Synthèse architecturale interactive et flexible

Hong Ding

THESE

présentée par

Hong DING

pour obtenir le titre de **DOCTEUR**

de l'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

(arrêté ministériel du 30 mars 1992)

Spécialité : **Microélectronique**

SYNTHESE ARCHITECTURALE INTERACTIVE ET FLEXIBLE

Date de Soutenance : 2 Avril 1996

Composition du jury :

Messieurs :	Guy MAZARE	*Président*
	Michel AUGUIN	*Rapporteur*
	Ramayya KUMAR	*Rapporteur*
	Ahmed Amine JERRAYA	
	Kevin O'BRIEN	*Invité*

Thèse préparée au sein du Laboratoire TIMA/INPG

献给欣
献给我的父母们
献给阳，丹，炜，雯
献给我的朋友们

Remerciement

Je tiens à remercier:

Monsieur Bernard Courtois, Directeur de recherches au CNRS et Directeur du Laboratoire TIMA, pour m'avoir accueilli au sein du Laboratoire.

Monsieur Guy Mazaré, Directeur de l'Ecole Nationale Supérieure d'Informatique et de Mathématiques Appliquées de Grenoble pour m'avoir fait l'honneur d'être président de mon jury.

Monsieur Ahmed Amine Jerraya, Chargé de recherches au CNRS et Responsable de groupe "System-Level Synthesis", pour les nombreuses discussions et les encouragements qui m'ont beaucoup aidé dans l'orientation et l'avancement de mes travaux.

Messieurs Michel Auguin, Professeur à l'Université de Nice, et Ramayya Kumar, Directeur du département ACS à l'Université de Karlsruhe m'avoir fait l'honneur d'être rapporteurs de cette thèse et membres du jury.

Monsieur Kevin O'Brien, Ingénieur à LEDA pour avoir accepter mon invitation à être membre du jury.

Tous les membres de l'équipe de Synthèse au Niveau Système.

Tous ceux qui ont contribué à la correction et à la préparation de ce manuscrit, et ils sont nombreux: Philippe Guillaume, Maher Rahmouni, Elisabeth Berrebi, Polen Kission, Mohamed Hedi Touati et Haibin Yin.

Le reste du Laboratoire TIMA dans son ensemble, et notamment sa partie administrative (Corinne, Patricia, Chantal, Isabelle 1 & 2)

SYNTHESE ARCHITECTURALE INTERACTIVE ET FLEXIBLE

Résumé

Cette thèse présente plusieurs travaux visant à l'amélioration de la synthèse architecturale réalisée à l'aide de l'outil de synthèse de haut niveau AMICAL. Un point clé de ce travail est la notion d'interactivité. Le processus de synthèse se décompose en un ensemble de raffinements successifs. L'utilisateur a la possibilité d'intervenir au cours de ces différentes étapes et d'agir manuellement, ou au contraire de laisser se dérouler seules l'ensemble des étapes tout en gardant une vision claire des actions effectuées. Ce dernier a de plus le choix entre plusieurs styles architecturaux qu'il pourra implémenter à son gré, ce qui autorise une grande flexibilité.

Les points principaux abordés au cours de cette thèse sont les suivants :

- Les étapes et modèles successifs de raffinement au cours du processus de synthèse : chaque sous-tâche engendre un modèle architectural intermédiaire à partir duquel la sous-tâche suivante pourra agir.
- La notion d'interactivité : celle-ci inclue la mise au point d'un modèle de performance permettant d'estimer la qualité du circuit synthétisé, et permet au concepteur d'être le véritable acteur de la synthèse tout en l'assistant lors de la prise de décisions.
- La génération de plusieurs types d'architectures et les problèmes algorithmiques qui y sont liés.

Mots-Clefs: Synthèse architecturale, Synthèse interactive, Modèle architectural, Modèle de performance, Génération d'architectures.

Abstract

This thesis presents an interactive High Level Synthesis environment called AMICAL. The synthesis process is decomposed into a set of refinement steps. The user can execute these steps automatically, manually or in interactive mode when needed. The synthesis scheme is flexible; it allows several architectural models for the generated data-path (bus model, multiplexer model) and controller (hardwired, programmable).

The main issues developed in this thesis are:

- The models and steps used for refinements in a synthesis process. Several architectural models are defined for bridging gap between two synthesis steps.
- The interactive synthesis model. It includes a performance model allowing to estimate the synthesized results, and allows the designer to be a real actor of the synthesis process.
- The generation of different architectures and their algorithm issues. These architectures are usable as inputs for lower synthesis tools.

Keywords: Architectural synthesis, Architectural models, Algorithms, Functional unit allocation, Control-flow dominated circuits, Performance model, Interactive synthesis, Architecture generation.

Table des Matières

Liste des Figures

Liste des Tableaux

CHAPITRE 1

Introduction

Chapitre 1

Introduction

Depuis l'arrivée des circuits VLSI (Very Large Scale Integration), la complexité des circuits implantés n'est plus limitée par la technologie, mais par le processus de conception. Un concepteur ne peut plus réaliser certains circuits sans l'aide d'outils de CAO tant la complexité des circuits est grande, et ceci pour plusieurs raisons : le temps de conception, l'optimisation et la fiabilité des circuits. L'automatisation de la réalisation des circuits intégrés est donc aujourd'hui une nécessité[Arm88][CR89].

Avec l'évolution des outils de CAO de VLSI, la conception au niveau architectural devient de plus en plus le goulet d'étranglement dans le processus de conception d'un circuit intégré[Cam90].

Depuis l'apparition des premiers outils de compilation d'architecture tels que ALERT[FY69], MIMOLA [Zim79] et MacPitts [Sou83], plusieurs compilateurs ont été publiés dans la littérature. Malgré les étapes accomplies par les recherches dans le domaine de la compilation de comportement [MPC90], peu d'outils ont été utilisés de manière industrielle. Quelques expériences seulement sont citées dans la littérature sur des essais industriels de certains de ces compilateurs.

La non-prolifération de ce type d'outils peut s'expliquer par la difficulté d'intégration de ces outils dans des environnements de conception existants tels que ceux fournis par CADENCE, SYNOPSYS[Inc92], MENTOR, VIEWLOGIC, etc. En fait, jusqu'à une date récente, peu de ces compilateurs ont utilisé des langages de spécification standards.

Avec l'arrivée de VHDL une nouvelle génération d'outils de compilation de comportement est en train de faire son apparition. L'utilisation de VHDL, en tant que langage standard, facilitera l'intégration de ces outils dans les environnements de CAO de VLSI existants[CST91].

C'est à partir de la deuxième moitié des années soixante-dix, que le domaine de la compilation d'architecture a commencé à faire l'objet d'une littérature abondante. Les outils ALERT et MIMOLA sont les précurseurs dans ce domaine. MacPitts a été le premier compilateur de comportement commercialement disponible. Il a été le premier à avoir généré un dessin de masques en partant d'une description comportementale. Peu de compilateurs de comportement ont pris en compte les contraintes dues aux étapes de compilation de bas niveau. Parmi les compilateurs de comportement, on peut citer, par exemple, et la liste n'est pas exhaustive, les outils suivants:

- HYPER à l'université de Berkeley [CPTR89],
- VSS à l'université d'Irvine [LG88],
- ELF, HAL à l'université de Carleton [GBK85, PKG86],
- CMU-DA, SAW et ArchitectWorkBench à l'université de Carnegie Mellon [DPST81, TDW+88],
- CHIPPE, SLICER, SPLICER à l'université de l'Illinois [BG87, PG87, Pan88],
- CADDY/DSL à l'université de Karlsruhe [KNRR88],
- MIMOLA à l'université de Kiel [Mar79, Mar86, Zim79],
- CAMAD à l'université de Linköping [Pen86],
- MOVIE à l'université de Lund [AP89],
- ADAM à l'université de Southern California [GDP85],
- FLAMEL, HERCULES à l'université de Stanford [MK88, Tri87],
- DAGAR à l'université de Texas [RP91],
- LYRA, ARYL à l'université de Tsing Hua [HCLH90],
- SPAID à l'université de Waterloo [HE89],

- BRIDGE à AT&T Bell Labs [TWRT88],
- YASC à Control Data Corporation [KSJ85],
- YSC et V-compiler à IBM, centre T. J. Watson [BCM+88, Ber87],
- CATHEDRAL I, II, III, et IV à l'IMEC [DCG+90, RMVea88],
- PARSIFAL à General Electric [Cas89],
- SILC à GTE [BFR85],
- HARP à NTT LSI [TKk89],
- PHIDEO à Philips [LMdWV91],
- SYCO au TIMA, Grenoble [Jer89, JMGC88],
- ASYL au CSI, Grenoble [dPDL+89].

Un compilateur d'architecture permet d'automatiser le processus de conception de l'architecture d'un circuit intégré à partir d'une spécification comportementale. Ce processus comporte une série de transformations qui permettent de passer de la description comportementale à une description architecturale satisfaisant un certain nombre de contraintes. Cette architecture doit aussi tenir compte des outils et méthodologies qui seront utilisés pour la conception logique et physique en vue de la génération des masques.

Parmi les sujets tournants autour de la synthèse de haut niveau, la modélisation des architectures a pour objet de formaliser la définition de modèles architecturaux qui pourront être utilisés au cours du processus de synthèse. Le but de cette modélisation est d'essayer de résumer les problèmes existants à chaque niveau de transformation afin de déterminer et d'optimiser les algorithmes utilisées. Basé sur celle-ci, un modèle de performance global peut être extrait avec lequel il devient possible d'évaluer les résultats issus de cette synthèse. La génération d'architectures, sur la base des modèles définis, permettent de jeter un pont entre un outil de synthèse comportementale et des outils de synthèse RTL. S'ajoutant aux points précédents, la notion d'interactivité pour un outil de CAO est primordiale; son attrait est de fournir diverses facilités au concepteur afin d'intervenir au cours des processus de raffinements, d'obtenir des informations en temps réels sur ces derniers et d'accéder aux informations liées aux performances du circuit synthétisé.

Cette thèse présente la conception du compilateur d'architecture AMICAL[Par92], englobants les points cités précédemments : modèles architecturaux, synthèse interactive, modèle de performance et génération d'architectures.

La contribution de cette thèse consiste en certains nombre d'evolutions majeures. Basé sur la première version du système AMICAL, ce travail de thèse est construit dans l'optique d'apporter plusieurs améliorations et optimisations par rapport au système préexistant. La table 1.1 résume les principales caractéristiques de la nouvelle version par rapport à la précédente.

Aspects	**AMICAL V1**	**AMICAL V3**
Exploration Architecturale	architecture basée basée sur les Bus	architecture optimale basée sur les Bus architecture basée sur les multiplexeurs architecture micro-programmable
Résultat de la compilation	réseau d'interconnexions de la partie opérative(BUS)	réseau d'interconnexions du circuit avec ou sans interface entre la partie opérative(BUS,Mux) et la partie contrôle(micro-programmable)
Fichiers d'entrée	grammaires fixées	grammaires flexibles (lex, yacc) paramétrée par la technologie multi-entreés : SOLAR/mac
Unité de connexions	bus, switch bi-directionnel	bus, multiplexeurs switch unidirectionnel
Evaluation des résultats	estimation de la surface de partie opérative	estimation de la surface du circuit, estimation de la vitesse du circuit estimation de la puissance du circuit
Langage de programmation	C (25000 lignes) bibliothèque de SUNVIEW	C (70000 lignes) bibliothèque de XVIEW

Table 1.1: *Contribution*

Celles-ci concernent principalement :

- La possibilité d'ajouter une interface entre partie contrôle et partie opérative, permettant d'une part de réaliser un pipeline entre ces deux blocs, augmentant le schéma de synchronisation de base, et d'autre part d'ajouter, au sein même du circuit, des cellules de complexité variée pouvant être utilisées par exemple pour du test intégré.
- La mise au point d'un modèle d'estimation des performances globales du circuit synthétisé, destiné à fournir au concepteur des mesures fiables concernant surface, vitesse et consommation permettant notamment de mieux mettre en évidence

l'impact de décisions au cours de la synthèse interactive. Cette partie fait l'objet d'une étude soutenue.

- L'extension du choix d'architectures cibles aux architectures à base de multiplexeurs et aux architectures micro-programmable. De plus, la solution bus initiale, composée de commutateurs bi-directionnels, à été orientée vers une solution à commutateurs uni-directionnels, plus réaliste et moins coûteuse en terme de surface.

Les deux derniers points présentent une caractéristique commune, qui est de permettre une exploration architecturale plus aisée et donc plus approfondie.

Le déroulement de cette thèse suivra les étapes suivantes :

- Au cours du chapitre immédiatement suivant, une vue globale du système AMICAL sera présentée, précisant ses spécification d'entrée, ses étapes de synthèse et ses modèles d'exécution générés par les différents algorithmes caractéristiques du système.

- Le chapitre 3 présentera le flot de synthèse global ainsi que les différents modèles architecturaux intermédiaires utilisés et générés par AMICAL durant le processus de synthèse.

- Le chapitre 4 présentera les algorithmes de génération des types d'architectures pouvant être obtenus en sortie d'AMICAL. Cette étape de génération permet d'assurer le comportement de la spécification initiale et de générer une description pouvant être utilisée par d'autres outils. Basé sur la même représentation abstraite, plusieurs architectures peuvent être réalisées selon le besoin de l'utilisateur. Des architectures variées pourront donc être fournies afin de s'adapter à différents objectifs de conception.

- Le chapitre 5 discutera de la notion d'interactivité *outil-concepteur* et de son influence sur la qualité du circuit final. A ce propos, le modèle de performance présent au sein d'AMICAL et son utilité dans le contexte de l'interactivité seront étudiés.

- Finalement le chapitre 6 conclura cette thèse en donnant une idée de la contribution apportée tout au long de ce travail au système AMICAL. De futures directions d'étude seront esquissées.

CHAPITRE 2

Le système AMICAL

Chapitre 2

Le système AMICAL

Ce chapitre décrit le compilateur d'architecture AMICAL. Un compilateur d'architecture est un outil de synthèse de haut niveau permettant d'automatiser le processus de compilation de l'architecture d'un circuit intégré à partir d'une spécification comportementale. AMICAL est un assistant interactif pour la synthèse de haut niveau et l'exploration architecturale. Partant d'une spécification purement comportementale, donnée en VHDL, et d'une bibliothèque externe d'unités fonctionnelles, AMICAL exécute plusieurs tâches essentielles (l'ordonnancement, l'allocation des unités fonctionnelles et des connexions, génération de l'architecture) et génère une architecture abstraite, au niveau transfert de registres (RTL), composée d'une partie opérative et d'une partie contrôle. Cette architecture peut être utilisée par les outils de conception au niveau transfert de registres.

2.1 Introduction

Un compilateur d'architecture (appelé aussi compilateur de comportement) tel que le système AMICAL permet d'automatiser le processus de compilation de l'architecture d'un circuit intégré à partir d'une spécification comportementale[CT90, MPC90, MPC88]. Ce processus comporte une série de transformations qui permettent de passer d'un niveau comportemental à un niveau plus bas, soit le niveau structurel, tout en satisfaisant un

certain nombre de contraintes [All90]. Cette architecture générée doit tenir compte des outils et des méthodologies qui seront utilisés pour la conception logique et physique.

L'utilisation des compilateurs d'architecture en général, et d'AMICAL en particulier, présente plusieurs avantages et doit répondre à un certain nombre de contraintes :

1- L'utilisation d'un outil logiciel de synthèse comportementale va permettre aux concepteurs d'effectuer une exploration architecturale plus approfondie avant de choisir la solution présentant le meilleur compromis entre les critères physiques (surface, vitesse, consommation, etc.);

2- Le temps de conception global va se trouver fortement réduit et simplifié, notamment par la garantie de la fonctionnalité, assurée et vérifiée par l'outil durant toutes les étapes de la synthèse, ce qui va permettre d'éviter les aller-retour coûteux en termes de temps, entre les différents niveaux d'abstraction. De plus, la durée du cycle de conception va être influencée par la notion de réutilisation de composants existants ou issus d'une synthèse précédente.

3- Le fait de travailler à un niveau architectural, va permettre d'ouvrir le monde de la conception de circuits aux concepteurs de systèmes. Actuellement, les domaines de conception de systèmes et de circuits sont séparés. D'un côté, on trouve les concepteurs de systèmes qui utilisent des méthodes et des outils pour la spécification et la conception, basés sur des langages orientés logiciel tels que SDL, ESTELLE [ISO87], LOTOS [ISO89], ESTEREL[BC84], etc. De l'autre côté, on trouve les concepteurs de VLSI qui utilisent des méthodes et des outils, pour la spécification et la conception de circuits, basés sur des langages de description de matériel (dominés par VHDL, VERILOG). Ainsi, les circuits sont généralement spécifiés du côté système pour être ensuite réalisés par les concepteurs de VLSI. Ce sont les outils de synthèse de haut niveau qui vont relier ces deux mondes.

La synthèse architecturale, objet principal de la compilation d'architecture, se compose de deux tâches principales : l'ordonnancement et l'allocation.

- L'ordonnancement analyse les dépendances entre les opérations pour extraire leur parallélisme et affecter les opérations aux transitions. Il réalise généralement un compromis entre le degré de parallélisme et le nombre de transitions.

- L'allocation fait correspondre les éléments d'une description comportementale à des ressources matérielles. Les ressources sont les unités fonctionnelles (UFs), les registres, les bus, les commutateurs, les multiplexeurs et les interconnexions. Celle-ci est donc divisée en trois sous-tâches permettant de réaliser l'allocation des unités fonctionnelles, l'allocation des registres et l'allocation des connexions.

Deux contraintes ont guidé le choix des algorithmes utilisés par AMICAL : *l'efficacité et la flexibilité*. L'efficacité implique la possibilité de trouver des solutions qui soient proches de la solution optimale : celle-ci est notamment assurée par la possibilité de combiner la conception interactive et la conception automatique. La flexibilité implique la possibilité de compiler plusieurs types de circuits dans plusieurs architectures cibles, ainsi que d'introduire de nouvelles fonctions ou bien d'utiliser des composants existants : l'utilisation d'une bibliothèque de fonctions externes y contribuera[KDJ94, KDJ95, JBD+95].

Ce chapitre a pour but d'essayer de donner une vue globale du système AMICAL. La section principale le constituant exposera donc les particularités propres à ce système, les notions manipulées et explicitées tout au long de cette thèse, ainsi que l'environnement auquel un utilisateur éventuel sera confronté. Ceci comprend l'interface *outil/concepteur*, l'architecture générale ciblée, les données à fournir au système et le découpage du processus de synthèse en sous-tâches successives.

2.2 Vue globale du système AMICAL

Le système AMICAL est un assistant pour la compilation architecturale. AMICAL est composé de cinq fonctions principales [Par92] (figure 2.1):

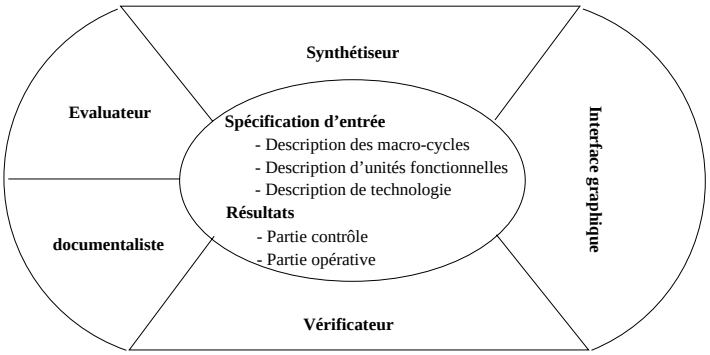


Figure 2.1: *Configuration globale du système AMICAL.*

Partant d'une description comportementale donnée en VHDL et d'une bibliothèque externe d'unités fonctionnelles, AMICAL réalise les étapes de synthèse de haut niveau et génère une architecture abstraite d'un circuit, composée d'une partie opérative et d'une partie contrôle, qui peut être facilement être adapté pour être utilisée en entrée des outils de conception et de simulation au niveau transfert de registres existants (figure 2.2).

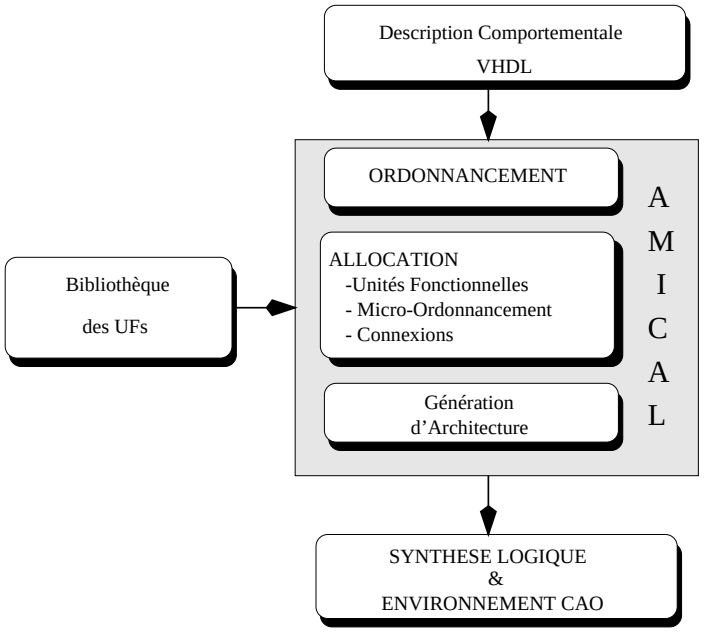


Figure 2.2: *Vue Globale d'AMICAL.*

La première étape consiste à traduire la description VHDL en une table de transitions très proche du format SOLAR [JO92, O'B93]. Le sous-ensemble VHDL accepté par AMICAL est assez large pour permettre la description d'applications industrielles[PFea95]. AMICAL réalise les étapes classiques d'un compilateur de comportement, soient l'ordonnancement et l'allocation.

AMICAL se présente comme un environnement pour la compilation d'architecture, au sein duquel la synthèse peut se faire manuellement, automatiquement, interactivement ou en combinant ces trois modes. En mode automatique, les étapes de la synthèse sont enchaînées sans intervention du concepteur. En mode interactif, les tâches sont exécutées itération par itération à l'initiative du concepteur. Ce mode permet à ce dernier, de suivre l'évolution du processus de synthèse et d'intervenir à tout moment en passant en mode manuel, pour modifier les décisions des algorithmes de synthèse. En mode manuel, le concepteur a en charge tout le processus de la synthèse. Dans ce cas, le système agit en tant qu'assistant; il vérifie la cohérence des décisions prises et ne permet que les opérations correctes. En mode manuel, le concepteur est responsable de l'efficacité du résultat (voir le chapitre 5).

AMICAL est un assistant pour la compilation architecturale. Il permet au concepteur de guider la synthèse à travers une interface graphique. La figure 2.3 montre une vue du système AMICAL pendant la synthèse. Cette vue est composée de quatre fenêtres dont trois sont permanentes pendant tout le processus de synthèse.

La fenêtre de droite montre un exemple de description VHDL, elle ne fait pas partie des trois fenêtres d'AMICAL. Les deux fenêtres de gauche (en haut) montrent le résultat de la compilation. La fenêtre du haut expose le résultat de l'ordonnancement sous la forme d'une table de transitions. La fenêtre du milieu montre la partie opérative; ici le résultat d'une allocation exécutée automatiquement est représenté. La fenêtre du bas est utilisée comme une fenêtre d'information et de dialogue, cette fenêtre d'information permet de suivre la synthèse en affichant la commande en cours et toute erreur rencontrée. C'est elle qui offre à l'utilisateur AMICAL la flexibilité de passer d'un mode à un autre.

AMICAL maintient des liens entre les différents aspects de la description. Par exemple,

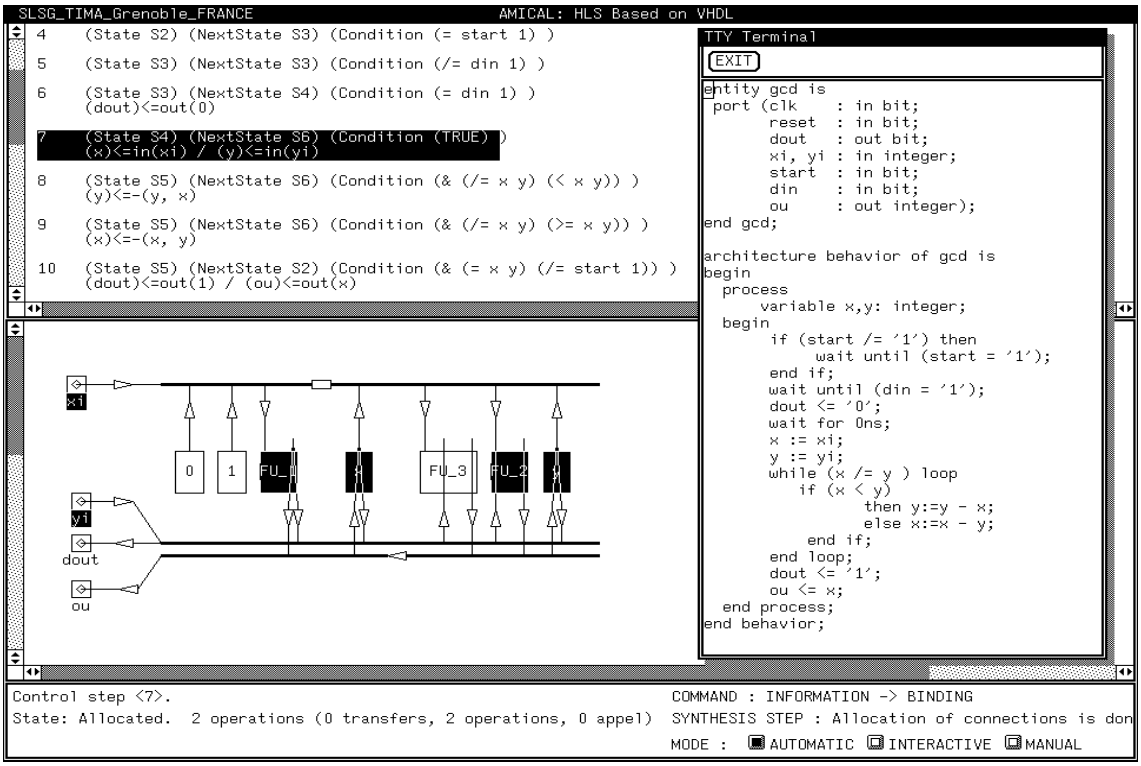


Figure 2.3: *Vue d'ensemble du système AMICAL.*

dans la figure 2.3, le système montre la correspondance entre la table de transitions et la partie opérative générée (les ressources nécessaires à l'exécution de la transition 7).

2.2.1 Architecture cible d'AMICAL

L'architecture cible d'AMICAL est un modèle général composé d'une partie contrôle, d'un ensemble d'unités fonctionnelles et d'un réseau de communication. Les deux dernières parties constituent la partie opérative. Le schéma de la figure 2.4 montre une telle architecture.

