and Tools.* Addison-Wesley Publishing, 1986.

[Bar73] M. Barbacci. *Automatic Exploration of the Design Space for Register Transfer (RT) Systems.* PhD thesis, Dept. Of CS, Carnegie-Mellon University, 1973.

[BDB94] S. Bhattacharya, S. Dey, and F. Brglez. Performance Analysis and Optimization of Schedules for Conditional and Loop-Intensive Specifications. In *Proc. of the 31th Conference on Design Automation*, pages 491–496, New York, NY, USA, juin 1994.

[Ber97] E. Berrebi. *Méthodologie pour l'application industrielle de la synthèse comportementale.* PhD thesis, Institut National Polytechnique de Grenoble, décembre 1997.

[Ber99] R.A. Bergamaschi. Behavioral Network Graph: Unifying the Domains of High-Level and Logic Synthesis. In *Proc. of 36th ACM/IEEE Design Automation Conference*, Juin 1999.

[BK92] J. Biesenack and M. Koster. The Siemens High-Level Synthesis System, CALLAS. In *Proc of High-Level SYnthesis Workshop*, novembre 1992.

[Bla97] David Black. Interactive TV Set-Top BOX ASIC With Behavioral Compiler. In *Apple Computer Designs*, ocotbre 1997.

[BRNT98] R. A. Bergamaschi, S. Raje, I. Nair, and L. Trevillyan. Control-Flow Versus Data-Flow-Based Scheduling: Combining Both Approaches in an Adaptive Scheduling System. In *Transaction On VLSI Systems.* IEEE, Mars 1998.

[Cam91] R. Camposano. Path-Based Scheduling for Synthesis. *IEEE transactions on CAD*, 10:85–93, janvier 1991.

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

[CDT91] R. Camposano, L.F. Daunders, and R.M. Tabet. VHDL as Input for High-Level Synthesis. *IEEE Design and Test of Computers*, pages 43–49, 1991.

[CLS93] L.F. Chao, A. LaPaugh, and E.H.M. Sha. Rotation Scheduling: A Loop Pipelining Algorithm. In *Proc. of the Design Automation Conference*, pages 562–566, juin 1993.

[Com96] The ATM Forum Technical Committee. Traffic Management Specification V4.0. Technical report, ATMC, avril 1996.

[Des99] Design Automation, High Performance Systems, ICL, Wenlock Way, West Gorton, Manchester M12 5DR, United Kingdom. *VHDL+ L.R.M. Extensions to VHDL for System Specification*, mars 1999.

[DW99] A. DeHon and J. Wawrzynek. Reconfigurable Computing: What, Why, and Implications for Design Automation. In *36th ACM/IEEE Design Automation Conference*, juin 1999.

[GABP98] G. Gogniat, M. Auguin, L. Bianco, and A. Pegatoquet. Communication Synthesis and HW/SW Integration for Embedded System Design. *IEEE Transaction*, pages 49–53, 1998.

[GDWL92] D. Gajski, N. Dutt, A. Wu, and Y. Lin. *High-Level Synthesis: Introduction to Chip and System Design.* Kluwer Academic Publishers, Boston, Ma. USA, 1992.

[GE92] C.H. Gebotys and M.L. Emasty. Optimal VLSI Architectural Synthesis: Area, Performance and Testability. page 289. Kluwer Academic Publishers, 1992.

[GG87] T. Gautier and P. Le Guernic. Signal, A Declarative Language for Synchronous Programming of Real-Time Systems. *Computer Science, Functional Languages and Computer Architectures*, 274, 1987.

[GIC+96] D. Gajski, T. Ishii, V. Chaiyakul, H. Juan, and T. Hadley. Interactive Behavioral Synthesis. In *Proc of SASIMI*, 1996.

[GKP85] J. Granacki, D. Knapp, and A. Parker. The (ADAM) Advanced Design Automation System: Overview Planner and Natural Language Interface. In *Proc. of the 22nd ACM/IEEE Design Automation Conference*, Los Alamitos, Ca. USA, Juin 1985.