Cette architecture est synchrone, parallèle et hétérogène. Elle est synchrone grâce au contrôleur qui ordonne le séquencement des opérations exécutées par les unités fonctionnelles et le réseau de communication. L'organisation de la synchronisation est "*virtuelle*". L'hypothèse faite est qu'à chaque cycle de base une nouvelle commande est envoyée à la partie opérative. La commande inclut l'activation de plusieurs chemins de communication (*transferts*).

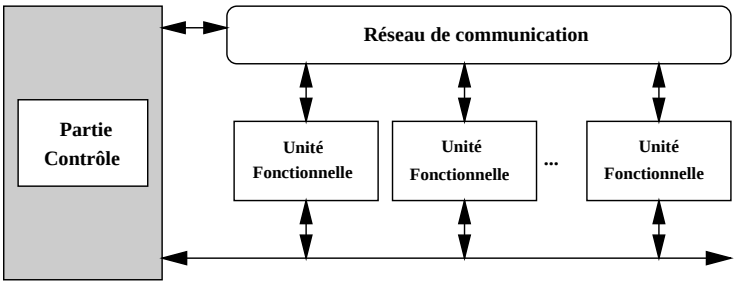


Figure 2.4: *Architecture cible d'AMICAL.*

L'architecture est parallèle; elle doit inclure plusieurs unités fonctionnelles qui peuvent s'exécuter en parallèle. L'ordonnancement, qui fixe les étapes de contrôle, est effectué automatiquement en début de synthèse et favorise un parallélisme maximum. Cependant, par des interventions manuelles, l'utilisateur pourra être à même de modifier cet ordonnancement pour un parallélisme moindre.

L'architecture est hétérogène puisqu'elle permet l'utilisation de composants produits par d'autres environnements de conception et vice versa.

2.2.2 Spécification d'entrée

AMICAL utilise trois types de spécifications en entrée:

- une spécification comportementale en VHDL;
- une bibliothèque de fonctions externes qui contient des unités fonctionnelles standards et des unités fonctionnelles privées définies spécialement par le concepteur. Les caractéristiques diverses d'une unité fonctionnelle sont décrites dans cette bibliothèque.
- une description technologique qui spécifie les contraintes, les tailles des composants et les paramètres de consommations, de délai maximal des composants. Les contraintes spécifient des limites de surface et de vitesse que la structure synthétisée doit respecter. Les tailles des composants et les paramètres permettent à AMICAL d'estimer la performance de la structure synthétisée. Cette partie est détaillée dans le chapitre 5.

2.2.2.1 *Spécification comportementale*

Comme mentionné plus haut, le processus de compilation part d'une spécification comportementale utilisant un sous-ensemble de VHDL donné. Cette description consiste en une paire *Entity/Architecture*, où l'architecture est limitée à un seul processus. Le sous-ensemble VHDL accepté est suffisamment large pour permettre la description de circuits même complexes, puisqu'il contient presque toutes les instructions séquentielles de VHDL [Std87]: les boucles, les branchements, les instructions de coupure de séquence (*exit*) et les appels de fonctions et de procédures. Ceci permet de décrire de grands exemples complexes et réels. De plus, l'utilisation de paquetages et de procédures VHDL permet d'importer les blocs existants dans une description comportementale (voir le chapitre 3).

L'entité est restreinte à la déclaration des signaux d'entrées/sorties, ceci étant considéré comme suffisant pour les besoins de la synthèse. La partie déclarative de l'architecture contient seulement les déclarations des signaux, des constantes, des types et des variables. Les instructions structurelles, ainsi que celles qui pourraient être utilisées pour la simulation, telles que *configuration*, *alias*, *file* et *composent*, ne sont pas incluses à ce niveau. La partie instruction de l'architecture contient la description d'un seul processus pouvant inclure la définition de fonctions ainsi que de procédures.

Un exemple typique d'une description comportementale est montré dans la figure 2.5.

La figure 2.5 représente la description d'un tri à bulles (*bubble-sort*) [BL90], un algorithme pris comme exemple pour illustrer le sous-ensemble de VHDL accepté par AMICAL (cf. chapitre 3 et annexe A.1). L'algorithme commence par remplir un tableau de 255 entiers lus d'une source externe et selon un protocole bien fixe (procédure *fillram*). La méthode du tri à bulles est alors utilisée pour ordonner les éléments du tableau dans l'ordre croissant. Finalement, le résultat est récupéré en sortie selon un protocole fixe (défini par la procédure *emptyram*).

```
1   Entity Bubble is
2       port (  reset, clk, start : IN BIT;
3               ackout, validin   : IN BIT;
4               datain            : IN INTEGER;
5               ackin, validout   : OUT BIT;
6               dataout           : OUT INTEGER);
7   end Bubble;
8   Architecture Behavior of Bubble is
9   Type memory is array(0 to 255) of INTEGER;
10  Begin
11      Bubble_sort: PROCESS
12      variable i, k, iter, size, t1, t2, t3, t4: INTEGER;
13      variable ram: memory;
14          Function get255 return integer is
15          Begin
16              return(255);
17          end get255;
18          Function decr2 (in1:in integer) return integer is
19          variable res1: integer;
20          Begin
21              res1 := in1 - 2;
22              return res1;
23          end decr2;
24          Procedure fillram is
25          Begin
26              i := 0; size := get255;
27              while (i <= size) loop
28                      wait until (validin = '1');
29                      t1 := datain; ackin <= '1';
30                      wait until (validin = '0');
31                      ram(i) := t1; ackin <= '0'; i := i+1;
32              end loop;
33          end fillram;
34          Procedure emptyram is
35          Begin
36              i := 0; size := get255;
37              while (i <= size) loop
38                      if (ackout /= '0') then wait until (ackout = '0'); end if;
39                      t1 := ram(i); validout <= '1'; dataout <= t1;
40                      wait until (ackout = '1');
41                      validout <= '0'; i := i+1;
42              end loop;
43          end emptyram;
44      Begin
45          if (start /= '1') then wait until (start = '1'); end if;
46          fillram; i := 1;
47          while i <= size loop
48              iter := size;
49              while (iter >= i) loop
50                      t1 := iter-1; t2 := decr2(iter); t3 := ram(t1); t4 := ram(t2); iter := t1;
51                      if (t3 < t4) then ram(t1) := t4; ram(t2) := t3; end if;
52              end loop;
53              i := i+1;
54          end loop;
55          emptyram;
56      end PROCESS Bubble_sort;
57  end Behavior;
```

Figure 2.5: *Un exemple d'une description comportementale.*

2.2.2.2 *Bibliothèque des unités fonctionnelles*

Le système AMICAL utilise une bibliothèque d'unités fonctionnelles (*Functional Units*: UFs). Les unités fonctionnelles doivent être fournies par l'utilisateur d'AMICAL, et permettent plus de flexibilité pour l'utilisation des blocs existants.

La figure 2.6 montre un exemple de fichier de bibliothèque qui contient une liste d'unités fonctionnelles nécessaires pour la compilation de l'algorithme du tri à bulles (*Bubble-sort*).

```
1   (Library Bub
2       (Path ./Library)
3       (Functional_Unit
4           ram (Operator read write)
5           SUB (Operator - decr2 get255)
6           ADD (Operator + get255)
7           ALU3 (Operator + - decr2 get255)
8       )
9   )
```

Figure 2.6: *Un exemple du fichier de bibliothèque (.lib) pour l'algorithme de tri à bulles.*

Ce fichier indique:

- le chemin pour la bibliothèque d'unités fonctionnelles (mot clé *Path*); le répertoire indiqué regroupe divers fichiers (un par unité fonctionnelle) décrivant le schéma d'exécution de chacune des UFs par les signaux de contrôle et les transferts correspondant à chaque opération.
- la liste des unités fonctionnelles (mot clé *Functional_Unit*) et les opérations qu'elles peuvent exécuter.

Pour chaque unité fonctionnelle, la liste des opérations qu'elle peut exécuter correspond à la liste précédée par le mot clé *Operator*. Les caractéristiques de chaque unité fonctionnelle sont quant à elles décrites dans un autre fichier (fichier du répertoire *Library* ici); ce fichier est nommé du nom de l'unité fonctionnelle suivi de ".fu"; (figure 2.7(d) correspondant à Ram.fu).

2.2.2.3 *Description des unités fonctionnelles*

Une unité fonctionnelle (UF) peut exécuter une ou plusieurs opérations; ces opérations comprennent aussi bien les opérations standards (addition, multiplication, opérations logiques, etc.), que toute nouvelle opération programmée par le concepteur. L'unité fonctionnelle peut être un bloc complexe tel qu'une mémoire cache, une unité d'entrée/sortie, etc.

Une UF peut être appelée à l'intérieur de la spécification comportementale par un appel de procédure afin de réaliser une opération donnée. L'utilisation de bibliothèques d'unités fonctionnelles offre la possibilité d'utiliser des circuits réalisés préalablement comme des blocs pour de nouveaux systèmes [KDJ94].

Une unité fonctionnelle peut être spécifiée à des niveaux d'abstraction différents comme le montre la figure 2.7.

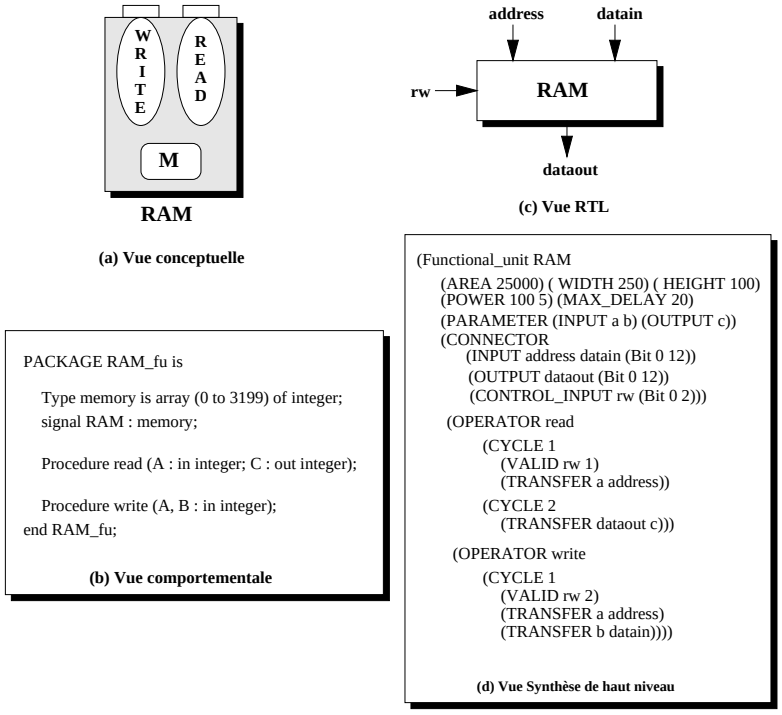


Figure 2.7: *Différents types d'abstraction d'une unité fonctionnelle.*

Cet exemple décrit une mémoire appelée *RAM*. Du point de vue conceptuel, la mémoire est un objet capable d'exécuter deux opérations : une écriture (*Write*) et une lecture (*Read*) qui manipulent une donnée commune (*M*, figure 2.7(a)). Au niveau transfert de registres (figure 2.7(b)), l'UF peut être décrite en VHDL dans un paquetage qui inclut deux procédures ayant accès à un objet commun (un signal global) *array RAM*. Chaque procédure spécifie l'exécution d'une opération. La figure 2.7(c) montre une vue externe de la mémoire. Elle a deux ports d'entrée principaux (*datain* et *address*), une sortie (*dataout*) et un signal de commande (*rw*) pour la sélection de la procédure à exécuter. La vue synthèse de haut niveau (figure 2.7(d)) englobe les vues *comportementale* et *transfert*

de registres. Elle comprend:

- L'interface de l'unité fonctionnelle.
- Les paramètres d'appel de l'unité fonctionnelle (appel de procédures).
- L'ensemble des opérations exécutées par l'unité fonctionnelle.
- Le micro-ordonnancement pour chaque opération.

Dans la spécification d'entrée d'AMICAL, une unité fonctionnelle est invoquée par un appel de procédure ou de fonction. Un des traits majeurs d'AMICAL est que la bibliothèque d'unités fonctionnelles est extensible, permettant ainsi l'utilisation de blocs déjà synthétisés. L'annexe A.2 fournit un aperçu de la grammaire utilisée pour l'abstraction d'une unité fonctionnelle lisible par le système (voir figure 2.7(d)).

2.2.2.4 *Description technologique*

Cette description contient les tailles de composants, les paramètres de performance (la consommation et le délai maximal), et les contraintes. Elles servent respectivement à estimer la performance de la structure synthétisée et donc à évaluer sa qualité. Un problème critique dans la synthèse est l'estimation de la performance en terme de surface, vitesse et consommation à partir de la structure du niveau transfert de registres. Le modèle d'évaluation sera discuté dans le chapitre 5.

La figure 2.8 donne un exemple de fichier contenant les paramètres technologiques :

La syntaxe de description d'un tel fichier est donnée en annexe A.3.

2.2.3 Etapes de synthèse

Après la compilation de la spécification d'entrée, AMICAL doit exécuter un certain nombre de tâches afin de générer l'architecture abstraite. Les étapes apparaissent dans la figure 2.9:

```
1   ( TECHNOLOGY_FILE
2       ( PARAMETER
3           ( CONSTANT_REGISTER ( WIDTH 10)( HEIGHT 50.0)
                ( AREA 0)( POWER 100 2)( MAX_DELAY 1))
4           ( FLAG_REGISTER ( WIDTH 10)( HEIGHT 50.0)
                ( AREA 500)( POWER 200 2)( MAX_DELAY 10))
5           ( VARIABLE_REGISTER ( WIDTH 10)( HEIGHT 30.0)
                ( AREA 300)( POWER 200 2)( MAX_DELAY 7))
6           ( EXTERNAL_REGISTER ( WIDTH 10)( HEIGHT 20.0)
                ( AREA 200)( POWER 100 2)( MAX_DELAY 5))
7           ( SWITCH ( WIDTH 5)( HEIGHT 10.0)
                ( AREA 50)( POWER 100 2)( MAX_DELAY 1))
8           ( MUX ( WIDTH 5)( HEIGHT 10.0)
                ( AREA 50)( POWER 100 2)( MAX_DELAY 1))
9           ( BUS ( HEIGHT 2.0) ( POWER 10 2)( MAX_DELAY 1))
10      )
11      ( CONSTRAINT
12          ( MAX_MUX 50 ( WEIGHT 1))
13          ( MAX_MICRO 100 ( WEIGHT 1))
14          ( MAX_BUS_CHANNEL 9 ( WEIGHT 10))
15          ( MAX_FU 10 ( WEIGHT 5))
16          ( MAX_SWITCH 90 ( WEIGHT 5))
17          ( MAX_WIDTH 500.0 ( WEIGHT 0))
18          ( MAX_HEIGHT 100.0 ( WEIGHT 0))
19          ( MAX_AREA 50000.0 ( WEIGHT 10))
20          ( MAX_POWER 2000 ( WEIGHT 5))
21      )
22      ( FACTOR
23          ( FDATAPATH 1.2)
24          ( FCONTROLLER 0.44)
25          ( FCIRCUIT 1.3)
26          ( TR2AREA 0.0002)
27          ( SWPOWER 1.3)
28      )
29  )
```

Figure 2.8: *Description technologique*

2.2.3.1 *Ordonnancement*

AMICAL utilise un algorithme d'ordonnancement à boucles dynamiques (*Dynamic Loop Scheduling*) [ROA94]. Cet algorithme est adapté à des circuits dominés par le flux de contrôle et décrits en VHDL. C'est une approche améliorée de l'algorithme à base de chemins (*path-based approach*) proposé par Camposano [Cam91] et Fisher [Fis81]. L'outil d'ordonnancement lit la description donnée en VHDL et produit une machine d'états finis (*Finite State Machine; FSM*) sous forme de table de transitions. Chaque transition est définie par un état courant, une condition, un état suivant et un ensemble d'opérations à exécuter. Cette machine d'états finis est représentée selon le modèle du graphe de Mealy. Les transitions portent le nom de *macro-cycles*. Un macro-cycle peut utiliser plusieurs cycles de base (cycles d'horloge) pour l'exécution de ses opérations.

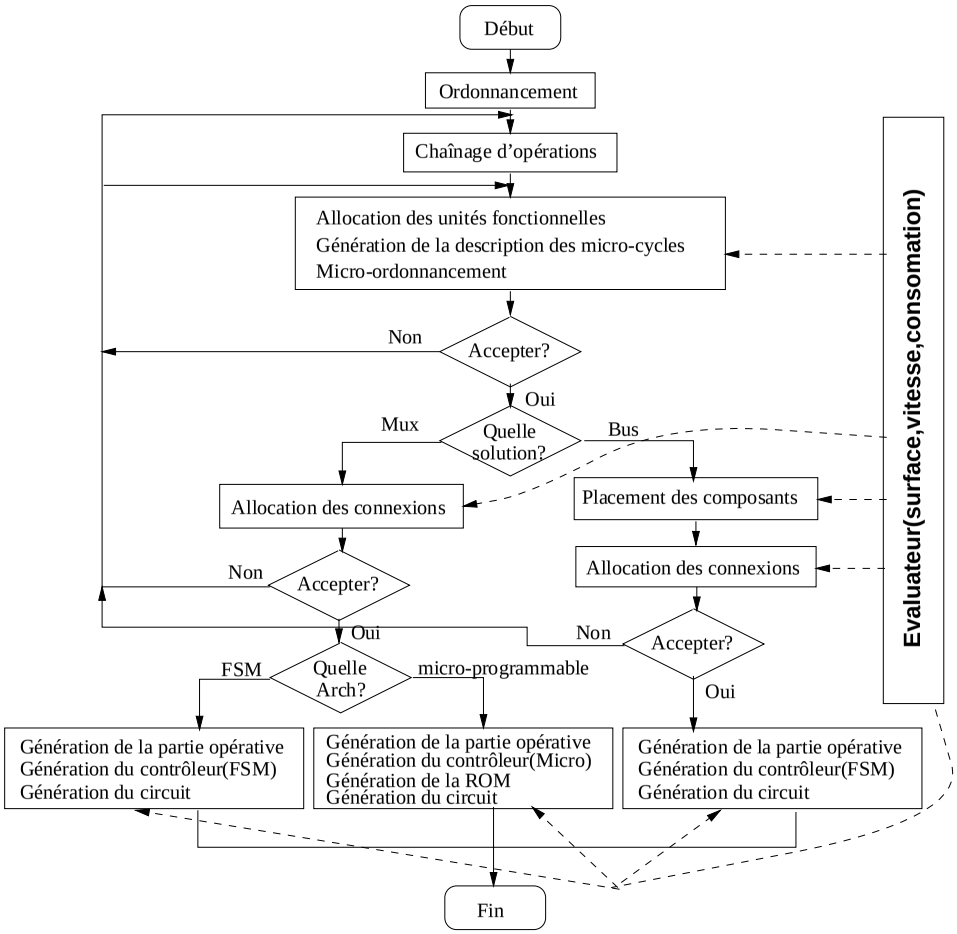


Figure 2.9: *Flot d'exécution global du synthétiseur*

2.2.3.2 *Chaînage d'opérations*

Par défaut, les opérations d'un macro-cycle sont exécutées en parallèle en utilisant des unités fonctionnelles différentes. A chaque opération est associé un délai de calcul. Dans un macro-cycle, l'opération la plus lente détermine donc la durée du macro-cycle. Le chaînage permet d'exécuter plusieurs opérations d'un même macro-cycle sur une seule unité fonctionnelle. Dans ce cas, la durée du macro-cycle est déterminée soit par la somme de délais des opérations chaînées, soit par le délai de l'opération la plus lente. Dans certain cas, le chaînage d'opérations permet de diminuer la surface (nombre d'unités fonctionnelles) sans diminuer la vitesse. En général le chaînage permet de réaliser des compromis entre la vitesse et la surface.

Un exemple de chaînage est donné par la figure 2.10. Le macro-cycle de la figure 2.10(a)

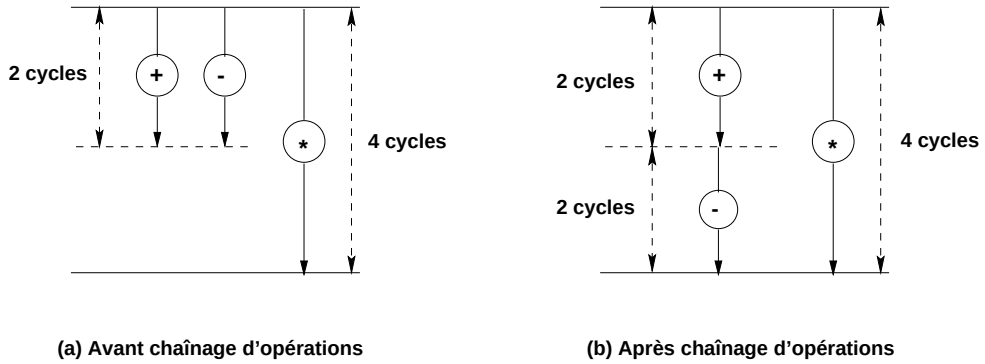


Figure 2.10: *Un exemple du chaînage d'opération*

utilise trois opérations: une addition (+), une soustraction (-) et une multiplication (*). On suppose que l'exécution de ces opérations nécessite, respectivement, 2, 2 et 4 cycles. Dans ce cas, trois unités fonctionnelles, un additionneur, un soustracteur et un multiplieur peuvent être allouées. Après le chaînage, (voir la figure 2.10(b)), deux unités fonctionnelles, une pour exécuter deux opérations chaînées et l'autre pour exécuter la multiplication peuvent être allouées.

Dans ce cas, le chaînage d'opérations permet de diminuer à la fois le coût en unités fonctionnelles et le coût en connexions sans diminuer la vitesse.

Le chaînage est manuel. A chaque itération le concepteur sélectionne deux opérations à chaîner. Ces deux opérations doivent appartenir au même macro-cycle. Un chaînage n'est accepté que si les deux conditions suivantes sont vérifiées:

- Le chaînage ne doit pas induire de conflits de données;
- La bibliothèque contient au moins une unité fonctionnelle capable d'exécuter les opérations à chaîner.

2.2.3.3 *Allocation des unités fonctionnelles*

Cette étape sélectionne un ensemble d'unités fonctionnelles qui peuvent exécuter les opérations parallèles et les opérations chaînées, et réalise les liaisons entre les opérations

et les unités fonctionnelles allouées. Le problème principal est la minimisation du coût total des unités fonctionnelles.

Les trois modes de fonctionnement (automatique, manuel, interactif) sont autorisés et peuvent être combinés.

L'allocation d'unités fonctionnelles est suivie de deux tâches: la transformation des opérations en transferts et le micro-ordonnancement initial.

En mode automatique, AMICAL associe un coût aux opérations et considère la surface individuelle d'une unité fonctionnelle comme son coût. A chaque opération, on associe la liste d'unités fonctionnelles capable de l'exécuter. La meilleure unité fonctionnelle de cette liste est sélectionnée.

Le mode interactif permet au concepteur de suivre l'algorithme itération par itération. A chaque itération, une nouvelle unité fonctionnelle est allouée.

En mode manuel, le concepteur sélectionne une unité fonctionnelle et un ensemble d'opérations pouvant être liées à cette unité fonctionnelle. Cette sélection se fait de manière interactive. Ces interactions sont expliquées plus en détail dans le chapitre 5.

2.2.3.4 *Micro-ordonnancement*

Cette étape génère le schéma d'exécution de chaque opération selon l'unité fonctionnelle utilisée. Chaque opération est décomposée en un ensemble de transferts entre registres selon la définition des unités fonctionnelles. Les transferts sont ordonnés en *micro-cycles*. Chaque micro-cycle contient un ensemble de transferts parallèles qui s'exécutent en un seul cycle de base.

Le but de cette étape est de minimiser le nombre de transferts parallèles à chaque micro-cycle en permettant de diminuer le coût des connexions.

Le résultat de cette étape est une description au niveau du micro-cycle. Chaque micro-cycle est composé d'un ensemble de transferts pouvant s'exécuter en parallèle.

2.2.3.5 *Placement des composants*

Cette étape place les registres et les unités fonctionnelles côte à côte de façon à réduire le plus possible les pistes de bus (connexions). Tant qu'AMICAL utilise une architecture cible à base de bus, le placement des registres et des unités fonctionnelles a une influence sur le modèle topologique.

2.2.3.6 *Allocation des connexions*

Le but de cette étape est de générer des chemins de connexions pour exécuter tous les transferts tout en minimisant le coût de ces connexions. Un chemin de connexions est une série d'éléments interconnectés permettant de réaliser un transfert. Dans l'architecture cible d'AMICAL, deux structures de connexions sont disponibles: une basée sur les bus et les commutateurs et l'autre basée sur les multiplexeurs. Pour ces deux solutions, les chemins de connexions correspondant aux transferts parallèles doivent être indépendants afin de préserver le comportement.

Pour la solution basée sur les bus et les commutateurs, les trois modes sont autorisés. Pour la solution basée sur les multiplexeurs, le mode automatique seulement est permis.

2.2.3.7 *Génération d'architecture*

AMICAL produit une architecture abstraite composée de deux sous-systèmes : une partie contrôle et une partie opérative. Cette architecture est donnée sous une forme intermédiaire nommée SOLAR [JO92, O'B93] par deux fichiers décrivant les deux parties mentionnées ci-dessus et un fichier décrivant l'interconnexion entre elles (circuit global). Les détails sont décrits dans le chapitre 4.

2.3 Conclusion

Dans ce chapitre, une vue globale du système AMICAL a été présentée, rappelant les spécifications d'entrée nécessaires, les étapes de synthèse et les modalités d'exécution de celles-ci.

Partant d'une description comportementale donnée en VHDL et d'une bibliothèque externe d'unités fonctionnelles, AMICAL génère une architecture abstraite au niveau transfert de registres composée d'une partie opérative et d'une partie contrôle.

Afin de manipuler des circuits très complexes, AMICAL permet l'utilisation d'unités fonctionnelles complexes ainsi que la réutilisation de composants existants pour la mise au point de nouveaux circuits.

Son interface graphique contribue à rendre cet outil attrayant et autorise la notion d'interactivité : le concepteur ne se contente pas de pousser un bouton, il participe aux diverses étapes de synthèse s'il en éprouve le besoin.

Les modifications apportées au système initial, qui ont fait l'objet des travaux relatifs à cette thèse, seront présentées en détails au cours des chapitres suivants.

CHAPITRE 3

Modèles architecturaux d'AMICAL

Chapitre 3

Modèles architecturaux d'AMICAL[1]

La modélisation des architectures est un sujet important dans le domaine de la synthèse de haut niveau. Les étapes de synthèse présentent des étapes de raffinement ou de transformation d'un modèle de haut niveau à un modèle de bas niveau. Selon les étapes, l'algorithme de synthèse manipule des modèles d'architecture intermédiaires, transformant un modèle en un suivant, plus proche de la réalisation finale.

Ce chapitre présente en détail ces différents modèles architecturaux, du niveau comportemental au niveau transfert de registres. Les caractéristiques principales de la synthèse à l'aide d'AMICAL, à savoir l'entrelacement des étapes d'allocation et d'ordonnancement, sont décrites et explorées. Leur aboutissement est une architecture composée d'un contrôleur et d'une partie opérative. Cette architecture, dont la description finale est donnée en VHDL, peut se présenter sous diverses formes et autorise l'utilisation d'unités fonctionnelles complexes (issues, par exemple, de synthèse précédentes) et réutilisables.

[1]Ce chapitre a été écrit en collaboration avec P.Kission

3.1 Introduction

Le flot de synthèse pour la génération d'une description au niveau transfert de registres à partir d'une description comportementale inclut plusieurs étapes. Ces tâches essentielles concernent l'ordonnancement et l'allocation. Elles sont dans certains cas décomposées en sous-tâches. L'introduction de diverses opérations met à jour des modèles architecturaux intermédiaires [WHHY90].

Dans ce chapitre, les différents modèles architecturaux intervenant lors des étapes de synthèse du système AMICAL sont détaillés. Le flot de synthèse correspondant comporte deux étapes d'ordonnancement et deux étapes d'allocation exécutées de manière entrelacée : en d'autres termes, une étape d'ordonnancement (resp. d'allocation) est toujours consécutive à une étape d'allocation (resp. d'ordonnancement).

Toute étape de synthèse consiste à engendrer à partir d'une description initiale une description d'abstraction moindre. Ainsi la synthèse comportementale produit une description au niveau transfert de registres définissant une architecture, et cela à partir d'un algorithme. Les tâches effectuées lors d'une telle synthèse correspondent alors à des raffinements successifs de la description initiale.

La figure 3.1 illustre la notion de synthèse comportementale réalisée grâce à AMICAL.

La description de départ est une description VHDL composée d'un processus principal qui, via des appels de procédures et de fonctions, utilise de manière implicite un ensemble de ressources (unité d'entrée-sortie : E/S, unité fonctionnelle : UF, co-processseur : co-proc), ensemble de ressources situé dans une bibliothèque externe de composants. Ces composants de la description comportementale deviendront des éléments de la partie opérative générée à l'issue de la synthèse. L'architecture obtenue, décrite au niveau transfert de registres, comporte également un contrôleur relié à la partie opérative par l'intermédiaire d'un réseau d'interconnection. Le processus de synthèse comporte plusieurs étapes successives de raffinement qui pourront s'exécuter manuellement ou de façon automatique. A chaque itération correspond un modèle architectural intermédiaire précisant de nouveaux détails d'implémentation jusqu'à l'obtention de l'architecture finale.

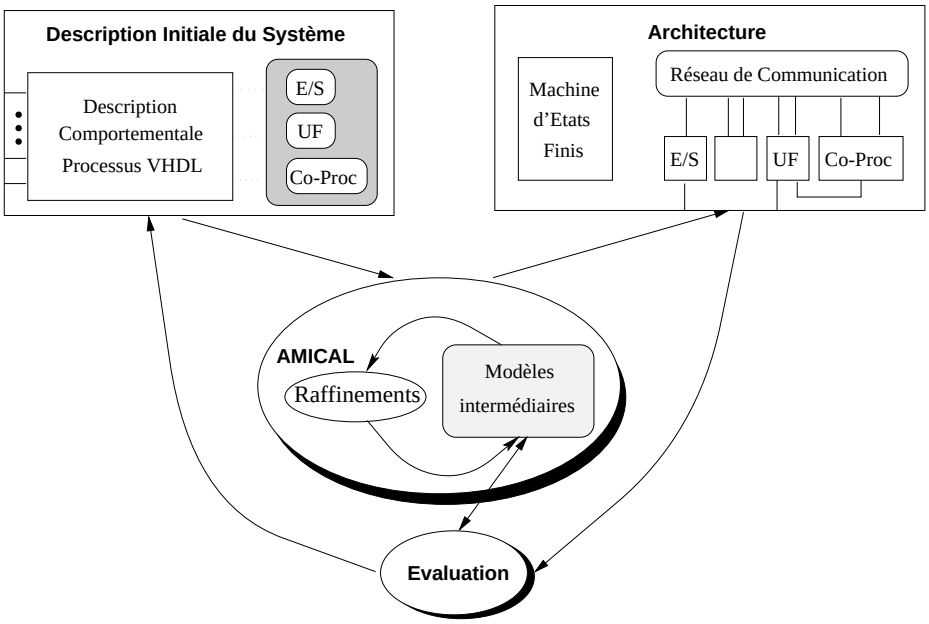


Figure 3.1: *Processus de synthèse par raffinement successifs.*

Ce chapitre décrit chaque étape de la synthèse architecturale à l'aide d'AMICAL.

La première section permettra de se faire une idée précise du flot de synthèse global mis en œuvre. Chaque tâche successive sera brièvement décrite et illustrée.

Les sections suivantes décriront précisément chaque partie spécifique et les modèles architecturaux intermédiaires manipulés par le système pour atteindre l'objectif final : la génération d'une architecture au niveau transfert de registres sous la forme d'une description VHDL, composée d'un contrôleur et d'une partie opérative associée.

3.2 Flot de synthèse au sein d'AMICAL

La synthèse avec AMICAL débute à partir de la donnée d'une description comportementale décrite en VHDL et basée sur un sous ensemble d'instructions suffisant pour permettre de décrire des comportement complexes (voir section 3.3).

Utilisant une bibliothèque de composants externe, le système va générer en se basant sur la description initiale, un certain nombre de description intermédiaires qui vont correspondre à des étapes de raffinement successives.

Le flot global de synthèse est illustré sur la figure 3.2

A la suite de la compilation et donc de la génération du graphe de contrôle, et comme introduites au chapitre 2 , les étapes de synthèse mises en œuvre par AMICAL sont : le macro-ordonnancement, l'allocation d'unités fonctionnelles, le micro-ordonnancement et l'allocation de connexions. Ces étapes constituent les points principaux du processus général de raffinement.

Chaque étape représente une tâche indépendante. Les objets manipulés par les divers modèles architecturaux sont présentés dans le tableau de synthèse (3.1).

Modèle Architectural	Organisation	Base de Temps	Objets Manipulés
Description Comportementale d'Entrée(VHDL)	processus Ensemble d'UFs explicité	Etape de Calcul	Entrées/Sorties, Signaux,Variables, sous-programmes, Opérations,...
Graphe de Contrôle	Graphe de Flot de Contrôle et de Données	Etape de Contrôle	Actions, Tests, Conditions
Machine d'Etats Finis Comportementale	Machine d'Etats Finis	Macro-cycle	Etats, Conditions Actions
Machine d'Etats Finis Comportementale Reliée	Machine d'Etats Finis + Ressources Allouées	Macro-cycle	Etats,Conditions Actions,UFs, Registres
Machine d'Etats Finis au Niveau RT	Machine d'Etats Finis + Ressources Allouées	Micro-cycle	Etats,Conditions Transferts,UFs, Registres, Eléments de Connexion
Architecture Abstraite	Contrôleur + Chemin de Données	Micro-cycle	Instances, Connexions,...
Description au Niveau RT	Contrôleur + Chemin de Données	Cycle d'horloge	Instances, Machine d'Etats Finis

Table 3.1: *Modèles architecturaux*

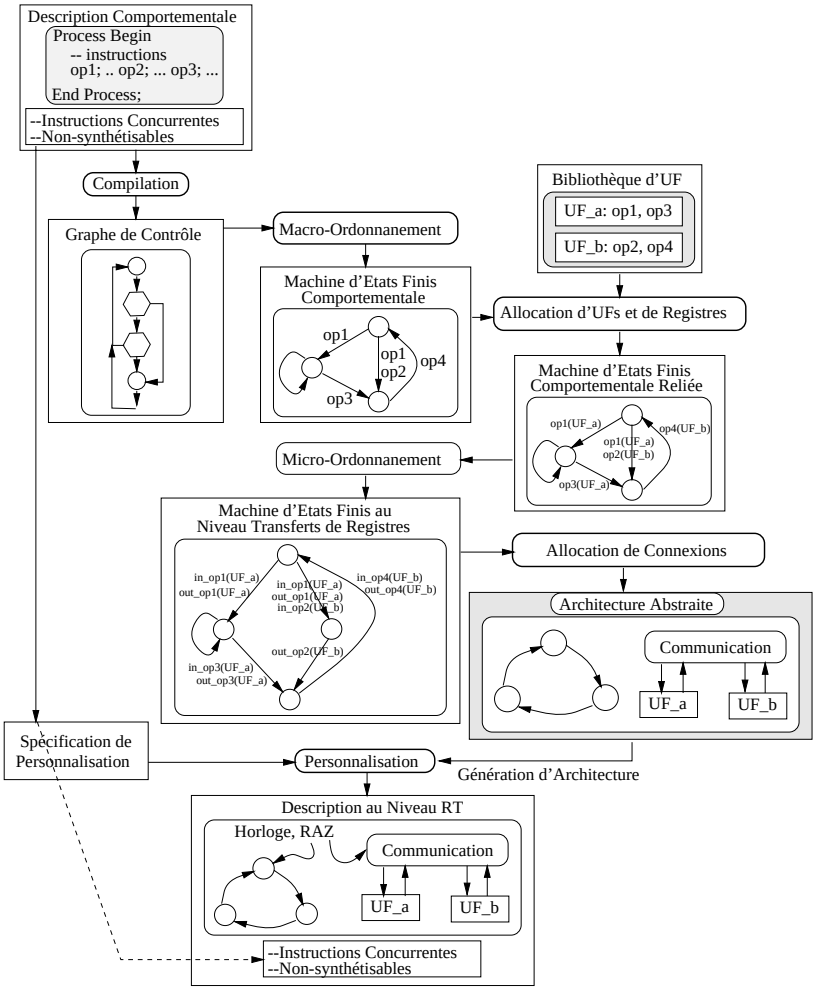


Figure 3.2: *Flot de synthèse.*

La *machine d'états finis comportementale* construite après l'ordonnancement du graphe de contrôle se présente sous la forme d'une table de transition. Elle est assimilable à un macro-contrôleur, les opérations décrites pouvant être de complexité illimitée et pouvant s'exécuter en un nombre fixé de cycle d'horloge. La structure finale visée, consistant en un contrôleur et une partie opérative, commence à se préciser.

Comme son nom l'indique, l'allocation des ressources produit une *machine d'états finis comportementale avec ressources*. A ce stade, chaque opération est associée à une unité fonctionnelle présente dans la bibliothèque externe de composants.

La *machine d'états finis au niveau transfert de registre* issue du micro-ordonnancement est proche de la description du contrôleur final. Chaque opération est décomposée en un

ensemble de transferts élémentaires nécessaire à son exécution explicite. Le découpage de l'opération, présent dans la description de l'unité fonctionnelle associée, devient effectif. L'unité de temps, implicite, est le cycle de base qui va correspondre, à terme, au cycle d'horloge. La distinction *contrôleur-partie opérative* est très claire à ce stade.

L'allocation des connexions va déterminer l'*architecture abstraite*, dite telle car on ne considère qu'une structure *PO/PC (partie opérative/partie contrôle)* pure. A ce stade, toute la structure finale des *PO/PC* sont définies de manière précise : elle va correspondre exactement à l'architecture décrite en VHDL. Le modèle RTL va contenir des parties non-synthétisable tel que la synchronisation et les liens directs entre unités fonctionnelle. Le chemin de données comporte toutes les unités nécessaire à son fonctionnement, y compris le réseau d'interconnection.

La dernière étape, dite étape de *personnalisation*, va autoriser le concepteur à rajouter des connexions reliées à des blocs concurrents au circuit synthétisé (ayant par exemple déjà fait l'objet d'une synthèse avec AMICAL); c'est à ce moment de la synthèse que le contrôleur et la partie opérative seront reliés à un bloc de synchronisation explicite.

Une description détaillée de ces étapes de synthèse est disponible en annexe B.

Les sections suivantes vont permettre de plonger en détail au sein des caractéristiques de chaque étape, à commencer par l'étape initiale : la description du système à synthétiser en VHDL comportemental et la donnée d'une bibliothèque externe de composants.

3.3 Descriptions d'Entrée au Niveau Comportemental

La synthèse utilise deux principaux types d'informations et cela quelque soit le niveau d'abstraction traité. La première information est évidemment l'algorithme ou la fonction à réaliser. En second lieu, une bibliothèque de cellules est requise. En effet, toute synthèse inclut un découpage technologique; cependant selon le type de synthèse en cours, les composants de la bibliothèque utilisée seront plus ou moins complexes. Par exemple, pour une synthèse logique, les cellules de la bibliothèque seront des portes logiques et donc de complexité bien moindre que les unités fonctionnelles utilisées lors d'une synthèse comportementale : ces dernières peuvent par exemple exécuter des fonctions nécessitant plusieurs cycles d'horloge ou provenir d'une synthèse de haut niveau préalablement exécutée.

D'autres informations sont généralement spécifiées en entrée d'un outil de synthèse par un cahier des charges. Elles consistent en la définition des contraintes temporelles, de surface ou de consommation. Celles-ci n'interfèrent pas directement lors de la synthèse par AMICAL, mais seront utilisées pour des évaluations permettant de guider le concepteur durant le processus de synthèse.

La spécification comportementale est mise en œuvre a priori sans prendre en compte l'architecture cible; elle ne décrit que l'algorithme et ceci de manière séquentielle. Toutefois puisqu'à chaque opération une unité fonctionnelle sera allouée, l'écriture de la description comportementale peut influencer l'architecture. L'élaboration de la description comportementale et celle de la bibliothèque d'unités fonctionnelles se font de manière indissociable.

3.3.1 Description comportementale

Pour la synthèse architecturale par AMICAL, le circuit (représenté par l'algorithme) à concevoir est décrit par l'intermédiaire d'un processus VHDL standard. Selon les caractéristiques du langage de description de matériel VHDL, le processus contient un ensemble d'instructions séquentielles. Dans le cas où la description comportementale VHDL contient plus d'un processus (autres processus ou/et instructions concurrentes telles que

des instanciations ou plus simplement des instructions de type flot de données), un processus unique sera traité lors d'une session de synthèse par AMICAL. AMICAL offre cependant à l'utilisateur les moyens d'intégrer les parties non-synthétisées à la partie synthétisée au niveau transfert de registres : c'est l'étape dite de personnalisation [Moh94].

Le choix du langage VHDL présente plusieurs avantages. En premier lieu, VHDL est un langage standard et normalisé pour la description de matériels. Par conséquent, l'utilisation de VHDL pour les descriptions comportementales en entrée d'AMICAL tout comme pour ses descriptions de sortie rend AMICAL intégrable dans les environnements existants de conception assistée par ordinateur (synopsys, compass, cadence, ...).

Le sous-ensemble VHDL accepté par AMICAL pour la description comportementale comprend un éventail assez large d'instructions séquentielles pour permettre la description d'algorithmes complexes représentant des circuits de taille et de complexité conséquentes. Parmi ces instructions, on distingue les opérations conditionnelles ("**if ... then ... end if;**", "**if ... then ... else ... end if;**", "**case ... when ... end case;**"), les boucles ("**loop ... exit loop; ... end loop;**", "**while ... loop ... end loop;**"), les appels de procédures et de fonctions, les affectations de signaux et de variables, et les instructions d'attente ("**wait until ...;**", "**wait for ...;**").

Il est à noter que l'instruction VHDL "**for ... loop ... end for;**" ne fait pas partie des instructions disponibles mais qu'elle peut être exprimée à l'aide de l'instruction "**while ... loop ... end loop;**". De même nature, l'instruction "**wait on ...;**" n'est pas admise mais peut être traduite par un "**wait until ...;**" avec des tests de condition si nécessaire. Les instructions citées ci-dessus peuvent être utilisées de manière imbriquée tant qu'elles respectent la syntaxe VHDL.

La description comportementale peut faire appel à des types ou sous-programmes (procédures ou fonctions) propres à l'utilisateur. Ce dernier peut les avoir définis soit directement dans l'unité de conception secondaire VHDL qui est l'architecture, soit par l'intermédiaire d'un paquetage; en d'autres termes, AMICAL accepte toute fonction ou procédure VHDL.

La notion de procédure dans la description comportementale constitue une extension majeure pour la synthèse de haut niveau. En fait, cette caractéristique permet d'avoir une compilation extensible. Contrairement aux outils de synthèse classiques qui n'acceptent que des descriptions réalisant des opérations de base, telles que +, −, *, ..., l'appel de procédure donne accès à une forme infinie d'opérations complexes.

La description comportementale acceptée par AMICAL peut comporter plusieurs instructions d'attente dans un même processus. Il est possible de combiner une multitude d'instructions d'attente sur des signaux variés et des conditions complexes avec des boucles et des instructions de contrôle complexes. Cette caractéristique constitue l'une des principales valeurs ajoutées par rapport aux outils de synthèse au niveau transfert de registres. En effet, pour être synthétisables par les outils de synthèse au niveau transfert de registres existants, les descriptions au niveau transferts de registres doivent généralement satisfaire aux limitations imposées dans le domaine temporel au niveau de chaque processus. Ces restrictions concernent les instructions d'attente (exprimées par "wait") qui ne doivent s'appliquer qu'à un seul signal d'horloge, les instructions entre deux instructions d'attente qui s'exécutent obligatoirement pendant un seul cycle d'horloge, etc.

Faire usage de plusieurs instructions d'attente dans un même processus permet de décrire des algorithmes comprenant des protocoles d'échange de données avec le monde extérieur (par l'intermédiaire des ports ou plus localement avec des opérations concurrentes au processus traité grâce à des partages de signaux). Les figures 3.3(a) et 3.3(b) illustrent deux exemples : I et II, correspondent respectivement à un exemple typique de description comportementale, et à un extrait du code VHDL décrivant un répondeur téléphonique. L'exemple I comprend trois blocs de traitements différents : une boucle de calcul comprenant une instruction d'attente rencontrée à chaque itération, précédée d'une phase d'initialisation et de lecture des valeurs d'entrée, et suivie d'une phase de synchronisation entre le circuit et le monde extérieur pour l'envoi du résultat de calcul. Il s'agit d'un cas très courant de description comportementale. La description au niveau transfert de registres d'un tel comportement nécessiterait plusieurs processus et serait bien plus grande en taille et en complexité. L'exemple II est un extrait d'une description d'un cas réel. Il s'agit d'un répondeur téléphonique qui sera réutilisé par la suite dans ce chapitre

pour illustrer les divers modèles architecturaux rencontrés lors d'une synthèse. Il montre la combinaison d'instructions de contrôle de boucles, d'attentes et d'"exit" de boucle. Ce type de combinaison est généralement interdit au niveau transfert de registres.

```
ENTITY circuit IS
PORT(
--Déclaration des entr ées/sorties
);
END circuit;
---------------------------
ARCHITECTURE algo OF circuit IS
--Déclaration de signaux, "components"
-- et sous-programmes
BEGIN
PROCESS
--Déclaration de variables
BEGIN
[1]  --Init : Instructions d'initialisation
[2]  WHILE cond1 LOOP
[3]     --Iter_cal1 : Instructions de calcul
[4]     WAIT UNTIL cond2;
[5]     --Iter_cal2 : Instructions de calcul
[2']  END LOOP;
[6]  --Cal : Instructions fin de calcul
[7]  WAIT UNTIL cond3;
[8]  --Sortie : Instructions envoi
END PROCESS;
-- Autres instructions concurrentes,
-- y compris des instanciations
END algo;
```

(a) Example I

```
PROCESS
--Déclaration de variables
BEGIN
[01]  --E1 : Ensemble d'instructions
[02]  decrocher: LOOP
[03]     saisie_3_chiffres : LOOP
[04]        WAIT UNTIL (touche_appuyee=0);
[05]        tmp_max := fct_rem(delai_ecoule,20);
[06]        WAIT UNTIL ((delai_ecoule=tmp_max) OR
                touche_appuyee/=0 OR raccroche);
[07]        IF (raccroche OR delai_ecoule=tmp_max)
[08]        THEN EXIT decrocher;
[07']        END IF;
[09]        nouveau_chiffre := rom_mot_passe(nb_chiffres);
[10]        IF (touche_appuyee/=nouveau_chiffre)
[11]        THEN mot_passe_invalide := '1';
[10']        END IF;
[12]        nb_chiffres := nb_chiffres + 1;
[13]        IF (nb_chiffres=3)
[14]        THEN EXIT saisie_3_chiffres
[13']        END IF;
[03']     END LOOP saisie_3_chiffres;
[15]     --E2 : Ensemble d'instructions
[02']  END LOOP decrocher;
[16]  --E3 : Ensemble d'instructions;
END PROCESS;
```

(b) Example II

Figure 3.3: *Exemples de descriptions comportementales.*

3.3.2 Unités fonctionnelles et autres composants

Lors de la mise en œuvre de la partie opérative d'un circuit synchrone, une bibliothèque de modules est nécessaire. Dans le cas d'une synthèse au niveau comportemental, celle-ci doit contenir des unités fonctionnelles permettant l'exécution des opérations décrites dans la description initiale. De tels modules sont de complexité pouvant varier d'une simple fonction logique à un co-processeur. La synthèse comprend une allocation de telles unités fonctionnelles accompagnée de la définition de l'interconnexion entre elles. La communication entre unités fonctionnelles est gérée par l'intermédiaire de registres, de commutateurs (portes-trois-états), de multiplexeurs et de bus. Les caractéristiques de tels composants et leur allocation sont influencées et définies par l'architecture ciblée.

AMICAL repose sur l'utilisation de deux types de composants :

- un ensemble de composants prédéfinis : ces éléments correspondent aux éléments de base de l'architecture, tels que les registres, les multiplexeurs, ...
- une bibliothèque extensible d'unités fonctionnelles.

Les éléments du premier type sont fonctionnellement figés; les différents éléments de connexion instanciés sont connus par le système et leurs fonction et protocole d'interface sont tenus pour acquis. Chacun de ces composants est de taille générique; la taille de chacune de leurs instanciations est déterminée par le nombre de bits ou de signaux (au niveau physique, cela correspondra au nombre de fils) nécessaires. La liste de ces éléments (illustrés par la figure 3.4) est exhaustive; elle est énumérée ci-dessous :

- Registre avec une sortie à connecter avec un autre élément du chemin de données (*VariableRegister*);
- Registre avec une première sortie identique avec celle du registre précédent et avec une deuxième sortie prévue pour des connexions vers le contrôleur (*FlagReg*);
- Constante de valeur générique (*ConstReg*);
- Boîtier d'interface entre le monde extérieur et le chemin de données, ou vice versa (ExtCon_IN, ExtCon_OUT et ExtCon_INOUT); cette interface peut être une simple connexion aussi bien qu'une fonction complexe telle que l'amplification de signal, le codage ou le décodage;
- Multiplexeur un parmi N (où N$\geq$2; le nombre d'entrées possibles est déterminé par AMICAL et celui-ci instancie le multiplexeur optimal : Mux_2 / Mux_3 / ... / Mux_i / ... / Mux_N);
- Porte-trois-états essentielle lorsque l'on cible une architecture à base de bus (Switch ou Commutateur)

- Boîtier d'interface entre contrôleur et chemin de données, ou vice versa (pc_po_int_cell et po_pc_int_cell respectivement; selon l'anticipation demandée pour la génération des signaux de commande du contrôleur vers le chemin de données, pc_po_int_cell sera dans certains cas mémorisant; selon le modèle de test ciblé, pc_po_int_cell et po_pc_int_cell seront des cellules particulières).

Tout autre composant est introduit et utilisé comme une unité fonctionnelle par AMICAL. Il est appelé explicitement par une opération (c'est-à-dire, soit un appel de procédure ou de fonction, soit un appel d'opération standard) ou une référence de tableau. Ce type de composant est présent dans la bibliothèque utilisateur qui autorise la notion de réutilisation.

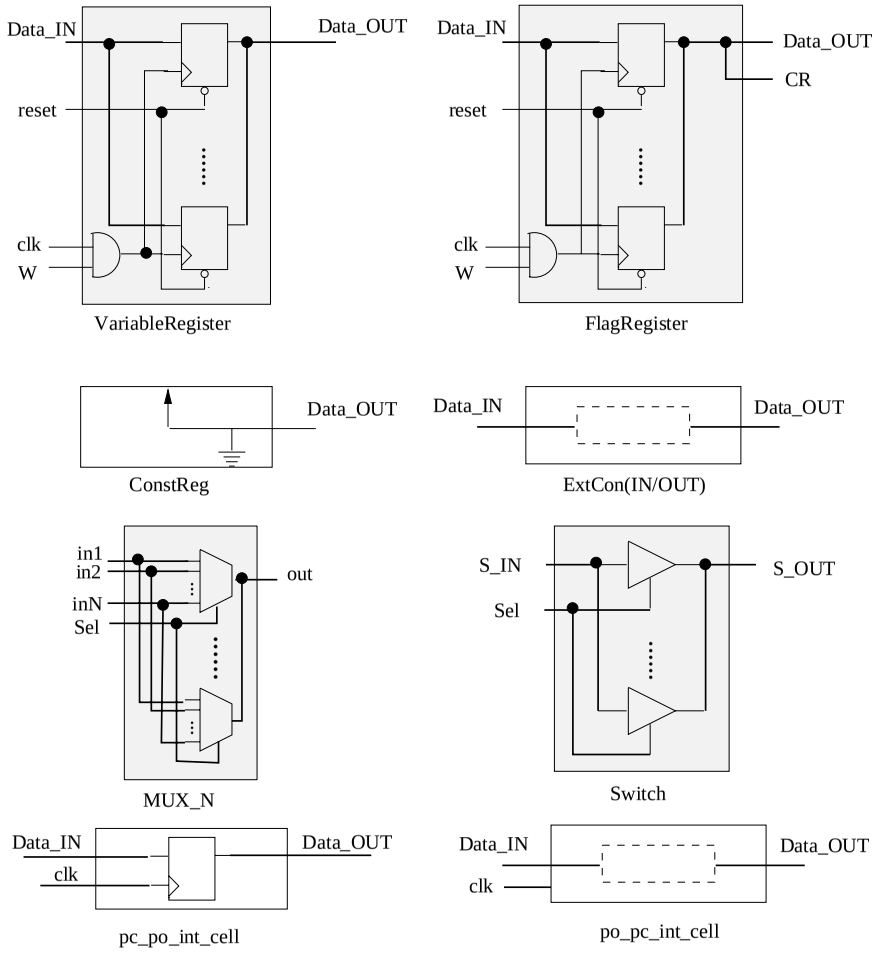


Figure 3.4: *Les composants de base.*

3.4 Modèles de descriptions intermédiaires

Le processus de synthèse est composé d'un ensemble d'étapes permettant de raffiner de manière sucessive une description comportementale en vue de produire l'architecture désirée. Le modèle de synthèse adopté par AMICAL repose sur le fait que le concepteur dispose d'une connaissance partielle de l'architecture qu'il désire synthétiser. Ces informations concernent les composants à utiliser ou l'organisation des données. Elles peuvent être prises en compte dans la description comportementale. Ainsi la description de départ peut être vue comme une représentation abstraite de l'architecture, où certains éléments d'organisation tels que le contrôleur et la partie opérative peuvent être distingués.