[GP92] A. Grenier and F. Pêcheux. ALLIANCE : A Complet Set of CAD Tools for Teaching VLSI Design. In *3e'me Eurochip Workshop on VLSI Design Training*, 1992.

[GPR98] L. Guerra, M. Potkonjak, and J. Rabaey. A Methodology for Guided Behavioral-Level Optimization. In *Proc. of Design Automation Conference*, pages 309–314, 1998.

[GR94] D.D. Gajski and L. Ramachandran. Introduction to High-Level Synthesis. *IEEE Design And Test of Computers*, Winter:44–54, 1994.

[HCRP91] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. Programmation et Vérification des Langages Réactifs : le langage Lustre. *Technique et Science Informatique*, 10(2), 1991.

[Hil85] P. N. Hilfinger. A High Level Language and Silicon Compilation for Digital Signal Processing. In *Proc. of IEEE Custum integrated Circuits Conference*, pages 213–216, 1985.

[Hu61a] T. C. Hu. Parallel Sequencing and Assembly Line Promblems. *Operations Research*, pages 841–848, novembre 1961.

[Hu61b] T.C Hu. Parallel Sequencing and Assembly Line Problems. *Operations Research*, pages 841–848, novembre 1961.

[IAJ94] T. D. Ismail, M. Adib, and A. A. Jerraya. COSMOS: A Codesign Approach for Communication Systems. In *Proc. of Workshop on Hardware/Software Codesign (Codes)*, pages 17–24, 1994.

[ITU93] ITU-T, CCITT Specification and description language SDL. *Recommendation Z.100.*, 1993.

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

[JKC98] David M. Johnson, Brian T. Kelley, and Jopse G. Corleto. Design Automation of a ReceiverL: Breaking the RTL Cycle Time Barrier Using Behavioral Compiler. avril 1998.

[Jon93] G.G. Jong. *Generalized Data Flow Graphs, Theory and Applications.* PhD thesis, Eindhoven University of Technology, 1993.

[KB98] M. Keating and P. Bricaud. Reuse Methodology Manual for System-On-A-Chaip Designs. *Kluwer Academic Publishers*, page 240, juin 1998.

[KCG+98] K. Kuçukçakar, C.T. Chen, J. Gong, W. Philipsen, and T.E. Tkacik. Matisse: An Architectural Design Tool for Commodity ICs. *IEEE Design & Test of Computers*, avril-juin 1998.

[KJ95] P. Kission and A.A. Jerraya. High Level Specification in Electronic Design. *IEEE International Symposium on Industrial Electronics*, 1995.

[KM90] D. Ku and G. De Micheli. HardwareC - A Language for Hardware Design. Technical report, Computer Systems Laboratory, Universite' de Stanford, 1990.

[Kna96] D. W. Knapp. *Behavioral Synthesis: Digital System Design Using the Synopsys Behavioral Compiler.* Prentice Hall, 1996.

[KP91] D. Knap and A. Parker. The ADAM Design Planning Engine. *IEEE trans. on Computer-Aided Design*, 10(7):829–846, 1991.

[KPSG99] C. D. Kloos, S. Pickin, L. Sanchez, and A. Groba. High-Level Specification Languages for Embedded System Design. In *System-Level Synthesis*, volume Kluwer Academic Publisher, 1999.

[LG88] J.S. Lis and D.D. Gajski. Synthesis form VHDL. In *Proc. of International Conference on Computer-Aided Design*, pages 378–381, octobre 1988.

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

[Mar79] P. Marwedel. The MIMOLA Design System: Detailed Description of the Software System. In *16th Design Automation Conference Proceedings*, pages 59–63, New York, USA, Juin 1979.

[Mar86] P. Marwedel. A New Synthesis Algorithm for the MIMOLA Software System. In *Porc. of the Design Automation Conference*, 1986.