Ainsi, on peut reconnaître dans le processus VHDL à synthétiser des procédures et fonctions définissant les accès aux unités fonctionnelles. Elles impliquent l'instanciation d'unités fonctionnelles et leur placement au sein de la bibliothèque externe. Quant au séquencement et aux tests de condition, ils constituent le contrôleur. Dans l'exemple illustré à la figure 3.5, la procédure appel_opA positionne deux signaux de contrôle (à savoir, start_instA et sel_instA) et un signal de donnée (param1_insA) d'une instance définie dès le niveau comportemental. Cet appel de procédure implique la présence d'une unité fonctionnelle pouvant exécuter cette opération. Ceci est valable pour tout appel de procédure ou de fonction ou toute utilisation d'opérateurs standards.

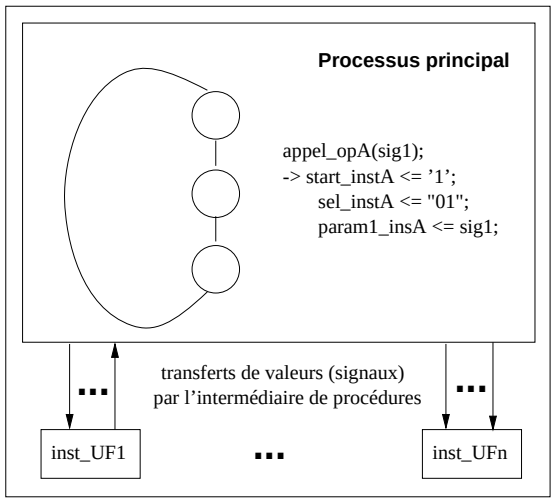


Figure 3.5: *Organisation de la description comportementale*

3.4.1 Graphe de contrôle

Quel que soit le langage de description comportementale utilisé, il est nécessaire de compiler la spécification et d'en extraire les informations réellement utiles pour la synthèse. Pour cela l'outil de synthèse AMICAL exploite la base de données de compilateurs VHDL existants. Ainsi, le flot de synthèse inclut actuellement une première étape de compilation du code VHDL comportemental soit avec CLSI de COMPASS, soit avec LVS de LEDA, soit avec LEAPFROG de CADENCE. L'étape de compilation effectue la vérification syntaxique de la description VHDL et génère un arbre syntaxique correspondant. Un traitement supplémentaire permet d'en extraire les informations pertinentes sous la forme d'un graphe de contrôle.

Le graphe de contrôle (graphe de flot de contrôle et de données) ainsi obtenu est composé :

- de nœuds pouvant correspondre soit à des opérations, soit à des instructions d'attente;
- d'arcs rattachés à une condition et décrivant le séquencement possible des opérations;
- d'entrées et de sorties pour les connexions du circuit avec le monde extérieur.

La figure 3.6 illustre les graphes de contrôle correspondant aux descriptions VHDL ((a) et (b) respectivement) présentées sur la figure 3.3. Les numéros des nœuds (entre crochets) de la figure 3.6 correspondent aux numéros de ligne de la figure 3.3.

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

3.4.2 Machine d'états finis comportementale

La machine d'états finis comportementale obtenue après l'ordonnancement du graphe de contrôle est définie comme une table de transitions où les opérations peuvent être

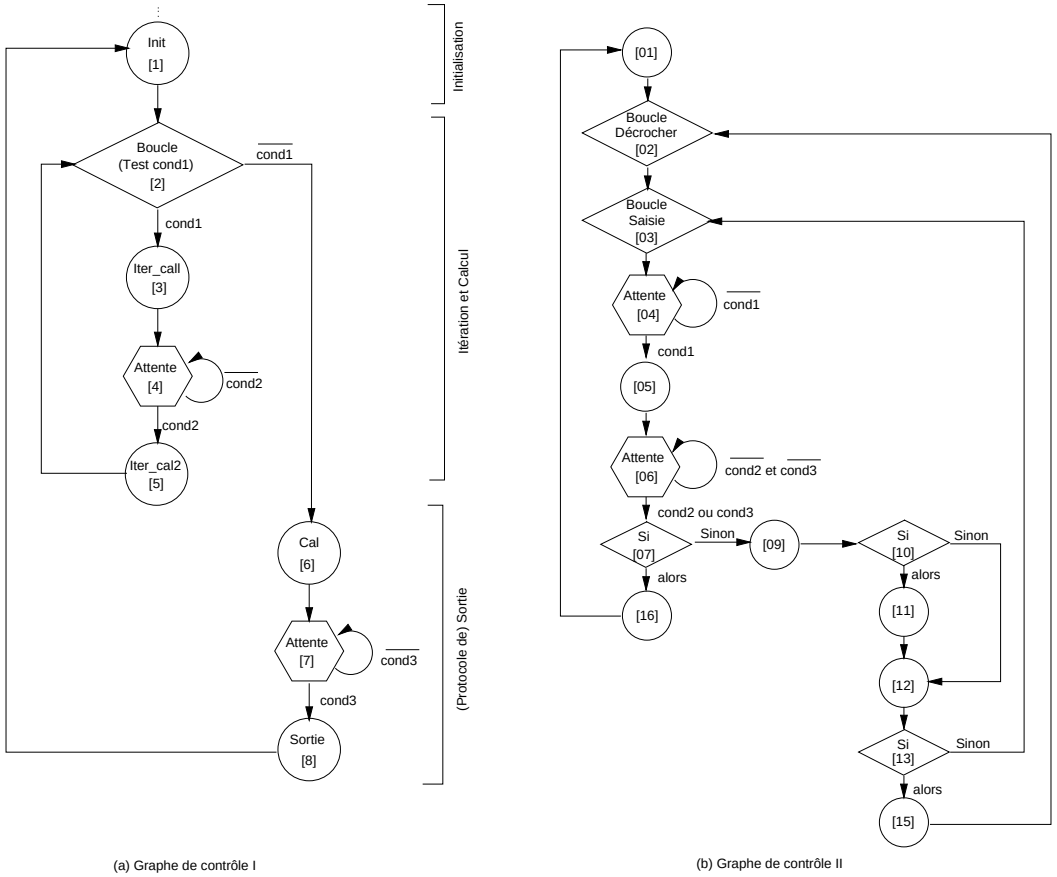


Figure 3.6: *Modèles de graphes de contrôle*

de complexité illimitée et peuvent nécessiter plusieurs cycles d'horloge. Les opérations associées à chacune des transitions de cette description peuvent s'exécuter en parallèle, mais ne requièrent pas nécessairement le même nombre de cycles d'horloge. Pour chaque transition de la machine d'états finis comportementale, la plus lente de ses opérations déterminera son délai effectif (nombre de cycles d'horloge nécessaire à son exécution réelle).

Chaque transition du contrôleur est définie par un état courant, une condition à satisfaire, et un ensemble d'opérations ou d'actions à exécuter. En d'autres termes, plusieurs transitions sont associées à un état courant, et selon la condition présente, la transition à effectuer sera déterminée. L'ordonnancement peut être considéré comme un raffinement au niveau des modules utilisés pour la construction du circuit (voir la figure 3.7); seul un extrait de l'exemple de la figure 3.3(b) a été utilisé pour la figure 3.7.

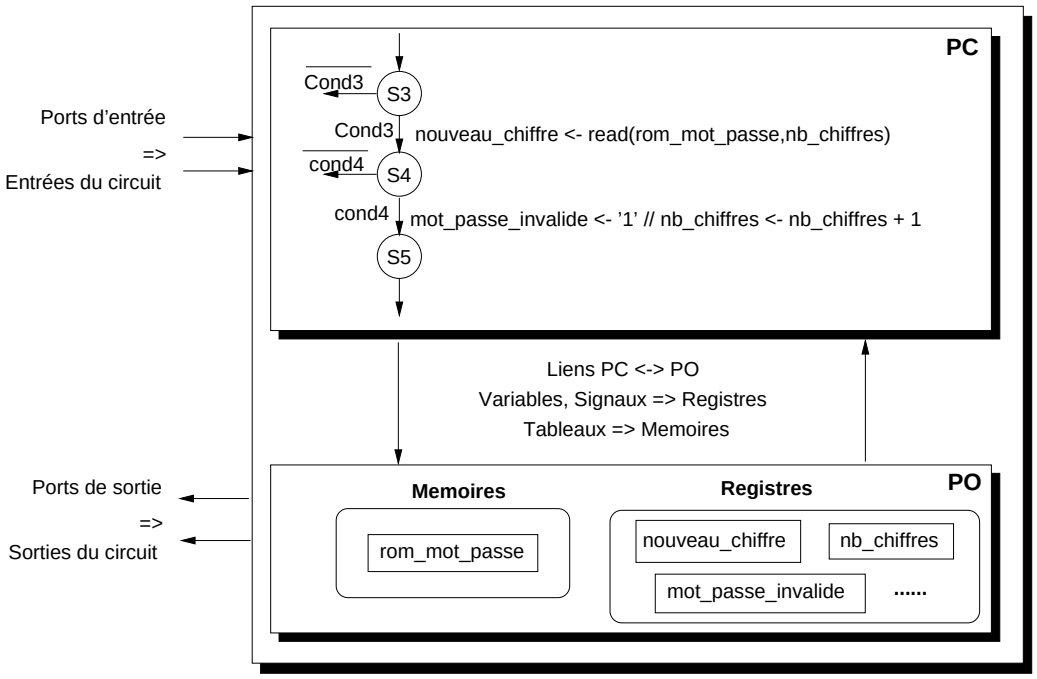


Figure 3.7: *Etat d'avancement après le macro-ordonnancement.*

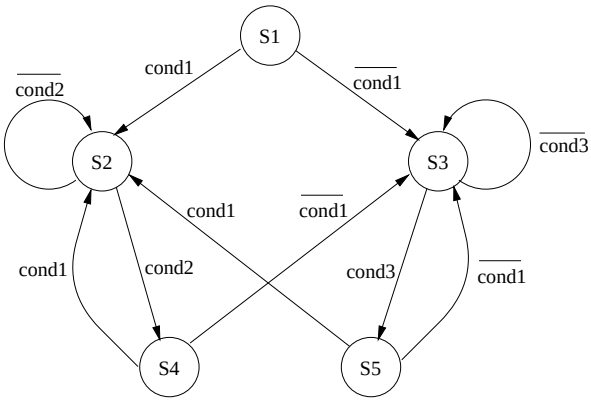
A ce stade, la structure *partie opérative-partie contrôle* devient plus explicite. De nouveaux détails de séquencement apparaissent au niveau du contrôleur. Les étapes de calcul de la description initiale sont réparties en étapes de contrôle ne comportant ni boucle ni dépendance de données. L'étape d'ordonnancement fixe aussi le nombre et le type de registres nécessaires. La partie opérative contient aussi les unités fonctionnelles inférées explicitement dans la description comportementale, telles que les mémoires ou les unités fonctionnelles exécutant des procédures se partageant des données communes. Ainsi de nouveaux liens sont créés entre la partie opérative, encore fictive, et la partie contrôle pour lier les variables et les signaux de la description comportementale aux registres et pour lier certains appels de procédure aux unités fonctionnelles correspondantes.

L'ordonnancement à ce niveau définit le rôle des parties contrôle et opérative respectivement. En effet, le traitement des états courant et suivant ainsi que le calcul de condition se font au niveau du contrôleur, et l'exécution des opérations se fait par le chemin de données. En ce qui concerne les opérations, l'ordonnancement fixe leur ordre d'exécution; celles pouvant s'exécuter en parallèle se retrouveront associées à une même transition, sans notion de séquentialité entre elles. Traitées telles quelles, des opérations parallèles

nécessiteront forcément des unités fonctionnelles distinctes pour leur exécution.

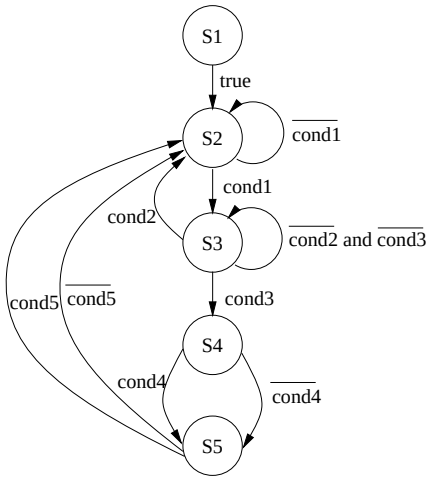
Les machines d'états finis comportementales correspondant aux descriptions des figures 3.3 et 3.6 sont décrites sur la figure 3.8, d'une part sous la forme d'une table de transitions et d'autre part sous la forme d'un automate.

Etat	Conditions	Actions	Etat Suivant
S1	cond1	Init; Itér_call	S2
S1	$\overline{cond1}$	Init; Cal	S3
S2	cond2	-	S4
S2	$\overline{cond2}$	-	S2
S3	cond3	-	S5
S3	$\overline{cond3}$	-	S3
S4	cond1	Itér_cal2; Itér_cal	S2
S4	$\overline{cond1}$	Itér_cal2; Cal	S3
S5	cond1	Sortie;Init;Cal1	S2
S5	$\overline{cond1}$	Sortie;Init;Cal	S3



(a)

Etat	Conditions	Actions	Etat Suivant
S1	true	[01]	S2
S2	cond1 touche_appuyée = 0	[05]	S3
S2	$\overline{cond1}$ touche_appuyée = 0	-	S2
S3	cond2 délai_écoulé = tmp_max or raccroché	[16];[01]	S2
S3	cond1 touche_appuyée = 0	[09]	S4
S3	$\overline{cond2}$ and délai_écoulé = tmp_max and cond3 touche_appuyée = 0 and not(raccroché)	-	S3
S4	cond4 touche_appuyée = nouveau_chiffre	[11];[12]	S5
S4	$\overline{cond4}$ touche_appuyée = nouveau_chiffre	[12]	S5
S5	cond5 nb_chiffres = 3	[15]	S2
S5	$\overline{cond5}$ nb_chiffres = 3	-	S2



(b)

Figure 3.8: *Modèles de machines d'états finis comportementales.*

Les facteurs déterminant les opérations pouvant (et celles ne pouvant pas) s'exécuter en parallèle sont [ORA93] :

- la dépendance de données apparaissant entre la production d'une valeur et sa utilisation;

- les coupures de chemins lors de la rencontre d'une instruction d'attente.

Par ailleurs, une utilisation suivie d'une modification de valeur d'une variable donnée dans la description comportementale pourra être traduite par l'exécution parallèle des deux opérations correspondantes dans un même cycle d'horloge. En effet, l'utilisation se faisant en début de transition et la production en fin de transition, il est assuré que la nouvelle valeur ne sera stockée qu'à la fin du cycle d'horloge.

3.4.3 Machine d'états finis comportementale avec ressources (reliée)

L'ordonnancement de la description initiale est suivi de l'allocation des ressources (ou unités fonctionnelles) assurant l'exécution des opérations. Cette étape produit à partir de la machine d'états finis comportementale une machine d'états finis comportementale dite "avec ressources".

Lors de l'allocation des unités fonctionnelles, chaque opération appelée au niveau comportemental est associée à une unité fonctionnelle précise. Dans le cas où deux opérations doivent s'exécuter en parallèle, elles sont associées à deux unités fonctionnelles différentes. Par contre les opérations appartenant à des étapes de contrôle différentes peuvent partager les mêmes ressources. Un exemple de lien entre opérations et ressources est donné dans la figure 3.9. Il correspond à l'exemple comportemental de la figure 3.3(b), soit à un extrait de description du répondeur téléphonique.

La machine d'états finis comportementale avec ressources correspond donc à une machine d'états finis comportementale dont chaque opération est associée à une unité fonctionnelle pour son exécution. Ces unités fonctionnelles constituent l'ensemble des unités fonctionnelles allouées. Elles correspondent à des composants de l'architecture finale.

Cette architecture intermédiaire comprend un ensemble d'états, de transitions et de composants alloués. Les transitions sont définies par des conditions et des opérations auxquelles sont rattachées des unités fonctionnelles allouées; l'exécution de chacune de ces opérations peut toujours nécessiter plusieurs cycles d'horloge.

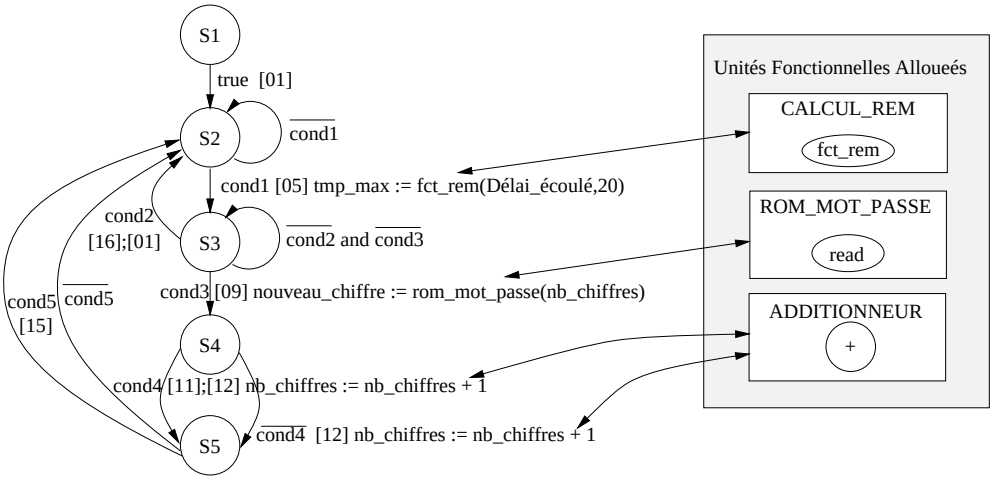


Figure 3.9: *Exemple de liens entre opérations et ressources.*

Le passage à une machine d'états finis comportementale avec ressources correspond à une étape de raffinement non pas du contrôleur mais uniquement de la partie opérative puisque aucune information complémentaire n'est introduite dans le contrôleur (figure 3.10).

A ce stade, la partie opérative est donc enrichie avec les unités fonctionnelles allouées et les liens entre le contrôleur et le chemin de données incluent désormais la correspondance entre les opérations et les opérateurs.

3.4.4 Machine d'états finis au niveau transfert de registres

Le micro-ordonnancement permet de traduire une machine d'états finis comportementale avec ressources en une machine d'états finis au niveau transfert de registres. L'allocation des unités fonctionnelles, étape précédente, ayant permis d'associer une opération à une unité fonctionnelle, ceci permet de connaître le modèle d'exécution de chaque opération au niveau transfert de registres car chaque élément de la bibliothèque comporte une description de son schéma d'exécution. Selon ce dernier, une opération peut nécessiter un ou plusieurs cycles.

Lors du micro-ordonnancement, des états intermédiaires sont introduits pour que les actions effectuées à chaque transition se bornent à un temps d'exécution correspondant physiquement à un cycle d'horloge. Dans le cas du répondeur téléphonique, si l'on suppose

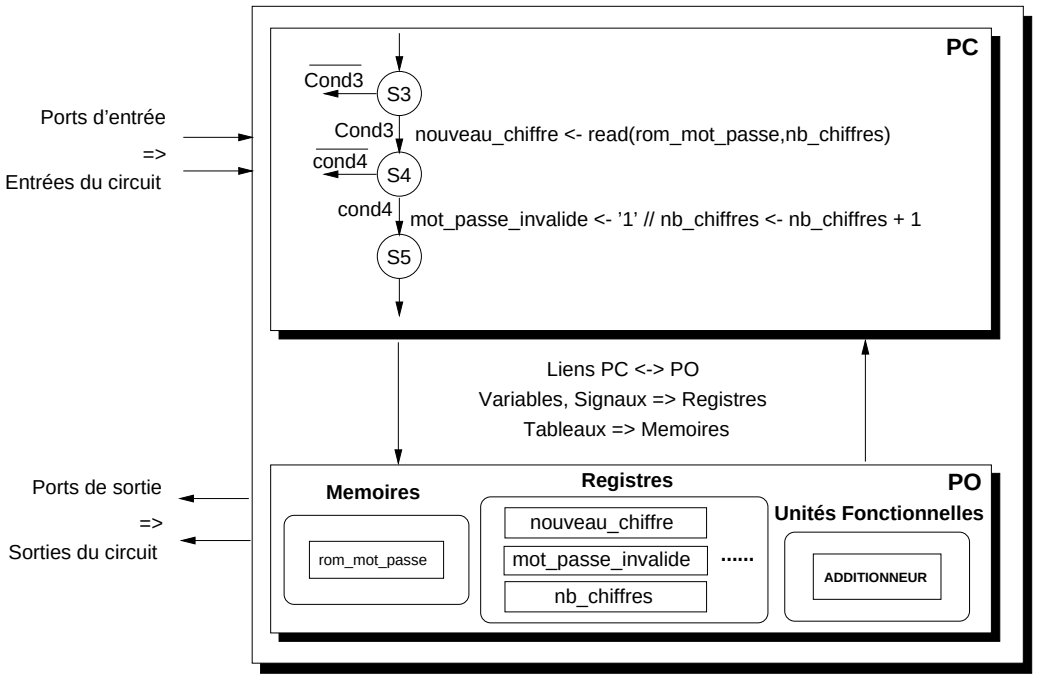


Figure 3.10: *Modèle de machine d'états finis comportementale avec ressources.*

que suite à l'allocation d'unités fonctionnelles l'addition requiert 2 cycles d'horloge, alors pour chaque transition comprenant une addition (par exemple la transition S4→S5) un nouvel état est inséré, et la transition précédente s'effectuera alors en deux micro-cycles (=micro-transitions). Ceci est illustré sur la figure 3.11. De même, si la fonction fct_rem nécessite trois cycles d'horloge, alors deux états sont créés et insérés au niveau de la transition S2→S3 qui s'exécutera dorénavant en trois micro-cycles.

Cette fois-ci, le raffinement se fait au niveau du contrôleur. Les divers transferts de données intervenant dans l'exécution de l'algorithme sont commandés par le contrôleur au micro-cycle près qui correspond à un cycle d'horloge anticipé (figure 3.12). L'étape de micro-ordonnancement engendre pour chaque micro-cycle les signaux de contrôle correspondant aux unités fonctionnelles (activation, sélection d'opération, ...) et à l'écriture des registres.

Le modèle correspondant à la machine d'états finis au niveau transfert de registres se compose :

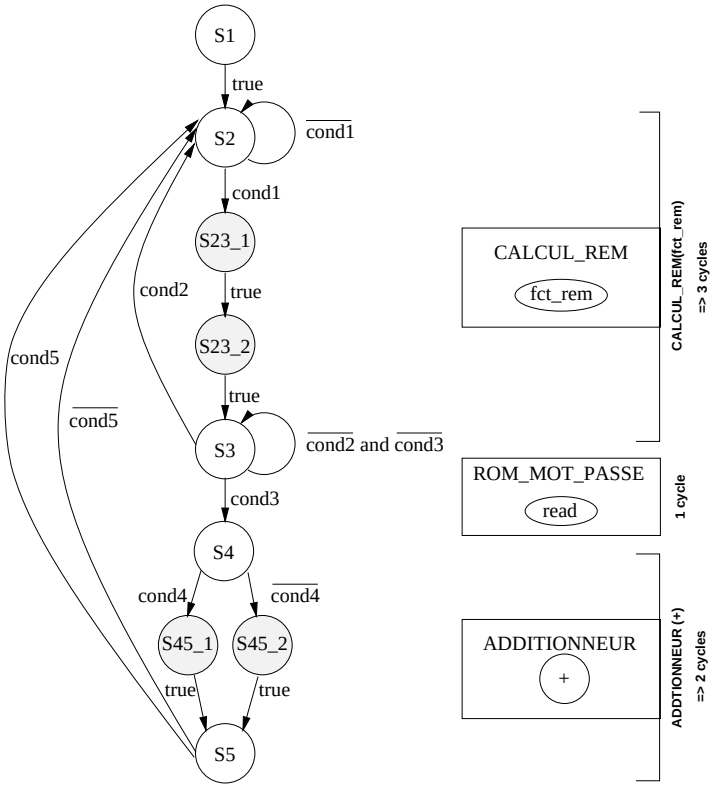


Figure 3.11: *Modèle de microordonnancement pour opérations multicycles.*

- d'un graphe de contrôle dont la base temporelle est le micro-cycle et dont les actions correspondent aux transferts entre composants fonctionnels et/ou de stockage;
- d'un ensemble de composants alloués pour constituer le chemin de données.

3.4.5 Architecture abstraite

L'architecture abstraite est générée à la suite de l'allocation des connexions et de la génération du contrôle. Le fait de connaître les transferts à effectuer pour l'exécution de l'algorithme décrit ainsi que leur ordonnancement au cycle d'horloge près permet de détecter et donc de résoudre les conflits qui peuvent se présenter pour l'interconnexion entre composants.

Le chemin de données à ce niveau contient tous les éléments nécessaires à son bon fonctionnement. S'ajoutant aux unités fonctionnelles et aux registres précédemment alloués, les éléments d'interconnexion permettent de définir les différents chemins possibles

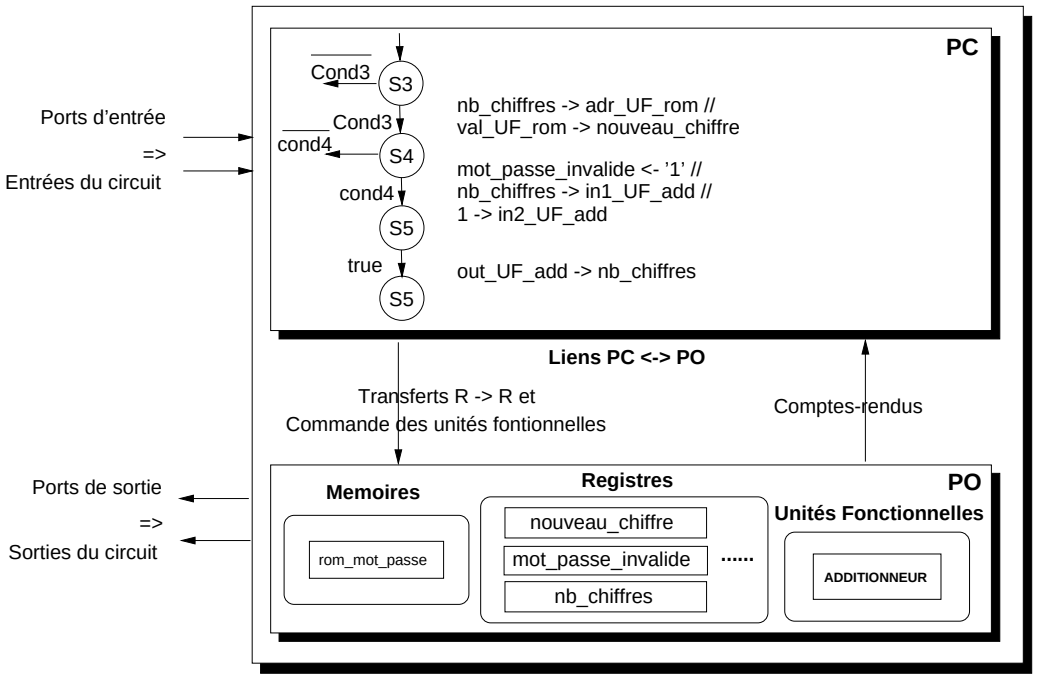


Figure 3.12: *Modèle de machine d'états finis au niveau transferts de registres.*

sans conflits (un conflit existe dans le cas où deux transferts ou plus s'exécutent par l'intermédiaire du même chemin d'interconnexion et au cours du même cycle).

Le contrôleur contient des informations supplémentaires par rapport à la machine d'états finis au niveau transfert de registres; il commande les transferts de données par la gestion des signaux de contrôle des multiplexeurs ou des commutateurs; autrement dits, les vecteurs de commandes associés à chaque micro-cycle sont définis précisément).

A ce niveau de la synthèse, la seule information manquante concerne la synchronisation explicite des parties contrôle et opérative. Ceci explique le fait que l'architecture est dite abstraite. De cette description d'architecture abstraite, une session de personnalisation permet d'obtenir une description au niveau transfert de registres. La figure 3.13 présente l'architecture abstraite et les liens existants entre la partie contrôle et la partie opérative, obtenus après l'allocation des connexions.

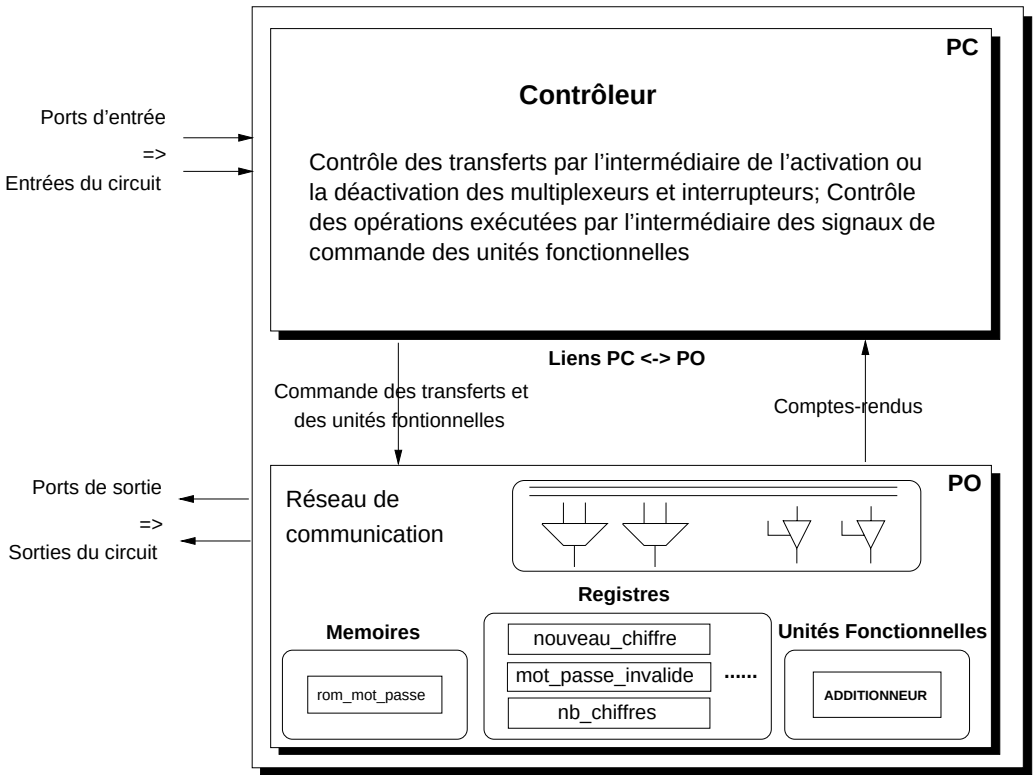
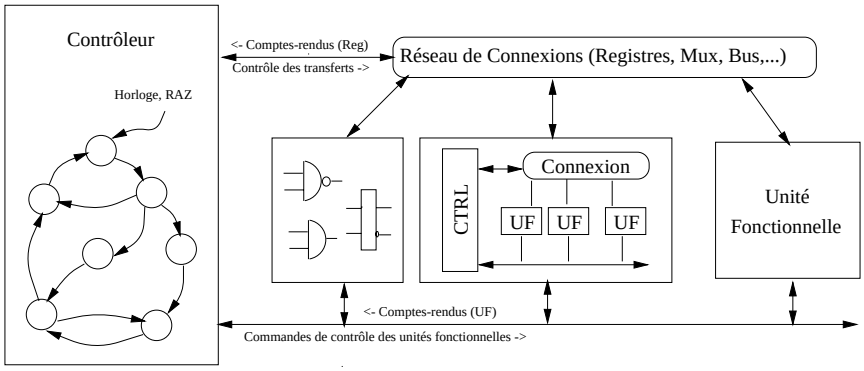


Figure 3.13: *Architecture abstraite*

3.5 Modèle général de l'Architecture Cible

Le résultat de la synthèse est un circuit synchrone comportant un contrôleur et une partie opérative, comme illustré par la figure 3.14.



Outre ces 2 blocs, le circuit généré peut, dans le cas d'une anticipation de la génération des signaux de contrôle, contenir un bloc supplémentaire d'interface pour la resynchronisation des commandes.

Les différentes architectures produites par AMICAL (chapitre 4)correspondent à la synthèse d'un seul processus ("process" VHDL). Des connexions supplémentaires correspondant au reste de la description comportementale peuvent être requises (blocs concurrents au "process" principal faisant l'objet de la synthèse); c'est le cas notamment du bloc de synchronisation (fig. 3.17). Ces connexions seront introduites lors de la phase de personnalisation de l'architecture abstraite [Moh94].

3.5.1 Contrôleur

La partie contrôle produite en sortie d'AMICAL correspond à une machine de Mealy. Les actions, définies pour chaque transition au cycle d'horloge près, correspondent à des commandes de transferts de données entre composants du chemin de données. Certains signaux sont utilisés pour la commande des unités fonctionnelles.

Ce contrôleur est obtenu à la suite des étapes de raffinement du graphe de contrôle extrait de la description comportementale, étapes décrites en détail au cours des sections

précédentes. Celui-ci commande, comme le niveau d'abstraction l'indique, les transferts entre unités fonctionnelles et registres. De tels transferts sont réalisés par l'activation d'éléments de communication ou de connexion : bus, multiplexeurs, commutateurs, ... Cette partie du circuit commande aussi les opérations à exécuter en générant les signaux de contrôle nécessaires pour chacune des unités fonctionnelles.

La description VHDL du contrôleur généré se présente sous le modèle habituel de 2 processus concurrents. Le premier d'entre eux produit les différents signaux de contrôle à partir des conditions présentes, ainsi que l'état suivant. Toute nouvelle situation (observée par un changement de condition ou de l'état courant) entraîne une nouvelle itération pour le calcul des signaux de commande et de l'état suivant. Le deuxième processus réalise l'initialisation de manière asynchrone lors d'une remise à zéro (reset) et permet le passage d'un état à l'état suivant évalué à chaque cycle d'horloge.

3.5.2 Chemin de données

La partie opérative, très souvent appelée chemin de données, correspond à un ensemble de blocs interconnectés de manière à permettre l'exécution d'un certain nombre d'opérations. Ces blocs peuvent être de tailles différentes.

Remarque :

> L'appellation "chemin de données" , par opposition à "partie opérative" correspond plus précisément à une structure ayant la particularité de manipuler des données et éléments (unités fonctionnelles, de stockage ou de communication) en sous-couches régulières[GDWL92]; cependant, ces deux appellations seront confondues au cours de cette thèse.

Parmi les composants du chemin de données, on peut distinguer trois types majeurs d'éléments : des unités fonctionnelles pouvant chacune exécuter une ou plusieurs opérations, des unités de stockage qui sont principalement des registres, et les unités de connexion.

L'agencement des divers modules est défini avant tout par l'architecture cible. Ainsi dans le cas d'AMICAL chaque entrée ou sortie d'unité fonctionnelle est connectée à un registre ou à un port (d'entrée ou de sortie); aucun transfert direct entre unités fonctionnelles n'est visible par le contrôleur. Par contre, la partie opérative peut contenir des unités fonctionnelles communicant directement[PFea95]. Ceci correspond à des éléments de la description VHDL dehors du processus principal. Les connexions entre les modules de la partie opérative sont propres à l'architecture visée; celles-ci sont définies par des éléments tels que des multiplexeurs, des bus, ... et seront allouées selon les transferts de données découlant de la description comportementale et selon le type d'architecture envisagé (cf. chapitre 4).

Les opérations effectuées par un chemin de données peuvent être définies globalement comme des opérations de registre à registre. L'expression registre s'étend dans ce cas aux entrées et sorties car elles sont très souvent stockées et synchronisées. En d'autres termes, toute opération inclut des lectures de registres, pour la prise en compte des valeurs d'entrée, et des écritures de registres, pour la récupération des résultats. Chacune de ces opérations peut nécessiter 1 cycle d'horloge ou plus. La figure 3.15 illustre un chemin de données produit par AMICAL dans le cas d'une architecture à base de multiplexeurs.

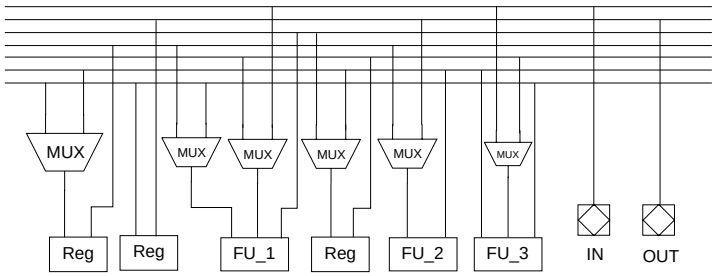


Figure 3.15: *Chemin de données à base de multiplexeurs.*

Les registres sont directement dépendants du signal d'horloge pour toute écriture. Aussi une opération registre à registre est-elle synchronisée avec le signal d'horloge. La valeur de chacun des registres ne peut être modifiée qu'au moment d'un front d'horloge (c'est-à-dire, une seule fois par cycle).

Même si ce modèle est synchrone, il n'impose pas de contrainte sur le temps d'exécution de l'opération proprement dite. Cependant AMICAL ne permet pas le séquencement

de deux opérations par "chaînage" pendant un même cycle d'horloge. Par conséquent le temps d'exécution de chacune des unités fonctionnelles de la bibliothèque disponible est exprimé par un nombre entier de cycles d'horloge ($\geq$ (délai d'exécution / période d'horloge)).

Comme spécifié précédemment, l'architecture synthétisée ne reflète que la mise en œuvre d'un unique processus de la description comportementale VHDL. Aussi des connexions supplémentaires peuvent être requises. Celles-ci peuvent être de style *unité fonctionnelle-unité fonctionnelle*, ou des transferts de valeurs plus généraux tels qu'entre les unités fonctionnelles et le monde extérieur au circuit. Ces connexions peuvent être rajoutées de manière semi-automatique durant l'étape de génération des description VHDL au niveau transfert de registres [Moh94].

Les unités fonctionnelles peuvent être des opérateurs standards tels un additionneur, un multiplieur, une UAL, ... mais aussi des blocs plus complexes tels qu'une mémoire-cache, une unité d'entrée-sortie, ou tout un circuit préalablement synthétisé. La description d'entrée au niveau comportemental peut inclure une (ou plusieurs) instanciation(s) explicite(s) de bloc(s) complexe(s) en complément du processus à synthétiser. Ce processus principal peut manipuler des signaux connectés aux ports des instances à travers des procédures. Ceci implique par conséquent qu'une partie de l'unité fonctionnelle (les ports interfacés par l'intermédiaire de procédures) est visible de l'architecture engendrée et que ses connexions se font automatiquement.

3.5.3 Synchronisation entre contrôleur et chemin de données

Comme énoncé précédemment, la partie contrôle d'un circuit engendre des signaux de commande qui seront ensuite consommés par la partie opérative. Cependant s'il est évident que le contrôleur générera ces signaux avant toute consommation, le champ est ouvert à toute anticipation de la génération des signaux de commande (par rapport à l'exécution des commandes). Il en découle qu'il existe plusieurs modèles de synchronisation entre contrôleur et chemin de données.

Dans le cas où les unités fonctionnelles sont purement combinatoires, les signaux de

commande peuvent être produits en début de cycle pour être consommés par la suite pendant le même cycle d'horloge, comme illustré par la figure 3.16(a). Dans le cas où un recouvrement (*pipeline*) est permis, le contrôleur et le chemin de données peuvent aussi travailler en parallèle sur des transitions différentes. Ceci implique que tout signal généré par le contrôleur ne sera pris en compte par la partie opérative que le(s) cycle(s) suivant(s), ce qui implique une anticipation du contrôleur par rapport au chemin de données. La figure 3.16 (b) illustre ce dernier cas avec une anticipation de degré 1. L'anticipation d'un cycle permet d'avoir une situation de "*pipeline*" entre la partie contrôle et la partie opérative. Dans certains cas des cycles vides seront introduits automatiquement pour l'attente de comptes-rendus [AKRJ94]. Ce deuxième modèle est restreint à certaines applications : ainsi si des données continues sont injectées dans le circuit, chacune valide pendant un cycle d'horloge et accompagnée d'un signal de contrôle externe.

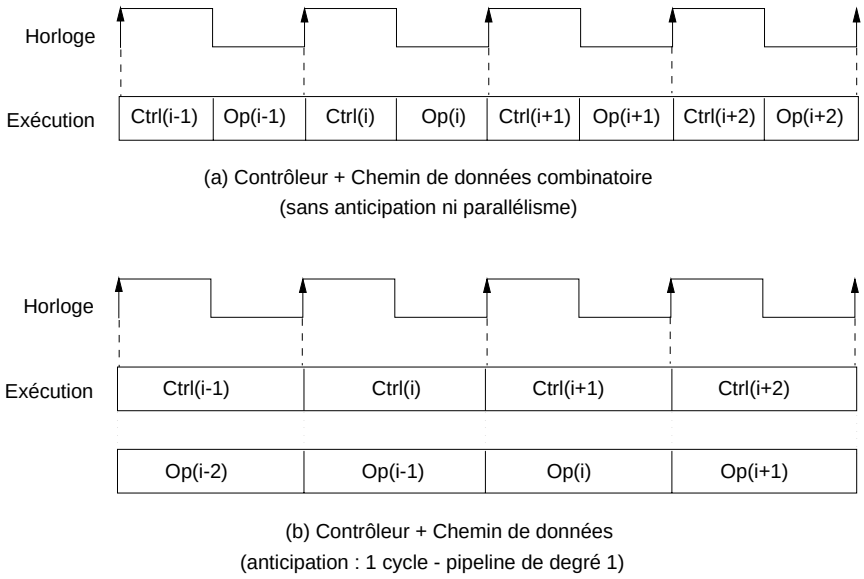


Figure 3.16: *Exemples de modèles de synchronisation*

Ces deux modèles de synchronisation se traduisent par des architectures légèrement différentes l'une de l'autre. Dans le cas d'une anticipation du contrôleur accompagnée d'un parallélisme dans l'exécution des parties contrôle et opérative (figure 3.16 (b)), une mémorisation des signaux de commande est effectuée. Le pipeline créé permet à la partie contrôle et à la partie opérative de travailler sur des transitions différentes. Aussi pendant que le chemin de données traitera les opérations de la transition i, le contrôleur produira les signaux liés à la transition (i+1), d'où la nécessité de mémoriser les signaux précédents

avant qu'ils ne soient modifiés. L'ensemble des cellules de mémorisation peut faire l'objet d'un bloc à part entière. Plus de détails à propos de ces modèles peuvent être retrouvés dans [Moh94]. La figure 3.17 illustre l'architecture obtenue dans ce cas. Les signaux de contrôle (commandant les transferts, les stockages des registres, unités fonctionnelles, ...) sont mémorisés le temps d'un cycle par le bloc d'interfaçage nommé PC-PO. Par symétrie et pour faciliter l'insertion de cellules de test, un deuxième bloc d'interfaçage : PO-PC a été introduit. Pour une architecture sans propriété de test, ce bloc représente de simples fils de connexion.

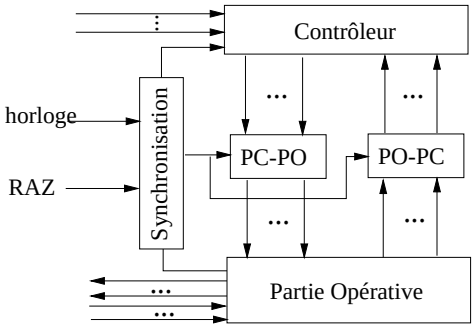


Figure 3.17: *Architecture générale avec anticipation.*

3.6 Conclusion

Ce chapitre décrit les modèles architecturaux manipulés par AMICAL afin d'obtenir, à partir d'une description comportementale initiale, une architecture au niveau transfert de registres composée principalement d'un contrôleur et d'une partie opérative. A l'exception des descriptions d'entrée et de sortie, ces modèles concernent des formats intermédiaires transparents à l'utilisateur effectuant une synthèse automatique. AMICAL procède par raffinements successifs au cours de la synthèse : à chaque étape, de nouveaux détails architecturaux sont introduits et un nouveau modèle intermédiaire est construit. Comme on le verra au chapitre 5, cette construction graduelle du circuit jusqu'au niveau transfert de registres fait d'AMICAL un véritable assistant à la synthèse. Ceci provient de la grande lisibilité des états intermédiaires et de l'opportunité offerte à l'utilisateur de suivre les étapes de raffinements et éventuellement d'intervenir au cours de celles-ci.

CHAPITRE 4

Génération d'architectures

Chapitre 4

Génération d'architectures

Ce chapitre a pour objet de présenter l'étape de transposition de la description algorithmique initiale, donnée en entrée du système AMICAL, à trois styles d'architecture : les deux premières basées respectivement sur l'utilisation de bus et sur l'utilisation de multiplexeurs, la partie contrôle étant formée d'une machine d'états finis de type Mealy, et la dernière possédant une partie contrôle micro-programmée (vecteurs de commandes regroupés dans une ROM). Cette étape de génération d'architectures, étape clé au sein du flot de synthèse, s'appuie sur un format de description interne au système, SOLAR, langage flexible pouvant s'adapter à chaque style visé. Cette étape donne lieu à la création d'une architecture décrite en SOLAR et dite abstraite car générique, architecture formée de deux blocs principaux : le bloc de contrôle et le bloc fonctionnel ou partie opérative. Chaque style architectural possédant ses propres caractéristiques et contraintes, chacun d'eux nécessite pour pouvoir être généré, l'application d'algorithmes spécifiques dont les points principaux seront décrits. Cette transposition multi-cible autorisée au sein d'AMICAL est une caractéristique clé donnant lieu à des possibilités d'exploration architecturale plus larges.

4.1 Introduction

La possibilité de réaliser une exploration architecturale est un des avantages de la synthèse comportementale, qui se place à un haut niveau d'abstraction. Pour exploiter au mieux cette caractéristique, l'idéal serait de posséder un outil de synthèse permettant de transposer une description comportementale à un grand nombre de styles architecturaux. La question du choix définitif de l'architecture serait cependant la principale difficulté à laquelle il faudrait alors faire face.

La solution de se restreindre à un style d'architecture bien précis et défini est une solution largement exploitée à travers les laboratoires de recherches étudiant la synthèse architecturale. Cette solution présente le principal avantage de simplifier l'étape de transposition et de réduire les problèmes notamment algorithmiques à un flot relativement mince. Un autre avantage de cette focalisation est d'obtenir une architecture bien optimisée car bien ciblée.

Une alternative à ces deux solutions consiste à se restreindre à un petit nombre d'architectures cibles bien définies, et non plus à une seule. C'est la solution adoptée pour le système AMICAL.

Une condition sine qua non est de posséder une représentation interne utilisée au cours de la synthèse suffisamment flexible pour pouvoir être adaptée à chaque style architectural visé. Au sein d'AMICAL, le langage intermédiaire SOLAR présente ces caractéristiques. Il autorise la description d'un circuit sous la forme d'un assemblage de composants génériques, aiséments transposables aux composants physiques de la bibliothèque.

Ce chapitre s'intéresse à la partie du flot de synthèse donnant lieu à la génération d'une architecture abstraite, issue de l'étape d'allocation des connexions. Cette architecture, décrite dans le format SOLAR, se base sur une notion de temps implicite, le micro-cycle ou cycle de contrôle de base, le temps explicite étant entièrement absent. Ce dernier sera explicité lors de la dernière étape du flot de synthèse : personnalisation et transposition SOLAR-VHDL RTL (cf. chapitre 3).

Cette étape de génération est une étape clé : c'est à ce stade que diverses décisions influeront sur les performances du circuit synthétisé; c'est à ce stade qu'il sera possible de décider du style d'architecture de la partie opérative (Multiplexeurs, Bus), et donc de l'implémentation finale du contrôleur (machine d'états finis, micro-contrôleur), le modèle de transfert de chaque style étant différent.

Ce chapitre va donc s'attacher à la description des problèmes et des solutions liés à la génération de cette architecture dite abstraite par son aspect générique et par l'absence de synchronisation explicite entre les divers blocs la constituant.

Une première section permettra de ce familiariser avec les concepts abstraits manipulés associés au format intermédiaire SOLAR et donnera un aperçu des contraintes générales liées à la génération du contrôleur et de la partie opérative.

La section suivante se focalisera de façon plus précise sur les étapes de génération des trois types d'architecture ciblées par le système AMICAL : l'une exclusivement basée sur l'utilisation de bus, dont une brève description des caractéristiques sera fournie, le sujet ayant déjà fait l'objet d'une description détaillée [Par92], la suivante basée sur l'utilisation exclusive de multiplexeurs, et la dernière fondée sur l'emploi d'un micro-contrôleur.

Les deux dernières sections permettront respectivement de discuter des performances de chaque style architectural généré par AMICAL et de conclure.

4.2 Notion d'Architecture Abstraite

4.2.1 Généralités

Comme mentionné dans le chapitre 3, l'architecture de base générée par AMICAL se compose, à l'instar de nombreux outils de synthèse de haut niveau, de deux entités [GDWL92, HT83, VRB+93, MLD92, Inc94]: un contrôleur ou partie contrôle, et un chemin de données ou partie opérative. La figure 4.1 illustre cette représentation.

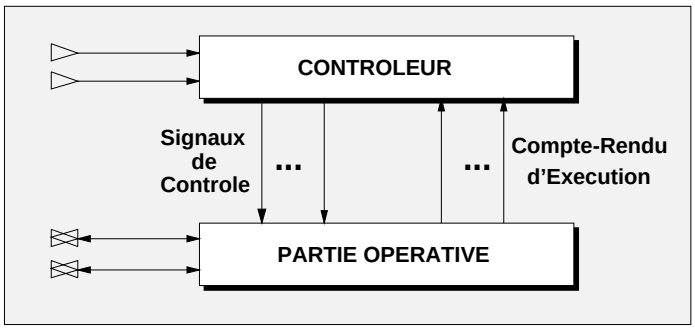


Figure 4.1: Représentation d'architecture

- Le contrôleur est une machine d'états finis (MEF) dont les ports d'entrée se composent des signaux de contrôle externes et des comptes rendus d'éxecution (flags) issus de la partie opérative, et dont les ports de sortie sont les signaux contrôlant l'exécution des tâches au sein du chemin de données.

- La partie opérative se charge d'exécuter les opérations demandées par le contrôleur et spécifiées dans la description comportementale, cela à travers un certain nombre de transferts élémentaires se déroulant en son sein entre les diverses unités la constituant. Ces unités peuvent être très simples (multiplexeurs, registres, ...) ou plus complexes (unités fonctionnelles complexes issues, par exemple, d'une synthèse précédente).

Pour la synthèse de haut niveau, une architecture cible simple et précise peut fortement réduire la complexité des problèmes qui se posent lors de la synthèse. D'un autre côté, une architecture moins contrainte (ciblée), bien que plus difficile à synthétiser, peut

contribuer à accroire la qualité des résultats obtenus[GDWL92, VRB+93, MLD92]. C'est ce que nous nous proposons de réaliser en prenant pour base un modèle générique, le plus général possible, bien que cette généralité reste relative et en rapport avec les architectures communément utilisées.

La génération d'une architecture abstraite avant l'étape de personnalisation et la génération de la description RTL finale va permettre l'application de solutions architecturales différentes réalisés à l'aide d'algorithmes spécifiques à chaque solution.

Cependant, avant d'aborder les difficultés spécifiques à chaque architecture cible, une brève présentation de ce que signifie la notion d'architecture abstraite dans l'environnement AMICAL ainsi qu'une explication générale concernant la génération des parties contrôle et chemin de données indépendamment de l'architecture visée, permettra de mieux saisir les caractéristiques liés à cette étape clé du processus de synthèse.

4.2.2 Architecture globale abstraite

Le format intermédiaire adopté pour la manipulation et la génération de l'architecture abstraite au sein d'AMICAL est le format SOLAR. Un exemple général de représentation d'architecture de la figure 4.1 à l'aide de ce format est illustré par la figure 4.2.

Ce fichier SOLAR comprend trois types d'informations: les interfaces externes, les composants utilisés et les connexions entre blocs. Le mot clé *Interface* spécifie les interfaces externes, correspondant aux signaux externes. Ces interfaces sont les mêmes que celles déclarées dans la description comportementale. Le mot clé *Contents* décrit l'organisation interne du circuit (elle est composée de deux blocs). Le contrôleur correspond au bloc *ControlPart*. Ses connecteurs (*PortInstance*) sont les signaux de contrôle envoyés à la partie opérative, les comptes-rendus venant de cette dernière et les signaux de contrôle externes. La partie opérative est décrite dans le bloc *DataPath*.

Les connexions entre la partie opérative et la partie contrôle sont spécifiées par une suite de définitions *Net*. Chaque connexion, identifiable par un nom unique, contient une référence à deux signaux dont chacun appartient soit à l'un des deux blocs, soit à

```
1  (Solar AMICAL_for_Circuit
2     (DesignUnit nom_du_circuit_circuit
3        (View structure (ViewType "InterconnectedSystems")
4           (Interface
5              {(Port (Array nom_du_connecteur longeur_de_bits)
6                 (Direction IN | OUT | INOUT)
7                 (Property PortType DATA | CONTROL)
8              )}
9              {(Port nom_du_connecteur
10                (Direction IN | OUT | INOUT)
11                (Bit)
12                (Property PortType DATA | CONTROL)
13             )}
14          )
15          (Contents
16             (Instance ControlPart
17                (ViewRef Behavior RT-Level ControlPart)
18                {(PortInstance nom_du_connecteur
19                   (Property PortType DATA | CONTROL)
20                )}
21             )
22             (Instance DataPath
23                (ViewRef Behavior RT-Level DataPath)
24                {(PortInstance nom_du_connecteur
25                   (Direction IN | OUT | INOUT)
26                   (Property PortType DATA | CONTROL)
27                )}
28             )
29             {(Net nom_d'interconnexion
30                (Joined
31                   (PortRef nom_du_connecteur
32                      (InstanceRef nom_du_composant))
33                   (PortRef nom_du_connecteur
34                      (InstanceRef nom_du_composant)))
35             )}
36          )
37       )
38    )
39 )
```

Figure 4.2: *Format SOLAR de la configuration globale du circuit générée par AMICAL.*

l'interface globale.