[Mar98] Ph. Le Marrec. HW/SW and Mechanical Cosimulation for Automotive Application. In *Proc. of RSP'98*, Belgium, 1998.

[MCG+90] H. De Man, F. Catthor, G. Goossens, J. Van Meerbergen, S. Note, and J. Huisken. Architecture-Driven Synthesis Techniques for VLSI Implementation of DSP Algorithms. In *Proc. of IEEE*, pages 319–355, fevrier 1990.

[Mig92] A. Mignotte. *Synthèse architecturale de circuits intégrés.* PhD thesis, Institut National Polytechnique de Grenoble, 1992.

[MK88] G. De Micheli and D.C. Ku. HERCULES - a System for High-Level Synthesis. In *Proc. of Design Automation Conference*, 1988.

[MSDP93] E. Martin, O. Sentieys, H. Dubois, and J. L. Philipe. GAUT: An Architectural Synthesis Tool for Dedicated Signal Processors. In *Proc. of the Europeen Conference on Design Automation*, Paris, France, mars 1993.

[MSS+99] I. Moussa, Z. Sugar, R. Suescun, A.A. Jerraya, M. Diaz-Nava, M. Pavesi, S. Crudo, and L. Gazzi. Comparing RTL and Behavioral Design Methodologies in the Case of a 2M Transistors ATM Shaper. In *Proc. of DAC'99*, New Orleans, USA, Juin 1999.

[NBD92] V. Nagasamy, N. Berry, and C. Dangelo. Specification, Planning and Synthesis in a VHDL Design Environment. *IEEE Design and Test of Computers*, pages 58–68, juin 1992.

[Not91] S. Note. CATHEDRAL III: Architecture-driven High-Level Synthesis for High Throughput DSP Applications. In *Proc. of the Design Automation Conference*, 1991.

[OG86a] A. Orailoglu and D.D. Gajski. Multi-paradigm Approach to Automatic Data-path Synthesis. In *Proc. of 23nd Design Automation Conference*, 1986.

[OG86b] A. Orailogo and D.D. Gajski. Flow Graph Representation. In *Proc. of the Design Automation Conference*, juin 1986.

[OM96] K. O'Brien and S. Maginot. Towards Maximising the use of VHDL for Synthesis. In *Proc. of the European Design Automation Conference*, Geneve, Septembre 1996.

[ORJ93] K. O'Brien, M. Rahmouni, and A.A. Jerraya. DLS: A Scheduling Algorithm for High-Level Synthesis in VHDL. In *Proc. of the European Conference on Design Automation*, Paris, France, fevrier 1993.

[PK89] P. G. Paulin and J. P. Knight. Force Directed Scheduling for the Behavioral Synthesis of ASIC's. *IEEE transactions on CAD*, 6:661–679, juin 1989.

[PKG86] P. G. Paulin, J. P. Knight, and E. F. Girzyc. HAL: A Multi-paradigm Approach to Automatic Data-Path Synthesis. In *Proc. of the Design Automation Conference*, 1986.

[PR94] M. Potkonjak and J. Rabaey. Algorithm Selection: A Quantitative Computation-Intensive Optimisation Approach. In *Proc. ICCAD Conference*, Santa Clara, novembre 1994.

[Rah97] M. Rahmouni. *Ordonnancement pour la synthèse de haut niveau.* PhD thesis, Institut National Polytechnique de Grenoble, 1997.

[RGC94] L. Ramachandran, D. D. Gajski, and V. Chaiyakul. An Algorithm for Array Variable Clustering. In *Proc. of the European Design Automation Conference*, Paris, France, 1994.

[RJ95] M. Rahmouni and A.A. Jerraya. PPS: A Pipeline Path based Scheduler. In *Proc. of the Europeen Conference on Design Automation*, Paris, France, mars 1995.