4.2.3 Contrôleur généré

4.2.3.1 *Représentation*

Le contrôleur généré consiste en un ensemble d'états, un ensemble de transitions, et un ensemble d'actions associées aux transitions (machine de Mealy).

De manière plus formelle, un contrôleur de type machine de Mealy peut se définir sous forme d'un 5-uple :

$$CT = (S, I, O, FS, FO) \tag{4.1}$$

où :

- $S = \{s_i\}$ est l'ensemble des états (states), s_1 est l'état initial;
- $I = \{i_j\}$ est l'ensemble des valeurs d'entrées (inputs);
- $O = \{o_i\}$ est l'ensemble des valeurs de sorties (outputs);
- $FS = \{fs \mid fs : S \times I \rightarrow S\}$ est l'ensemble des fonctions définissant un état suivant, produit de S et de I dans S, autrement dit, donnant l'état suivant en fonction des entrées et de l'état présent;
- $FO = \{fo \mid fo : S \times I \rightarrow O\}$ est l'ensemble des fonctions définissant une sortie, produit de S et I dans O, c'est à dire que la donnée d'une sortie dépend non seulement de l'état présent (machine de Moore) mais aussi des entrées.

Le contrôleur correspondant à l'architecture abstraite est lui-même abstrait en ce sens qu'il est indépendant du temps explicite. On ne considère à ce niveau que le cycle élémentaire ou de base correspondant à une transition élémentaire du graphe des transitions du contrôleur et, implicitement, à un cycle de la future horloge (synchronisation implicite).

La génération de la partie contrôle consiste à décrire l'interface et le comportement du contrôleur en format SOLAR (figure 4.3). La description de la machine d'états finis (MEF) consiste en une table d'états. Chaque état spécifie un certain nombre d'affectations de signaux de contrôles, affectations validées par un ensemble de conditions sur les entrées provenant de la partie opérative ou encore de l'extérieur (signaux de contrôle globaux).

Selon l'architecture choisie, on aura recours à des algorithmes différents pour cette génération (voir annexe C).

```
1  (Solar AMICAL_for_Controller
2     (DesignUnit nom_du_circuit
3        (View Behavior (ViewType RT-Level)
4           (Interface
5              {(Port (Array nom_du_connecteur longueur_de_bits)
6                 (Direction IN | OUT | INOUT)
7                 (Property PortType DATA | CONTROL)
8              )}
9              {(Port nom_du_connecteur
10                (Direction IN | OUT | INOUT)
11                (Bit)
12                (Property PortType DATA | CONTROL)
13             )}
14          )
15          (Contents
16             (StateTable Controller
17                {(State nom_du_state
18                   (Case
19                      {(Alt expression_du_condition
20                         (Micro-Cycle ID_du_micro_cycle)
21                         {(Path {nom_du_composant_dans_le_chemin)}
22                         {(Assign nom_du_connecteur valeur)}
23                         (NextState nom_du_state)
24                      )}
25                   )
26                )})
27             (StateList {nom_des_state})
28          )
29       )
30    )
31 )
```

Figure 4.3: *Format SOLAR du contrôleur généré par AMICAL.*

4.2.3.2 *Génération du contrôleur : aspects généraux*

Dans la spécification d'un contrôleur, la description d'un état (mot clé *State*) consiste à expliciter les informations sur le séquencement (l'évaluation de conditions, les affectations des signaux de contrôle des transferts et le passage à l'état suivant). Selon la condition, les chemins de connexion sont validés par les signaux de contrôle. Dans un micro-cycle, à chaque transfert de l'ensemble des transferts en parallèles correspondant au micro-cycle, on associe un chemin de connexion.

D'un point de vue transfert de registres, un micro-cycle est composé d'un ensemble de transferts élémentaires devant s'exécuter en parallèle. Par contre, d'un point de vue structurel (après la décomposition en une partie opérative et un contrôleur), il est nécessaire que chaque micro-cycle soit transformé en un vecteur de commande. Cette décomposition structurelle nécessite de générer deux types d'informations:

- les commandes de sélection des composants de la partie opérative,

- les chemins de connexion qui expriment le comportement du micro-cycle.

Ceci est nécessaire afin de générer le vecteur de commande complet (ce dernier commandant l'activité des unités fonctionnelles et les transferts à travers les chemins de connexions).

Deux parties sont à considérer pour générer le contrôleur :

- l'interface du contrôleur : mot clé *Interface*,

- le diagramme d'états : mot clé *StateTable*.

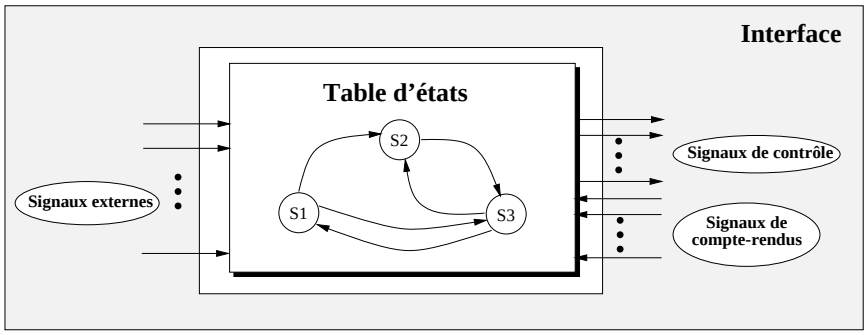


Figure 4.4: *Parties du contrôleur à générer.*

Les figure 4.3 et 4.4 donnent respectivement la description SOLAR et le schéma correspondant du contrôleur basé sur cette description, donc tel qu'appréhendé par AMICAL.

L'interface du contrôleur est composée de trois types d'objets: les ports de contrôle pré-définis du circuit (signaux externes) , les signaux de contrôle destinés à chaque composant de la partie opérative et les comptes-rendus de registres provenant de cette dernière (figure 4.5).

Le diagramme d'états contient toute l'information liée à chaque état. Les objets à traiter sont les macro-cycles, les micro-cycles qui les composents, les conditions et les

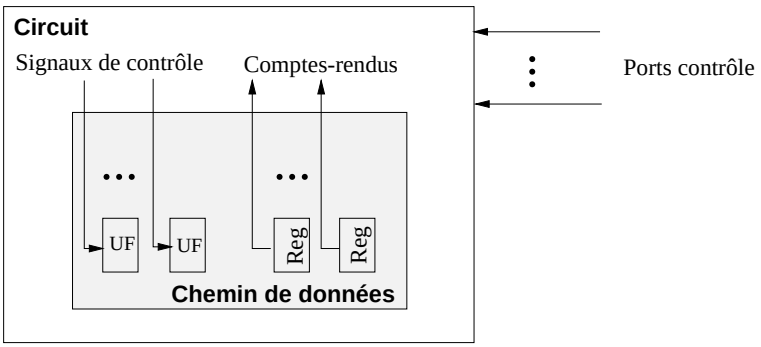


Figure 4.5: *Les objets à traiter de l'interface.*

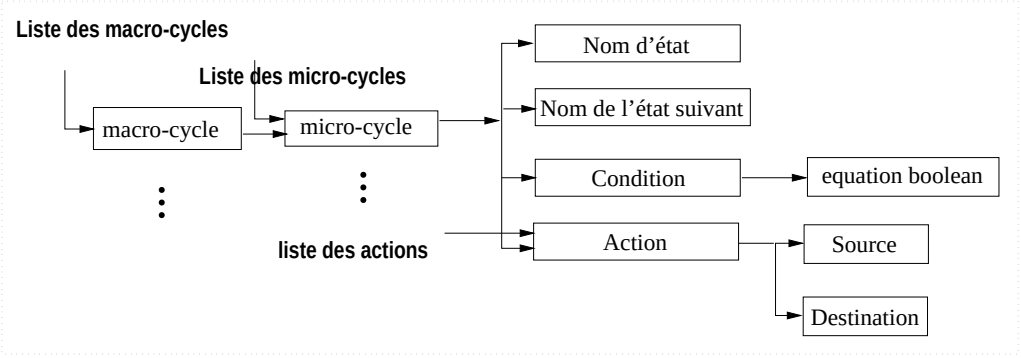


Figure 4.6: *Les objets à traiter du diagramme d'états.*

signaux de contrôle qui doivent être actifs. La figure 4.6 montre ces objets et les liaisons entre eux.

La première phase de génération du contrôleur consiste à construire son interface définie par :

- Ses ports d'entrées : ce sont les ports externes et les ports de comptes-rendus.
- Ses ports de sorties : ce sont les ports de contrôle destinés à activer en temps voulu les blocs formant la partie opérative.

L'étape suivante consiste à définir précisément pour chaque micro-cycle les vecteurs de commandes à appliquer pour activer l'ensemble des composants mis en jeu durant ce cycle de contrôle : il est donc nécessaire de déterminer l'ensemble des unités de connexion à activer pour permettre la communication entre les unités actives du micro-cycle, et ensuite

de générer les signaux de sélection correspondants à la fois aux unités de connexion et aux unités devant être activées (registres, unités fonctionnelles, unités d'entrée/sortie).

Ces deux étapes de génération du contrôleur dépendent bien sur étroitement du type d'architecture ciblé et de son mode de transfert. Deux cas seront étudiés par la suite : architecture à base de bus, et architecture à base de multiplexeurs.

4.2.4 Partie opérative générée

4.2.4.1 *Représentation*

La partie opérative abstraite générée est un assemblage de ressources allouées par AMICAL. Il s'agit d'une spécification structurelle. Un modèle SOLAR de celle-ci est représenté sur la figure 4.7.

Pour chaque composant (mot clé *Instance*), l'identificateur et les connecteurs sont précisément définis, ainsi que son fichier décrivant ses caractéristiques.

Pour chaque connexion (mot clé *Net*), les deux connecteurs reliés sont décrits..

Son interface avec l'extérieur (contrôleur et extérieur du circuit) est caractérisée par les connecteurs du chemin de données permettant d'échanger les données avec l'extérieur, les comptes-rendus (ports de registres utilisés par la partie contrôle) et les signaux de commande venant de la partie contrôle.

Une architecture de chemin de donnée est définie par un ensemble de composants *Instances* et un réseau d'interconnexions.

La représentation générique adoptée pour la partie opérative peut se définir sous la forme d'un 6-uple, représenté par la figure 4.8 :

- $\{UF\} = \{Ensemble\ des\ Unités\ Fonctionnelles\}$: il comprend toutes les unités nécessaires à l'exécution des opérations contenues dans la description comportementale;

```
1  (Solar AMICAL_for_DataPath
2     (DesignUnit nom_du_circuit_datapath
3        (View structure (ViewType RT-Level)
4           (Interface
5              {(Port (Array nom_du_connecteur longueur_de_bits)
6                 (Direction IN | OUT | INOUT)
7                 (Property PortType DATA | CONTROL)
8              )}
9              {(Port nom_du_connecteur
10                (Direction IN | OUT | INOUT)
11                (Bit)
12                (Property PortType DATA | CONTROL)
13             )}
14          )
15          (Contents
16             {(Instance nom_du_composant
17                (ViewRef Behavior RT-Level sorte_de_composant)
18                {(PortInstance nom_du_connecteur
19                   (Property PortType DATA | CONTROL))}
20                [(Property PlaceOrder entier)]
21             )}
22          )}
23          {(Net nom_d'interconnexion
24             (Joined
25                (PortRef nom_du_connecteur
26                   (InstanceRef nom_du_composant))
27                (PortRef nom_du_connecteur
28                   (InstanceRef nom_du_composant)))
29          )}
30       )
30    )
30 )
```

Figure 4.7: *Format SOLAR de la partie opérative générée par AMICAL.*

- $\{US\} = \{Ensemble\ des\ Unités\ de\ Stockage\}$: il contient les valeurs des variables générées et consommées au cours de l'exécution des opérations;
- $\{UC\} = \{Ensemble\ des\ Unités\ de\ Communication\}$: il se compose des unités qui entrent en jeux durant les transferts de données au sein du chemin de données; ces unités contrôlent l'exécution de chaque transfert;
- $\{UE\} = \{Ensemble\ des\ Unités\ de\ connexion\ Externes\}$: ce sont les ports externes;
- $\{INT\} = \{Réseau\ d'Interconnexion\}$: il relie entre eux unités fonctionnelles, unités de stockage et unités de communications;
- $MT = Modèle\ de\ Transfert$: il décrit les opérations de base exécutées par le chemin de données et visibles par le contrôleur. Il donne le schéma du modèle de transfert

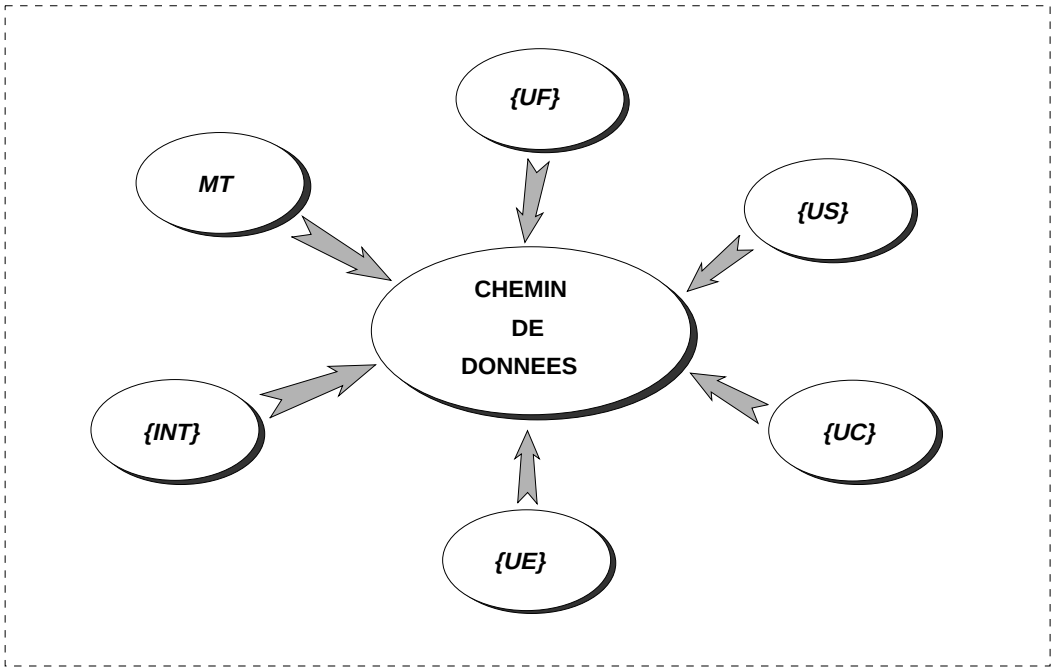


Figure 4.8: Composition du chemin de données

de registre à registre. Il comprend :

1. l'ensemble des instructions, qui défini les dépendances entre sources et destinations pour les transferts;
2. les règles de génération qui définissent la réalisation d'un transfert.

Chaque unité est définie de manière abstraite, sous forme d'une boite noire ou instance à laquelle est attachée un certain nombre d'informations nécessaires à la bonne marche du processus de synthèse et permettant d'évaluer les performances de la description post-synthèse.

- Unité fonctionnelle :

  Une unité fonctionnelle peut être décrite sous forme d'un 2-uple :

$$uf = (O, P), \quad uf \in \{UF\} \tag{4.2}$$

où O est l'ensemble des opérations pouvant être exécutées par l'unité, et P est l'ensemble des ports.

Chaque opération $Op \in O$ est définie par :

- *son nom* : Op_n;
- *les signaux de contrôle ou de sélection* : Op_{cont} : ils conditionnent l'exécution de chaque opération au sein de l'unité fonctionnelle;
- *sa description* : Op_{desc} : précise ce que réalise cette opération.

Cette description autorise l'exécution de plusieurs opérations par une même unité (alu, ...) ainsi que l'exécution d'une même opération par plusieurs unités différentes; dans ce dernier cas, le choix d'une unité particulière au cours de la synthèse, pourra être conditionné par la valeur des paramètres d'évaluation de performance.

Chaque port $p \in P$ est défini par :

- *son nom* : p_n;
- *sa direction* : p_{dir} : entrée, sortie, entrée/sortie;
- *sa taille* : p_{bit};
- *son type* : p_t : donnée, contrôle ou compte rendu d'exécution.

La figure 4.9 illustre le modèle générique d'une unité fonctionnelle défini ci-dessus [KDJ94]:

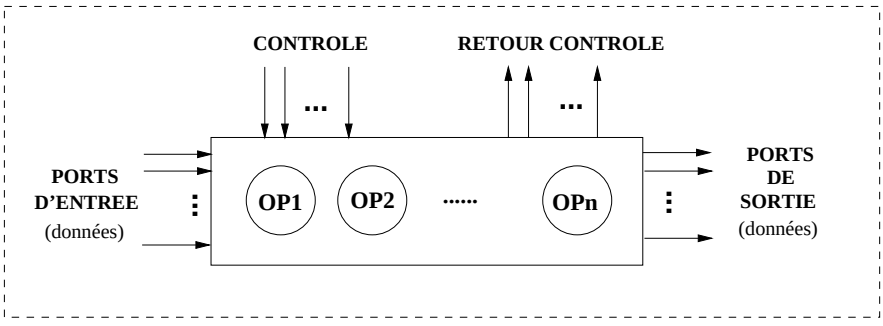


Figure 4.9: Représentation d'une unité fonctionnelle

De plus, chaque unité se verra associer un ensemble de paramètres, comprenant ceux nécessaires à l'évaluation des performances. Il est aussi possible d'avoir un ensemble

de paramètre d'évaluation des performances Op_{par} associé à chaque opération d'une même unité.

Toute unité, autre que fonctionnelle, peut se décrire de façon similaire : nom ou type, paramètres associés, signaux de contrôle, description des ports.

- Unité de Stockage

Les unités de stockage tels que registre, fichiers de registres, RAMs et ROMs, sont les composants permettant de stocker les valeurs des variables et constantes.

Fichiers de registres, ROM et RAM, éléments de mémoire plus complexes, sont décrits comme des unités fonctionnelles particulières pouvant exécuter des opérations liés aux protocoles d'accès mémoire par adressage, par exemple *Lire (add)* et *Ecrire (donnée, add)*.

La figure 4.10 donne le modèle générique de ce type d'unité de stockage.

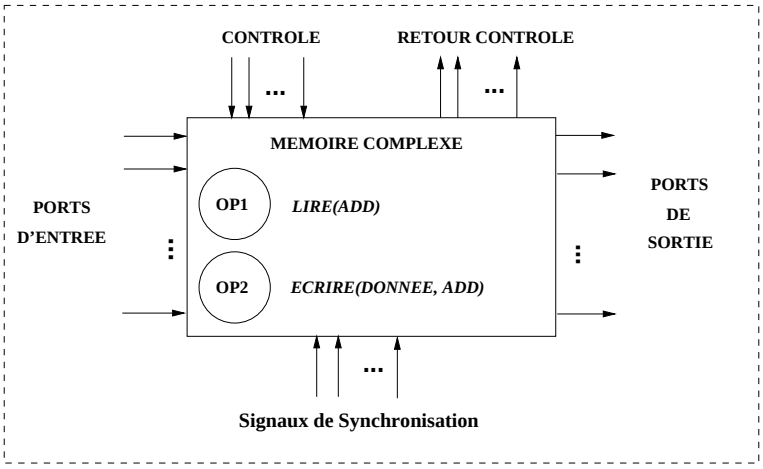


Figure 4.10: Représentation d'une unité de stockage complexe

Les registres simples et registres de constantes sont décrits comme des unités de stockage. Leurs description, très proche de celle d'une unité fonctionnelle, se distingue par leur type, us_t associé à chaque unité $us \in \{US\}$. Un exemple d'ensemble d'unité de stockage est donné dans la section 4.3.

- Unité de communication

Les unités de communication sont utilisées pour contrôler les transferts de données entre les diverses unités.

La sélection des unités de communication détermine la topologie des interconnexions. On rencontre, en général, deux types de modèles :

1. *Modèle réalisant un partage des interconnexions* : représentation à base de bus où le même bus peut être partagé entre plusieurs transferts de données, et cela sans création de conflits; les bus sont alors dits "à interconnexions partagées" (*shared-interconnetions*). Cette méthode maximise le partage des ressources. Quelques bus peuvent éventuellement être divisés en segments de telle façon que chaque segment de bus puisse être utilisé lors de transferts en parallèle. Quand un bus est alimenté par plusieurs sources, le contrôle des conflits d'accès se fait à travers l'utilisation de commutateurs. Le principal problème lié à cette méthode est la complexité des algorithmes d'allocation des connexions.
2. *Modèle point-à-point ou à base de multiplexeurs* : on réalise une interconnexion et une seule entre deux ports quelconques d'unités fonctionnelles et/ou de stockages. Cette méthode simplifie les algorithmes d'allocation des connexions : une connexion est crée entre deux unités fonctionnelles et/ou de stockages seulement si besoin est; si un port d'entrée d'unité se voit attribuer plus d'une connexion, donc possède plusieurs sources, un multiplexeur est employé.

Chaque unité de communication réalise une fonction combinatoire f qui détermine les transferts entrée-sortie orchestrés par les signaux de contrôle :

$$sortie = f(entrée(s), signaux\ de\ contrôle) \tag{4.3}$$

La figure 4.11 illustre la représentation générale d'une telle unité.

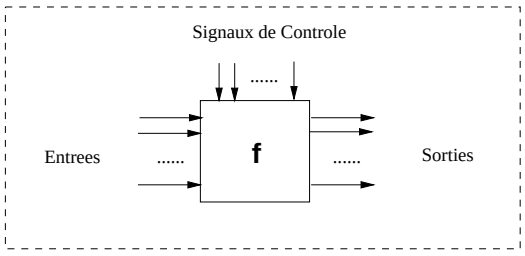


Figure 4.11: Schéma général d'une unité de communication

Nous traiterons ici deux sortes d'unités de communication :

- *Les commutateurs unidirectionnels* : ils ne peuvent transférer les données que dans une direction (une entrée, une sortie);
- *Les multiplexeurs* : ils contrôlent les transferts de données issus de multiples sources vers un seul port de sortie.

A chaque unité $uc \in \{UC\}$ est aussi associé un ensemble de caractéristiques similaires à ceux déjà définis pour les précédents types d'unités, notamment son type uc_t.

- Connexion

Les connexions sont les supports des transferts entre les diverses unités ainsi qu'entre les unités et les ports externes. Les connexions sont matérialisées que des fils.

Nous considérerons deux sortes de connexions :

- *Multi-sources ou Bus* : L'accès au bus doit être protégé par l'utilisation de commutateurs afin d'éviter d'éventuels conflits.
- *Source unique ou fil d'interconnexion*

- Unité de connexion externe

Les unités de connexion externes réalisent les liens entre les diverses unités et les ports externes.

La fonction de transfert u_f détermine les transferts entrée-sortie supervisés par les signaux de contrôle.

$$S = u_f(E, signaux\_de\_sélection) \tag{4.4}$$

où E est l'entrée de l'unité de connexion externe et S est la sortie de cette même unité.

La figure 4.12 montre plusieurs unités de connexion externe possibles :

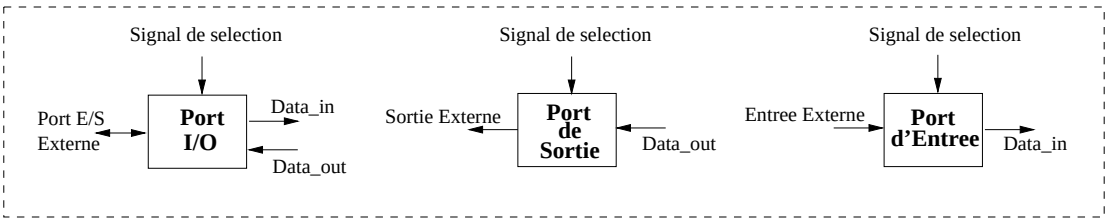


Figure 4.12: Schéma d'unités de connexion externe typiques

Ces unités peuvent être utilisées pour exécuter des transformations simples (conversion de type de donnée, modification du nombre de bits), aussi bien que pour des transformations complexes telles que conversions AD ou DA. En plus des signaux cités plus haut ces unités peuvent aussi avoir des signaux de synchronisation dans le cas où les unités sont mémorisantes.

En plus des informations structurelles, la définition du chemin de données comprend un ensemble d'instructions (voir la section 4.3).

Au cours de chaque cycle de base, la partie opérative exécute une instruction composée d'un ensemble de transferts qui se déroulent en son sein. Une instruction se compose :

- *d'une source* pouvant être une unité fonctionnelle, une unité de stockage ou une unité externe;
- *d'une destination* pouvant être une unité fonctionnelle, une unité de stockage ou une unité externe;
- *d'un chemin d'interconnexion* : ce dernier se compose d'unités de communication ($\{UC\}$) et de connexions.

Cet ensemble d'instructions exécutées par un chemin de données peut être défini par la donnée de l'ensemble des transferts autorisés par le chemin de données (voir figure 4.14)et par un ensemble de règles de génération de transferts définissant la réalisation effective des transferts d'après l'ensemble d'instructions (voir figures 4.16 et 4.19).

4.2.4.2 *Génération de la partie opérative : aspects généraux*

Pour générer la partie opérative, trois types d'informations sont nécessaires. Le premier type d'information porte sur la déclaration de l'interface, le second sur celle des composants utilisés, et le dernier sur celle des connexions entre les composants.

Selon l'architecture demandée, divers algorithmes sont utilisés dont la différence essentielle réside dans la génération du réseau de communication basé sur différents composants de connexion. En ce qui concerne la génération de la liste de connexions, deux différents algorithmes sont proposés en fonction de l'architecture choisie, sur lesquels nous reviendrons au cours des sections suivantes, et qui sont résumés dans l'annexe C.

4.3 Génération d'architecture

Au cours de cette section, nous allons nous focaliser sur deux styles d'architecture de chemin de données, à savoir une architecture à base de multiplexeurs et une architecture à base de bus, chacune d'entre elles donnant lieu à des considérations différentes. Nous verrons ensuite les solutions adoptées pour la génération d'une architecture microprogrammable dont le but principal est d'obtenir un contrôleur reprogrammable pour une partie opérative donnée, pouvant être l'une des deux précédentes.

Nous considérerons un ensemble d'unités de stockage $\{US\}$ composé de trois types de registres : *registre de constante*, *registre de variable* et *registre de compte-rendu* (flag-register). La figure 4.13 illustre ces trois types de registres.

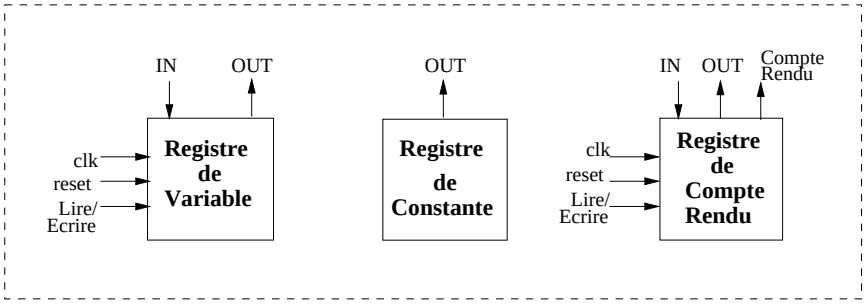


Figure 4.13: Ensemble des unités de stockage considéré : {US}

- *Un registre de constante* est utilisé pour générer une constante; il se caractérise par un seul port de sortie de donnée;
- *Un registre de variable* a les mêmes caractéristiques qu'un registre normal, autorisant lecture et écriture; le signal *Ecrire* sélectionne l'opération autorisant la sauvegarde d'une nouvelle valeur;
- *Un registre de compte-rendu* ressemble à un registre de variable, mais possède un autre port de sortie. Ce port de sortie de la valeur du registre est nécessaire lorsque le contrôleur prend en compte des valeurs générées par la partie opérative pour évaluer les conditions.

4.3.1 Jeu d'instructions et types de transferts

Les principales différences entre les deux modèles, multiplexeurs ou bus, vont se situer au niveau de l'ensemble d'instruction, et du réseau de communication.

Le modèle général commun aux deux styles et illustré sur la figure 4.14. les arcs du graphe designent les transferts autorisés.

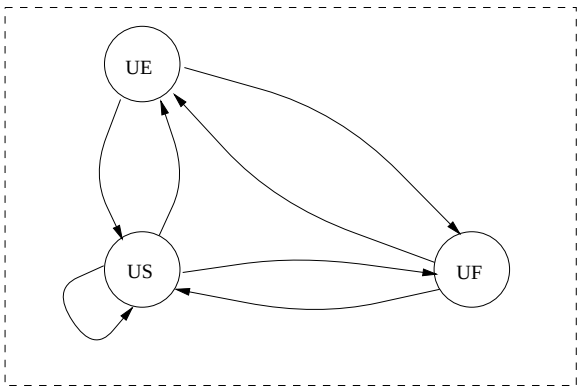


Figure 4.14: Types de transferts possibles

Ce modèle, très général, interdit cependant les transferts entre deux unités fonctionnelles ou deux unités externes sans passer par une unité de stockage intermédiaire.

4.3.2 Génération d'une architecture à base de bus

Les architectures orientées bus sont largement employées pour la synthèse de haut niveau. Les bus sont des ressources partagées pouvant être utilisées lors de transferts.

4.3.2.1 *Caractéristiques liées à ce style d'architecture*

Il y a deux sortes d'interconnexions au niveau de la partie opérative :

- *Les connexions internes*: connexions entre unités;
- *Les connexions externes*: connexions entre unités et ports externes;

Une connexion interne se compose de commutateurs unidirectionnels, qui font office d'unités de communication, et de bus, organes d'interconnexion. La figure 4.15 montre la structure d'un commutateur.

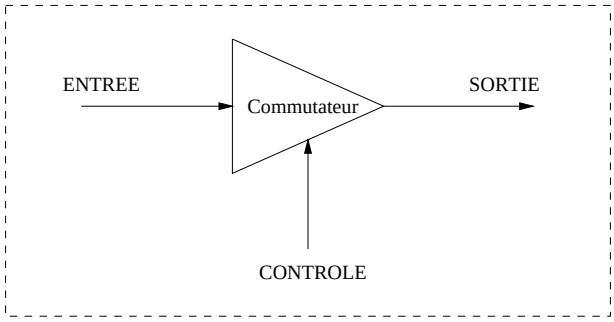


Figure 4.15: Modèle à base de bus : structure d'un commutateur

Les trois ports visibles sur la figure sont : un port de donnée d'entrée *ENTREE*, un port de donnée de sortie *SORTIE* et un signal de contrôle de sélection *CONTROLE*. Le commutateur agit comme une commande trois-états, dont la sortie peut être connectée à un bus. Ce modèle très simple peut être implémenté avec très peu de transistors.

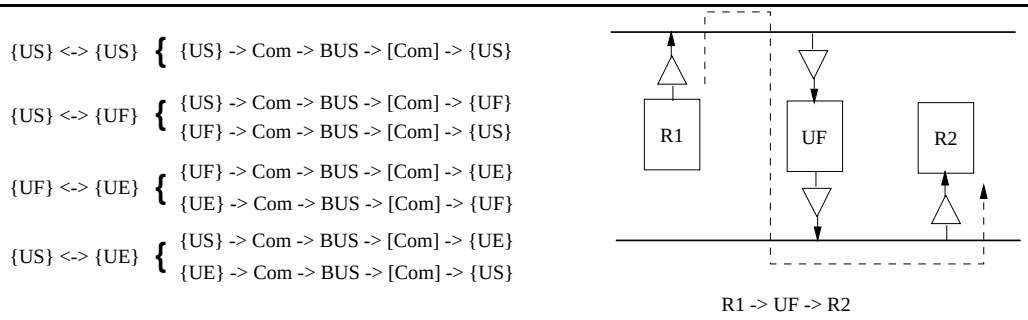
Les connexions externes fournissent les interconnexions entre unités et ports externes à travers des unités de connexion externes et des commutateurs.

Le modèle de l'ensemble d'instructions est illustré sur la figure 4.14; les règles de génération de transferts entre unités fonctionnelles, unités de stockage et unités externes permettent de faciliter les étapes telles que la préparation des données.

L'exécution de ces transferts est détaillée en figure 4.16.

où :
- Com : commutateur;
- BUS : bus;
- $\{US\}$: unités de stockage (registres);
- $\{UE\}$: unités de connexion externes;
- $\{UF\}$: unités fonctionnelles;
- $[Com]$: commutateur optionnel;

Des transferts plus complexes peuvent être réalisés en combinant plusieurs transferts de base. La figure 4.16 (b) montre un exemple de transfert complexe de type $\{US\} \leftrightarrow \{UF\} \leftrightarrow \{US\}$.



(a) Exécution des différents transferts possibles

(b) Exemple de transfert complexe R1 -> R2

Figure 4.16: Modèle à base de bus : Détail de l'exécution des différents transferts

4.3.2.2 *Génération de l'architecture*

Dans l'architecture basée sur les bus et les commutateurs, l'étape de placement de composants est déterminante pour minimiser le coût de connexions. L'allocation des connexions est organisée en trois sous-tâches séquentielles. La première sous-tâche consiste à trouver le nombre minimum de pistes de bus à allouer. La deuxième réalise l'allocation des commutateurs pour connecter les pistes de bus et les composants. Et la troisième est l'allocation des commutateurs entre les bus. L'algorithme est détaillé dans [Par92].

La figure 4.17 (b) montre une copie d'écran du chemin de données généré pour l'exemple du GCD. Il se compose de deux registres (x et y), d'une unité fonctionnelle (FU_1) et de trois ports externes (2 ports d'entrée x_i and y_i et un port de sortie ou). Le chemin de données comprend aussi douze commutateurs et trois bus.

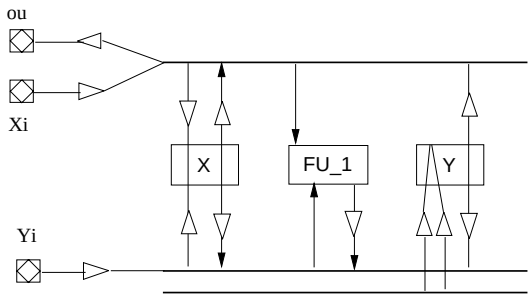
4.3.3 Génération de l'architecture à base de multiplexeurs

La topologie point-à-point ou architecture à base de multiplexeurs est la plus populaire pour la synthèse de haut niveau, car son emploi simplifie les algorithmes d'allocation. Dans cette topologie, on crée une connexion entre deux unités de stockage ou unités fonctionnelles seulement si besoin est [LCGM93].

Dans le cas où l'entrée d'une unité a N sources, un multiplexeur Nx1 peut remplacer

```
entity gcd is
 port (clk   : in bit;
       reset  : in bit;
       xi, yi : in integer;
       ou     : out integer);
end gcd;
architecture behavior of gcd is
begin
  process
     variable x,y: integer;
  begin
      wait for 0ns;
      x := xi;
      y := yi;
      while (x /= y ) loop
        if (x < y)
             then y:=y - x;
             else x:=x - y;
        end if;
      end loop;
      ou <= x;
  end process;
end behavior;
```

(a) Description comportementale du GCD



(b) Résultat de synthèse

Figure 4.17: Modèle à base de bus : Résultat de synthèse

N commutateurs. En terme de surface, un multiplexeur 2x1 est à peu prés équivalent à un commutateur. Ainsi, quand N tend vers une grande valeur, la solutions basée sur des multiplexeurs semble être la plus attractive.

Au lieu de commutateurs et de bus, le réseau d'interconnexion comporte seulement des multiplexeurs. En fait, un multiplexeur Nx1 est plus petit quéun commutateur au niveau de surface. En plus, il nécessite moins de fils de contrôles ($\lceil \log_2(N) \rceil$ au lieu de N).

4.3.3.1 *Caractéristiques liées à ce style d'architecture*

Il y a toujours deux sortent d'interconnexions au sein de la partie opérative :

- *Les connexions internes*: entre unités;
- *Les connexions externes*: entre unités et ports externes;

La base du réseau d'interconnexion est donc le multiplexeurs. La figure 4.18 montre la structure d'un multiplexeur.

Il se compose d'un certain nombre de ports d'entrée, d'un port de sortie et d'un signal de sélection *CONTROLE* qui sert à sélectionner une entrée. Le multiplexeur agit comme

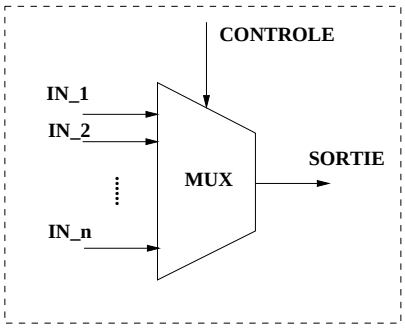


Figure 4.18: Modèle à base de multiplexeurs : Structure d'un multiplexeur Nx1

une fonction d'aiguillage N vers 1, dont la sortie peut être connectée à l'entrée d'un composant. Le nombre de bits nécessaire au signal de sélection est $\lceil \log_2(N) \rceil$.

Un multiplexeur Nx1 peut être réalisé de plusieurs façons. Par exemple, il peut être construit d'un arbre binaire de multiplexeurs 2x1.

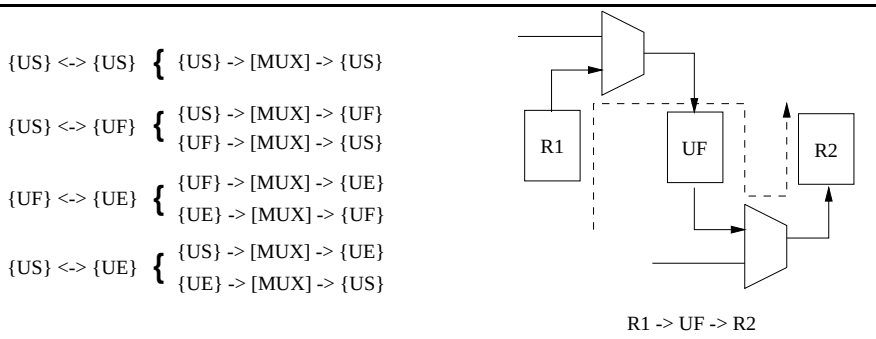
Avec ce modèle, on peut directement transférer des données entre ports externes et registres ou unités fonctionnelles à travers un seul multiplexeur. Cela permet de réduire le délai de transfert.

Utiliser des multiplexeurs pour connecter entre elles les unités correspond à la topologie point-à-point. Ce modèle permet quatre types de transferts de base au sein de la partie opérative, identiques au modèle précédent (voir figure 4.14).

La figure 4.19 détaille le modèle d'exécution pour ces transferts qui, par contre, diffère totalement du modèle à base de bus.

où :
- MUX : multiplexeurs;
- $\{US\}$: unités de stockage (registres);
- $\{UE\}$: unités de connexions externes;
- $\{UF\}$: unités fonctionnelles;
- $[MUX]$: multiplexeur optionnel;

En comparant ces résultats avec ceux du modèle à base de bus, on voit que l'ensemble des instructions est le même. Mais les règles de génération de transferts sont plus simples, ce qui nécessite moins d'unités pour réaliser un transfert.



(a) Exécution des différents transferts possibles

(b) Exemple de transfert complexe R1 -> R2

Figure 4.19: Modèle à base de multiplexeurs : Détail de l'exécution des différents transferts

De la même façon que pour le modèle précédent, il est possible, en combinant des transferts de base, de réaliser des transferts plus complexes.

Chaque transfert de données doit avoir un chemin d'inter-connexion de la source à la destination. Plusieurs transferts de données peuvent partager le même chemin d'inter-connexion si ils ne sont pas actif simultanément. En analysant la liste de transferts de la description, les multiplexeurs sont générés selon des destinations de transferts. Et pour chaque multiplexeur, le nombre d'entrées est déterminé par le nombre des sources de transferts connectées (voir figure 4.20).

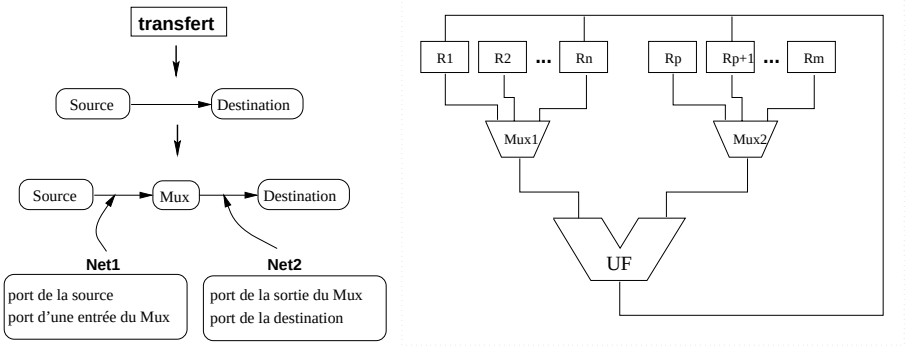


Figure 4.20: *Génération d'un multiplexeur*

Les algorithmes de génération de multiplexeurs, la partie opérative et le contrôleur sont expliqués dans la section C.1 en Annexe C.

La figure 4.21 présente les résultats de la synthèse de l'exemple du GCD utilisant ce modèle.

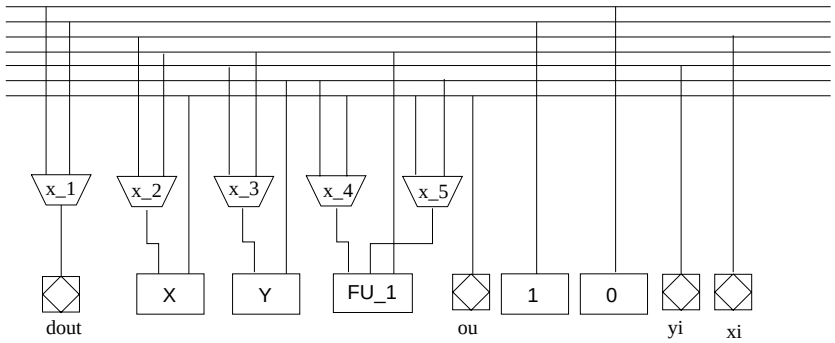


Figure 4.21: Modèle à base de multiplexeurs : résultat de synthèse

4.3.4 Génération de l'architecture micro-programmable

Le but de l'architecture micro-programmable est de générer un contrôleur reprogrammable en utilisant la même partie opérative. Nous nous intéresseront donc au cours de cette section, à la façon de concevoir le micro-contrôleur qui pourra être, à terme, adapté aux deux styles d'architecture de partie opérative vu précédemment.

Il est à noter que ce style d'architecture étant encore en cours d'évaluation, nous n'en verrons que les grandes lignes ainsi que ses avantages et inconvénients par rapports aux deux précédents styles architecturaux.

Cette architecture représente des avantages sur la conception des circuits intégrés ASIP[RBL+96].

- **la flexibilité de conception:** la programmabilité permet de faciliter la correction d'erreurs;
- **la réutilisabilité:** le circuit s'adapte à de nouvelles fonctions en modifiant le programme;

4.3.4.1 *Architecture du micro-contrôleur*

L'architecture de la partie contrôle est composée de (figure 4.22):

- **une ROM** contenant le code devant commander et synchroniser la partie opérative;
- **un Registre d'instructions** contenant les signaux commandant les transferts dans la partie opérative, l'adresse de la prochaine instruction et un indicateur du mode de calcul de l'adresse suivante;
- **un Séquenceur** calculant l'adresse effective de la prochaine instruction en fonction de la valeur du registre d'instructions.

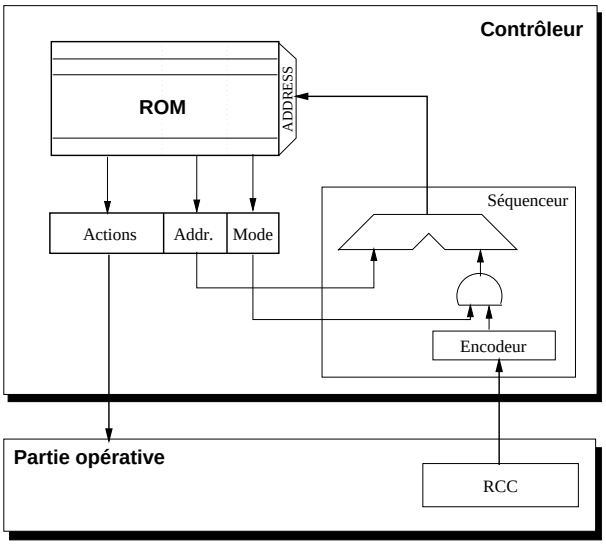


Figure 4.22: *Architecture du contrôleur micro-programmable*

Le séquencement des opérations entre la partie contrôle et la partie opérative se réalise sur deux ou trois cycles d'horloge (figure 4.23). A chaque front montant, en fonction de l'adresse de la transition courante (ADDR), la ROM génère l'instruction comprenant une partie "Actions" (ACT), une adresse suivante et un indicateur du mode (Mode). En même temps, le séquenceur calcule l'adresse pour la prochaine transition en fonction de la valeur de registre RCC (CC), l'adresse suivante donnée par la ROM et l'indicateur du mode. En parallèle, la partie opérative (PO) exécute les opérations commandées par ACT.

Dans cette architecture, il est à noter que:

- Les commandes envoyées par la partie contrôle vers la partie opérative pour calculer RCC doivent être mémorisées afin de synchroniser les événements entre les deux parties;
- En cas de dépendance d'une instruction à l'autre, l'insertion d'états intermédiaires est nécessaire (Mode = 1). Ceci permet l'attente du résultat du calcul de RCC.

L'exécution d'une opération nécessite deux ou trois cycles fondamentaux:

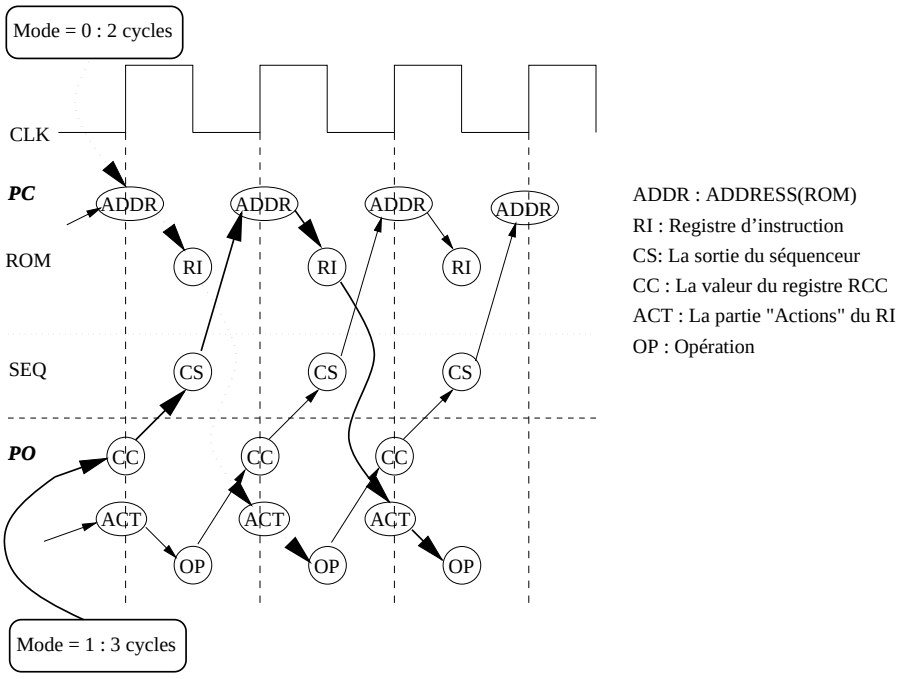


Figure 4.23: *Séquencement d'exécution*

- Premier cycle: Calcul de l'adresse de la prochaine instruction (Mode = 1);
- Deuxième cycle: Accès à l'instruction;
- Troisième cycle: Exécution de l'instruction dans la partie opérative.

Dans le premier cycle, si aucune dépendance existe entre deux instructions (Mode = 0), l'adresse suivante est calculée en parallèle pendant que la ROM envoie l'instruction (figure 4.23).

4.3.4.2 *Génération du micro-contrôleur*

La génération de la partie contrôle pour cette architecture se divise en plusieurs étapes, parmi lesquelles la plus importante est la génération du contenu de la ROM. L'architecture générée sera composée de plusieurs blocs.

Le séquenceur calcule l'adresse suivante en fonction de la nature de chaque transition. Si la transition concernée est simple (Mode = 0), l'adresse suivante sera celle qui est donnée par la ROM. Si les transitions sont multiples (Mode =1), sa valeur sera calculée

en ajoutant le contenu codé de RCC. Ce point doit être pris en compte lors de la génération de la ROM.

L'adressage et la génération des actions sont les deux points critiques dans la génération de la ROM. Comme mentionné plus haut, les adresses des transitions multiples doivent être codées consécutives (figure 4.24).

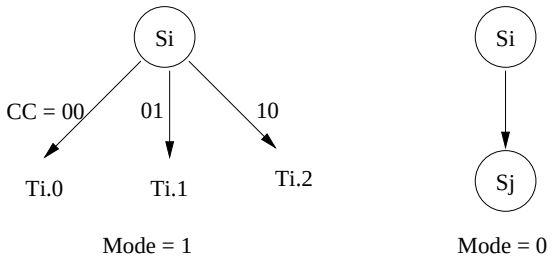


Figure 4.24: *Différents types de transitions*

Selon la valeur de Mode, l'adresse suivante sera soit une adresse simple fournie par la ROM, soit une adresse en tenant compte de la valeur du registre RCC (Addr. + RCC).

La génération des actions est fondée sur l'analyse des signaux de contrôle actifs selon la transition.

Les algorithmes de génération du contrôleur et de la ROM sont présentés dans la section C.2 de l'annexe C.

4.4 Comparaison entre architectures générées

A partir d'une représentation intermédiaire, selon le besoin de l'utilisateur, plusieurs architectures sont générées par AMICAL. Chaque architecture générée présente certaines propriétés qui lui sont propres qui pourront correspondre à certains types de conception.

L'architecture à base de multiplexeurs utilise des connexions privées. L'implantation des connexions par des multiplexeurs est souvent choisie pour des questions de rapidité de transferts: une connexion privée à travers un multiplexeur est plus rapide qu'une connexion commune par un bus. L'intérêt de l'utilisation de bus par rapport aux multiplexeurs est qu'un même bus sert à l'implantation de plusieurs connexions.

La table 4.1 présente une comparaison en terme de surface entre des architectures générées par AMICAL. **Opérateur** est une unité arithmétique "fixed-point" conçue à l'aide d'AMICAL. **Pid** est un **P**roportionnel **I**ntégral et **D**érivé. La technologie utilisée est AMS 0,8 (cyb). Les chiffres indiqués donnent le nombre de transistors de chaque solution.

Circuit	solution Bus	solution Mux
Opérateur	28132	22512
Pid	58044	51168

Table 4.1: *Comparaison des architectures générées par AMICAL*

On peut voir que la solution d'architecture à base de multiplexeurs donne des meilleures résultats. Ceci est du fait que la version actuelle d'AMICAL produit trop de commutateurs pour la solution à base de bus. Des travaux en cours doivent optimiser le nombre de commutateurs. On pourra aussi avoir des solutions à base de bus plus économiques en termes de surface.

4.5 Conclusion

Ce chapitre expose une étape importante du processus de synthèse : la transposition de la représentation du circuit du niveau transfert de registre à une architecture abstraite, après allocation des connexions.

Cette architecture, dite abstraite car générique et indépendante du temps explicite, est décrite dans un format intermédiaire : SOLAR. La flexibilité de ce format permet l'application d'une série d'algorithmes qui vont permettre de générer divers modèles d'architectures abstraites. Les deux premiers types d'architectures se focalisent sur la génération de chemins de données différents, l'un à base de multiplexeurs, l'autre à base de bus et de commutateurs (portes-trois-états). Chacun de ces chemins de données est orchestré par un contrôleur spécifique, décrit sous la forme d'une machine d'états finis, dont la spécificité découle du schéma de transferts propre à chaque partie opérative. Le troisième type d'architecture généré de manière abstraite se focalise sur le bloc de contrôle, décrit sous la forme d'un micro-contrôleur. L'objectif est clairement la possibilité de modifications supporté par ce type de structure : il suffit d'insérer un nouveau programme dans la ROM associée pour obtenir un nouveau comportement et ce à faible coût.

La transposition de cette architecture abstraite à une architecture finale synchronisée et décrite en VHDL fait l'objet de l'étape de personnalisation [Moh94].