[ROJ94] M. Rahmouni, K. O'Brien, and A.A. Jerraya. A loop-based scheduling algorithme for hardware description languages. *Parallel Processing Letters*, 4(3):351–364, 1994.

[Sau87] L.F. Saunders. The IBM VHDL Design System. In *Proc. of Design Automation Conference*, pages 484–490, 1987.

[SB99] Scott Smith and David Black. Pushing the Limits with Behavioral Compiler. In *http://www.synopsys.com.* Compaq Computer and Qualis Design Corporation, Mai 1999.

[SM98a] A. Seawright and W. Meyer. Partitioning and Optimising Controllers Synthesized from Hierarchical High-Level Descriptions. *35th ACM/IEEE Design Automation Conference*, juin 1998.

[SM98b] Luc Séméria and Giovanni De Micheli. SpC: Synthesis of Pointers in C, Application of Pointer Analysis to the Behavioral Synthesis from C. In *Porc. of the ICCAD*, San Jose, CA, USA, 1998.

[SMG95] S. Swanny, A. Molin, and B. Govnot. OO-VHDL: Object-Oriented Extensions to VHDL. *Computer*, octobre 1995.

[SN95] G. Schumacher and W. Nebel. Inheritance Concept for Signals in Object-Oriented Extensions to VHDL. In *Proc. of the European Design Automation Conference*, 1995.

[Sou83] J.R. Southard. MacPitts: An Approach to Silicon Compilation. *IEEE Computer Magazine*, 16(12):74–82, Decembre 1983.

[ST95] H. Schmit and D. Thomas. Array Mapping in Behavioral Synthesis. In *Proc. of International Sym. Sys. Synthesis*, septembre 1995.

[Sto91] L. Stok. *Architectural Synthesis and Optimization of Digital Systems.* PhD thesis, Eindhoven University of Technology, 1991.

[Sto94] L. Stok. A Decade of High Level Synthesis: Fundementals and Applicatinos. In *Proc. IX Congress of Brazilian Microelectronics Society (SBMICRO)*, Rio de Janeiro, août 1994.

[Syn98a] Synopsys. *Protocol Compiler Reference Manual.* Synopsys, 1998.

[Syn98b] Synopsys. *VHDL Compiler Reference Manual.* Synopsys, fevrier 1998.

[Syn99] Synopsys. Design Compiler Ultra Performance Capabilites. Technical report, Synopsys, Juin 1999.

[Tho81] D. E. Thomas. The Automatic Synthesis of Digital Systems. In *Proc. IEEE Digital Systems*, volume 69, pages 1200–1211, 1981.

[TLW+90] Thomas, Lagnese, Walker, Nestor, Rajan, and Blackburn. *Algorithmic and Register Transfer Level Synthesis: The System Architect's Workbench.* Kluwer, 1990.

[TM91] D. E. Thomas and P. Moorby. *The VERILOG Hardware Description Langauge*. Kluwer Academic Publishers, 1991.

[Ves98] Sarosh Vesuna. Automates System-to-GatesDesign Flow for Wireless LAN ASIC with COSSAP and Behavioral Compiler. In *Symbol Technologies Inc.* Alcatel and Synopsys, mars 1998.

[VHD94] Institute of Electrical and Electronics Engineers, 345 East 47th Street, New York, NY 10017, USA. *IEEE Standard VHDL Language Reference Manual*, Juin 1994. ANSI/IEEE Std 1076-1993.

[WBD+94] N. Wehn, J. Biesenack, Peter Duzy, M. Munch T. Landmaier, Michael Pilsl, and S. Rumler. Scheduling of Behavioral VHDL by Retiming Techniques. In *Proc. of European Conference on Design Automation*, 1994.

[WC91] R.A. Walker and R. Composano. *A Survey of High-Level Synthesis Systems*. Kluwer Academic Publishers, Boston, Ma., 1991.

[Wis89] A. Wise. Introduction to Motion Picture Coding and the CCITT Algorithm. decembre 1989.