CHAPITRE 5

Synthèse interactive

Chapitre 5

Synthèse interactive

Ce chapitre expose le concept d'interactivité au sein du système AMICAL, concept qui constitue une caractéristique puissante de ce dernier. Ce système est un assistant qui permet au concepteur de guider la synthèse au travers d'une interface graphique. Grâce à l'éditeur graphique d'AMICAL, une interaction "amicale" avec le concepteur est établie. Les informations diverses, par exemple, les décisions des algorithmes, la structure construite partiellement, les liaisons entre différentes représentations, ainsi que les résultats d'estimation, sont disponibles via des boites de dialogue. Le concepteur possède donc en permanence des informations sur les évolutions du processus de synthèse, et peux interagir avec le système pour éventuellement modifier le cours de celle-ci. Les résultats donnés par l'estimateur, à l'issus du processus de synthèse, vont donner une idée des performances du circuit obtenu qui pourront orienter le concepteur vers certaines modifications à apporter au cours d'une synthèse suivante.

5.1 Introduction

La notion d'interactivité nécessite la présence d'une interface qui dialogue avec l'utilisateur. Elle impose d'avoir la possibilité de réaliser certaines tâches, résumées par les points suivants :

- interpréter les commandes envoyé par l'utilisateur;
- montrer les résultats ou les propositions partiellement ou complètement;
- permettre à l'utilisateur d'intervenir sur certaines opérations;
- vérifier si une commande est permise, d'une part suivant le contexte, qui peut n'autoriser qu'un sous ensemble réduit d'actions (ex : le processus de synthèse ne pourra être lancé qu'aprés la compilation), et d'autre part suivant les conséquences possibles d'une action : certaines opérations doivent être interdites si elles provoquent une modification de la fonctionnalité spécifiée initialement.

Un système interactif est aussi un assistant de l'utilisateur. Selon l'étape, il doit permettre la visualisation des résultats ou des informations nécessaires. Il doit aussi supporter des fonctions d'analyse de résultats obtenus.

Plus qu'un outil de synthèse comportementale, le système AMICAL est un assistant à la conception, ainsi qu'il sera présenté au cours de ce chapitre. A travers cinq fonctions principales, qui sont le synthétiseur, l'éditeur graphique, le vérificateur, l'évaluateur et le documentaliste, ce système autorise trois styles de synthèse : automatique, interactive et manuelle.

- *Le synthétiseur* se charge de générer la partie opérative et la partie contrôle à partir de la description des macro-cycles. Cette description est obtenue après l'ordonnancement et l'allocation des registres du programme VHDL. Il organise le processus de synthèse en une succession de cinq étapes : le chaînage d'opérations qui permet de séquenceur l'exécution de certaines opérations parallèles, l'allocation des unités fonctionnelles qui réalise la sélection d'un ensemble d'unités fonctionnelles et détermine les liaisons avec les opérations, le micro-ordonnancement qui transforme chaque opération en une suite de transferts élémentaires, le placement des composants (architecture à base de bus) et l'allocation des connexions à l'issue de laquelle divers style architecturaux sont générés. Ces étapes s'effectuent automatiquement ou à l'initiative du concepteur qui garde le droit de regard.

- *L'éditeur graphique* propose une vision concrète du processus de synthèse et offre, outre le confort visuel, diverses facilités d'interaction avec le système.

- *Le vérificateur* est chargé de vérifier la cohérence des interventions de l'utilisateur. Cette fonction est nécessaire pour pouvoir combiner la conception automatique et la conception manuelle. A chaque étape du processus de synthèse, l'utilisateur doit respecter un certain nombre de règles de cohérence. Seules les interventions correctes sont acceptées.

- *L'évaluateur* analyse la structure synthétisée en se basant sur les contraintes spécifiées et donne des mesures estimées de surface, de vitesse et de consommation.

- *Le documentaliste* permet d'assister l'utilisateur, de comprendre et d'analyser la structure synthétisée. Il permet à l'utilisateur de voir la correspondance entre la structure et la description initiale durant toutes les étapes de la conception. Il permet aussi de produire plusieurs types de documents qui décrivent l'état d'avancement du processus de la synthèse, ainsi que les détails de la solution architecturale générée. Les commentaires du documentaliste sont donnés en mode graphique ou textuel.

L'objet de ce chapitre est de présenter les divers développements réalisés au sein d'AMICAL et visant à parfaire la relation *outil/utilisateur*.

Tout d'abord, sera présenté le modèle interactif disponible lors de l'utilisation d'AMICAL. L'éditeur graphique d'AMICAL permet à l'utilisateur d'intervenir au cours du processus de synthèse, d'accéder à des informations diverses comme par exemple les décisions des algorithmes, la structure construite partiellement, les liaisons entre différentes représentations, ainsi que les résultats d'estimations, etc.

L'environnement graphique d'AMICAL utilise la station SUN et la bibliothèque graphique OPEN LOOK (XVIEW). L'interaction se fait par l'intermédiaire d'un écran graphique s'adaptant sur plusieurs types de terminaux (Console de SUN, terminaux X, etc.). La configuration de l'écran par les différents types de fenêtres et la définition de menus permettent à l'utilisateur de communiquer facilement avec le système. Il est aussi possible

de lancer la synthèse avec un script (un ensemble de commandes) pré-défini ou bien d'exécuter en mode ligne de commande.

L'interactivité est favorisée par la présence d'un estimateur intégré à l'outil, permettant d'obtenir des mesures approchées des performances du circuit obtenu : cela fera l'objet de la section suivante, au cours de laquelle sera présentée le modèle de performance général intégré au système. Celui-ci donne, à l'issue de la synthèse, une évaluation des performances du circuit synthétisé incluant surface, vitesse, consommation et renseignements sur les composantes de la synthèse (taux d'utilisation des unités mise en jeu, nombre de micro-cycles, etc ...). Ces résultats sont fournis à la demande, en sélectionnant les commandes spécifiques à chaque estimation, sur des fenêtres spéciales et dans des fichiers séparés et réactualisés. Comparer entre eux des circuits issus de plusieurs synthèses d'une même spécification en est ainsi grandement facilité. De plus, cela permet d'avoir une indépendance plus grande vis à vis des outils de synthèse de plus bas niveau, dont l'utilisation sera optimisée en terme de temps de conception, les boucles de retour au cours des divers raffinement (méthodologie "Top-Down") étant minimisées.

Les informations fournies tout au long de la synthèse et les renseignements fournis à l'issue de celle-ci vont donner au concepteur les moyens de mieux appréhender les caractéristiques du circuit obtenu, et donc la possibilité d'agir sur celles-ci de manière optimale au cours des synthèses suivantes. La dernière section traitera donc de ces points et de la synthèse interactive permise à l'aide d'AMICAL.

5.2 L'environnement interactif d'AMICAL

5.2.1 Interface utilisateur

L'interface permet à l'utilisateur et à AMICAL de travailler ensemble tout au long du processus de synthèse. Les opérations laissées au soin de l'utilisateur sont:

- sélectionner une commande dans le menu,
- sélectionner un élément à manipuler,
- répondre à une question posée par le système,
- afficher la description comportementale,
- afficher la structure synthétisée partiellement ou complètement,
- afficher les liaisons entre ces deux représentations différentes (la partie opérative et la partie contrôle),
- afficher les informations sur le déroulement du processus du système,
- afficher des documents générés par le système et
- exécuter les commandes UNIX.

L'interaction se fait à l'aide d'un curseur à travers des aides graphiques, un menu hiérarchique, une fenêtre de dialogue, une fenêtre de document, une fenêtre UNIX et trois fenêtres principales (figure 5.1).

Les trois fenêtres principales sont présentes en permanence sur l'écran. La fenêtre du haut montre la description comportementale. La fenêtre centrale montre la structure de la partie opérative construite partiellement ou complètement. La fenêtre du bas donne quelques informations sur le déroulement du processus de synthèse: les messages d'erreur, la commande en cours, l'étape de la synthèse, le mode de synthèse en cours. Le multi-fenêtrage permet à l'utilisateur non seulement de voir les résultats intermédiaires de la

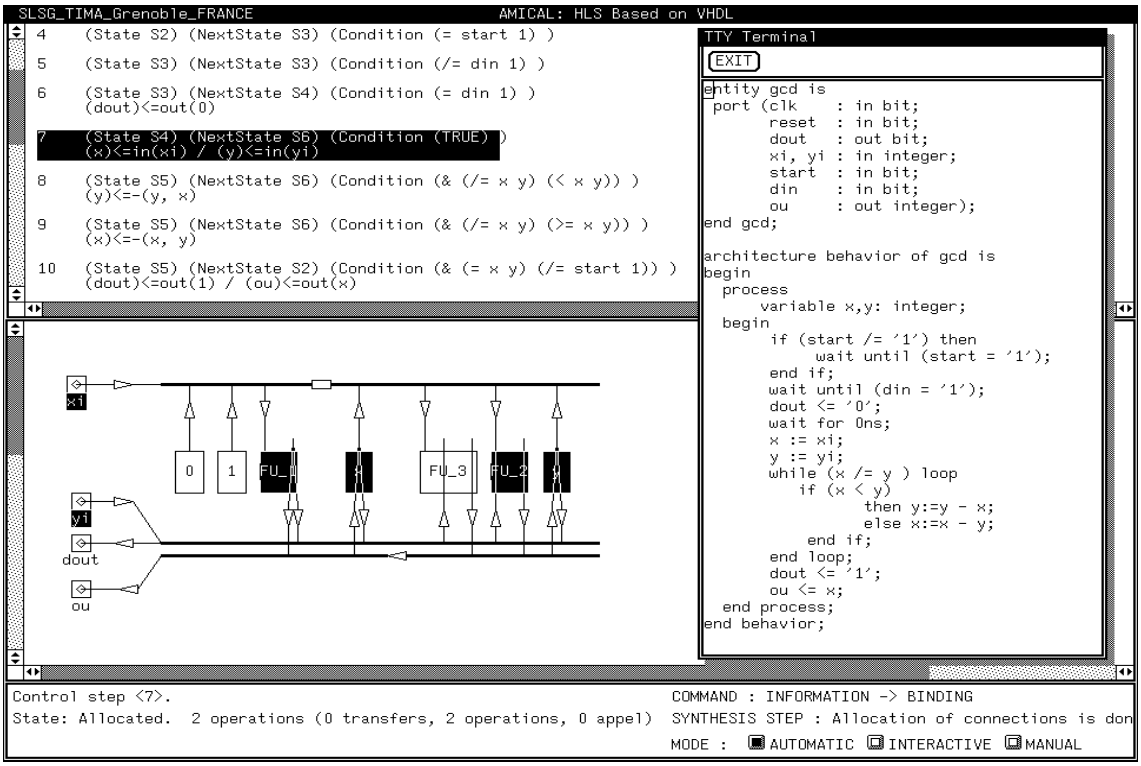


Figure 5.1: *Configuration de l'écran du système AMICAL.*

synthèse, mais aussi de voir les liaisons entre les éléments de la description comportementale et les composants de la structure.

Le curseur permet de sélectionner un élément appartenant à la description comportementale ou à la structure. Il permet aussi de choisir les commandes dans les divers menus déroulants.

Les menus contiennent toutes les commandes pouvant être exécutées par AMICAL. Ils sont classés et hiérarchisés. Les tâches similaires sont regroupées dans des sous menus. L'annexe C.2 montre l'organisation des menus et les tâches correspondant à chaque commande.

Les aides graphiques sont les dessins provisoires pour illustrer le traitement intermédiaire d'une commande. Ce sont des lignes, des rectangles blancs et des rectangles contrastés. Ces derniers sont utilisés pour marquer les éléments sélectionnés par l'utilisateur ou par le système.

La fenêtre de dialogue apparaît en bas de l'écran quand AMICAL a besoin d'un texte ou d'un choix parmi plusieurs options. Dès que la réponse d'utilisateur est entrée, cette fenêtre disparaît.

Les documents construits par AMICAL sont montrés dans une fenêtre de document à la demande d'utilisateur. Ces documents sont aussi sauvegardés dans la directoire "tmp".

AMICAL permet à l'utilisateur d'annuler chacune de ces étapes et de la relancer avec l'autres critères afin d'améliorer le résultat final ou de réaliser des compromis entre les différents coûts (surface et vitesse d'exécution). Par ailleurs l'utilisateur peut explorer l'espace des solutions en utilisant les trois modes de synthèse ou la fonction d'exploration automatique. Dans ce dernier cas AMICAL répète automatiquement la synthèse totale en faisant varier les paramètres et mémorise toutes les solutions obtenues.

5.2.2 Objets de base dans la représentation graphique

Du point de vue graphique, un objet de base est une entité que l'utilisateur peut sélectionner et manipuler. Dans la description comportementale, un objet de base peut être une variable, une opération, un macro-cycle, un transfert ou un micro-cycle. Dans la description structurelle, chaque composant correspond à un objet de base. Les objets de base entretiennent une relation très étroite avec les structures de données d'AMICAL.

5.2.2.1 *Objets de base dans la description comportementale*

Les objets de la description comportementale sont organisés hiérarchiquement. La description de macro-cycles contient trois types d'objets:

- les variables,
- les actions (transferts, opérations et appels)
- les micro-cycles.

Les opérandes et les résultats d'une opération sont des variables, y compris la source et la destination d'un transfert. Une action est aussi un objet contenant des variables. Un micro-cycle est un objet qui peut contenir plusieurs opérations parallèles.

Cette hiérarchie se retrouve au niveau des objets graphiques. Une variable occupe la plus petite surface rectangulaire élémentaire. Une action occupe un rectangle pouvant contenir des variables. De même, un micro-cycle peut contenir des actions.

Dans le cas où l'utilisateur indique une position pour sélectionner un objet, si la position appartient à plusieurs objets de base, AMICAL donne la priorité au plus petit objet (dans l'ordre variable, action et micro-cycle). Les objets de base peuvent être sélectionnés soit pour demander des informations, soit pour exécuter la synthèse manuelle.

Dans la description de micro-cycles, les objets de base sont les transferts et les micro-cycles. Un micro-cycle comprend un ensemble de transferts exécutés en parallèle.

5.2.2.2 *Objets de base de la description structurelle*

Dans la description structurelle (partie opérative), chaque composant est un objet de base. Cela peut être un connecteur externe, un registre, une unité fonctionnelle, un bus, un commutateur, un multiplexeur ou un fil. Chaque composant occupe une surface rectangulaire. Tous les objets de base ont la même priorité. Si l'utilisateur indique une position sur plusieurs objets de base, AMICAL sélectionne aléatoirement l'un d'eux.

5.2.3 Flot d'exécution global du système AMICAL

AMICAL permet un dialogue facile et efficace avec l'utilisateur. Une boucle d'interaction typique est montrée dans la figure 5.2.

A chaque itération, AMICAL vérifie la cohérence de la requête d'utilisateur et affiche ses effets. Certaines commandes nécessitent plusieurs interactions. Dans ce cas, les résultats partiels sont signalés à l'utilisateur sur l'écran. Le dialogue se termine par la sélection de la commande "QUIT". Une fois que toutes les données requises pour l'exécution d'une

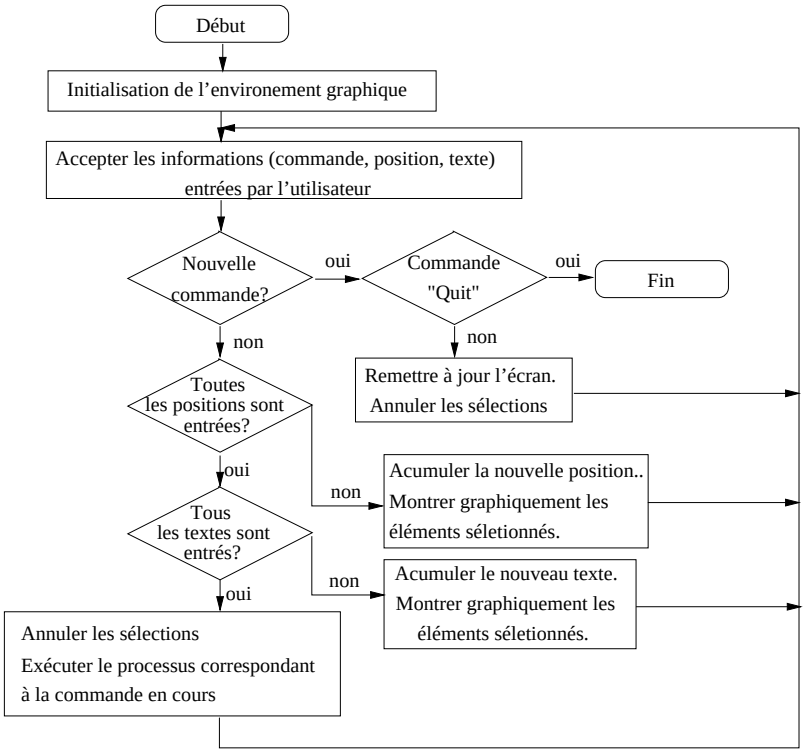


Figure 5.2: *Flot global du système AMICAL.*

commande ont été acquises, AMICAL exécute la commande.

Cette section a donc permis de présenter les possibilités environnementales d'AMICAL qui vont permettre une interaction efficace utilisateur/outil. L'objet des sections suivantes est d'expliquer quelles seront les données accessibles à l'utilisateur et par quels moyens ce dernier pourra s'en inspirer pour agir efficacement au cours du processus de synthèse.

5.3 Modèle de performance

La synthèse de haut niveau génère un modèle structurel traduisant une description comportementale donnée d'un système et qui satisfait les contraintes de conception telles que performance et surface. Des mesures de qualité sont nécessaires pour appuyer la synthèse de haut niveau de deux manières. Tout d'abord, des mesures sont nécessaires pour déterminer la qualité du circuit synthétisé final. Ainsi, les mesures permettent de comparer celui-ci avec des contraintes données et d'identifier les points critiques qui lui sont associées. De plus, les mesures de qualité sont utiles pour guider les outils de synthèse de haut niveau afin de rechercher la meilleure solution en comparant les divers modèles architecturaux[GDWL92, HT83].

Ainsi, les mesures de performances font partie intégrante de l'interactivité permise puisqu'elles vont conditionner celle-ci en partie. L'utilisateur y aura accès en fin de synthèse, au sein même de l'outil, sans passer par des moyens extérieur coûteux en terme de temps. Les estimations "on-line" rapides de surfaces et de délais des modules RTL conditionnent la sélection d' un composant, l'ordonnancement, et les décisions concernant l'allocation aussi bien que la comparaisons entre les différentes implémentations[JD93, SJ93].

Cette section a pour objectif de présenter de manière détaillées les modèles d'estimation intégrés au système. Les modèles d'estimation s'appuient sur des résultats expérimentaux (figure 5.3).

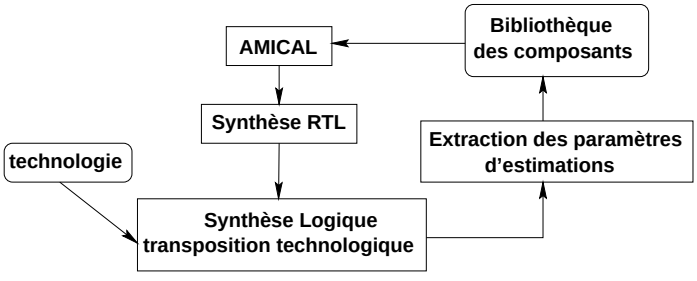


Figure 5.3: *Extraction des peramètres d'estimations.*

Trois sortes d'estimations sont visées :

- *Estimation de surface*;
- *Estimation temporelle*;
- *Estimation de puissance.*

5.3.1 Modèle d'estimation de la surface

Les mesures de surfaces traditionnelles divisent la surface de la conception en surface occupée par les unités actives et surface d'interconnexion. Les unités actives comprennent unités fonctionnelles et unités de stockage. L'interconnection se compose des unités d'interconnection : multiplexeurs, bus, commutateurs et fils[JD93, SJ93, NG92].

La surface totale d'unités actives est déterminée par la somme des surfaces des cellules. L'estimation de surface dépend de la bibliothèque technologique dont sont issues les cellules.

Le problème principal pour l'estimation de surface réside dans la mesure de la surface d'interconnection, puisque celle-ci dépend du placement et du routage réalisés pour obtenir le layout. Cette surface dépende aussi des outils de synthèse logique utilisés en aval de la synthèse comportementale.

L'unité de surface adoptée est le nombre de transistors à la place du micron-carré. Cela permet d'obtenir une estimation indépendante de la technologie. La conversion (nombre de transistors)-(unité carrée) peut se faire simplement par la multiplication par un facteur dépendant de la technologie.

La formule d'estimation de la surface pour la partie opérative peut s'écrire sous la forme :

$$AD = \alpha \times \sum Sizeof_d(D_i) \tag{5.1}$$

où :
- D_i est l'ensemble des unités actives du chemin de données;
- α est un facteur expérimental dépendant de la synthèse RTL du placement-routage et de la technologie;
- $Sizeof_d$ est une fonction retournant le nombre de transistors de chaque élément de D_i.

Le facteur α est déterminé expérimentalement et traduit les diverses optimisations effectuées au cours des synthèses de plus bas niveau.

Dans notre représentation, le chemin de données se compose d'unités fonctionnelles, de stockage, d'intercommunication et de connexion externe. Au vu de cette définition, la fonction d'estimation de la surface pour chaque type d'unité dépend de (voir la définition dans la section 4.2.4) :

$$UF = \sum Sizeof_d(Op_{pa}, p_b); \tag{5.2}$$

$$US = \sum Sizeof_d(us_t, p_b); \tag{5.3}$$

$$UC = \sum Sizeof_d(uc_t, p_b); \tag{5.4}$$

$$UE = \sum Sizeof_d(p_b); \tag{5.5}$$

La surface du contrôleur, décrit selon le modèle FSM, peut être estimé connaissant : le nombre d'entrées et de sorties, le nombre d'états, et la logique de contrôle. En s'appuyant sur le modèle de contrôleur défini plus haut, l'estimation de sa surface peut être calculée de la façon suivante :

$$\begin{aligned} AC &= f(nb\_states, inouts, nb\_transitions) \\ &= \beta(Sizeof_c(\lceil \log_2(nb\_states) \rceil + inouts)) \times (nb\_transitions) \end{aligned} \tag{5.6}$$

où :
- nb_states est le nombre d'états du contrôleur;
- $inouts$ sont les ensembles des ports d'entrée et de sortie du contrôleur ;
- $nb\_transitions$ est le nombre de transitions;
- β est un facteur expérimental dépendant de l'implémentation des conditions de contrôle et de l'optimisation logique;
- $Sizeof_c$ est une fonction qui estime la surface requise pour un registre d'état.

Donc la surface totale le l'architecture est donnée par :

$$AA = \gamma(AD + AC) \tag{5.7}$$

où γ est un facteur dépendant de l'interface entre chemin de données et contrôleur. Dans le cas d'une architecture où le chemin de données est connecté directement au contrôleur (sans pipeline par exemple), ce facteur vaut 1.

5.3.2 Modèle d'estimation des performances temporelles

L'estimation temporelle est un autre important problème pour l'estimation des performances générales d'un circuit donné. Elle peut fournir l'information nécessaire pour déterminer les paramètres de synchronisation tels que fréquence d'horloge et chemin critique. Dans cette section on ne considère que l'estimation du temps de cycle. Pour être complète l'estimation doit aussi considérer le nombre de cycle nécessaire pur la réalisation d'une fonction donnée. Une telle estimation peut nécessiter des analyses dynamiques dans le cas où on le calcul la dépendance des données. Par contre l'estimation du temps de cycle peut se faire de manière statique.

Cette estimation dépend du temps requis pour le plus long transfert à l'intérieur du chemin de données et pour la génération des signaux de contrôle.

Chaque transfert implique un certain nombre d'unités fonctionnelles et/ou de stockage ainsi que le réseau d'interconnection. Pour le chemin de données, le temps de transfert le plus long est déterminé par le temps d'exécution des opérations, le modèle de transfert et le délai de liaison. Cela signifie que trois types de délai doivent être considérés :

- Le délai d'exécution des unités fonctionnelles;
- Le délai d'exécution des unités de stockage;
- Le délai de propagation du réseau d'interconnection;

Dans ce modèle, le délai d'exécution de toutes les unités fait partie de leurs spécifications.

Le délai dû au réseau de communication résulte de deux types d'éléments : les unités de communication, et les liaisons.

Nous définissons les délais suivants :

Le délai d'une opération dans une unité fonctionnelle :

$$t_{op} = f(Op_t) \tag{5.8}$$

où $Op_t \in Op_{par}$ est le paramètre temporel de l'unité fonctionnelle qui est défiué dans la définition 4.2 du chapitre 4.

Le délai d'une unité de stockage:

$$t_{reg} = g(us_t) \tag{5.9}$$

où us_t représente le type d'unité de stockage.

Le délai d'une unité de communication:

$$t_{comm} = h(uc_t, uc_f) \tag{5.10}$$

où uc_t est le type d'unité de communication; uc_f est la fonction de transfert.

En supposant qu'un chemin de transfert impliquera des éléments des ensembles d'unités fonctionnelles UF, d'unités de stockage US, et d'unités de communication UC, on peut calculer le temps de délai total pour un transfert donné, de la façon suivante :

$$\begin{aligned} T_i &= \sum(T_{UF} + T_{US} + T_{UC}) \\ &= \delta(\sum(t_{op}) + \sum(t_{reg}) + \sum(t_{comm})) \end{aligned} \tag{5.11}$$

où δ est un facteur expérimental dépendant de la synthèse logique et du placement-routage.

Et donc le plus long délai de transfert, considérant n transfert, sera donné par :

$$T_{DP} = max(T_i), \quad i = 1, \ldots, n \tag{5.12}$$

où T_i est le délai du i-ème transfert.

Pour le contrôleur, c'est la logique de contrôle et le registre d'état qui conditionnent le temps de délai. Le temps de délai du registre d'état est similaire à celui d'une unité de stockage.

Concernant la logique de contrôle, nous utilisons là aussi une approximation basée sur les entrées/sorties et les états du contrôleur :

$$t_{cl} \quad = \quad \lambda f(Entrées, Etats, Sorties, Transitions) \tag{5.13}$$

Où λ est un facteur expérimental calculé en fonction de la technologie et de l'outil de synthèse logique utilisé.

Donc le délai d'exécution pour un changement d'état est :

$$T_C = t_{reg} + t_{cl} \tag{5.14}$$

En considérant ces deux parties, le délai minimal d'horloge du système sera :

$$T_{system} = max(T_{DP}, T_C) \tag{5.15}$$

Avec ce modèle, plusieurs paramètres temporel tels que la fréquence d'horloge et le délai du plus long transfert peuvent être estimés.

5.3.3 Modèle d'estimation de la puissance consommée par le circuit

Les points principaux du modèle d'estimation de la consommation pour AMICAL sont illustrés sur la figure 5.4 [Gui95]. Le circuit issu de la synthèse va être décomposé en quatre parties distinctes, dont l'estimation de la puissance consommée pose des problèmes distincts, à savoir : partie opérative, contrôleur, interconnection et horloge.

Un autre élément est à considérer : la détermination de statistiques au niveau du circuit complet permettant de tenir compte de l'aspect dynamique de l'énergie/puissance consommée. Ces statistiques seront de deux types : l'un, que l'on peut qualifier de structurel, et l'autre concernant la machine d'états finis (MEF).

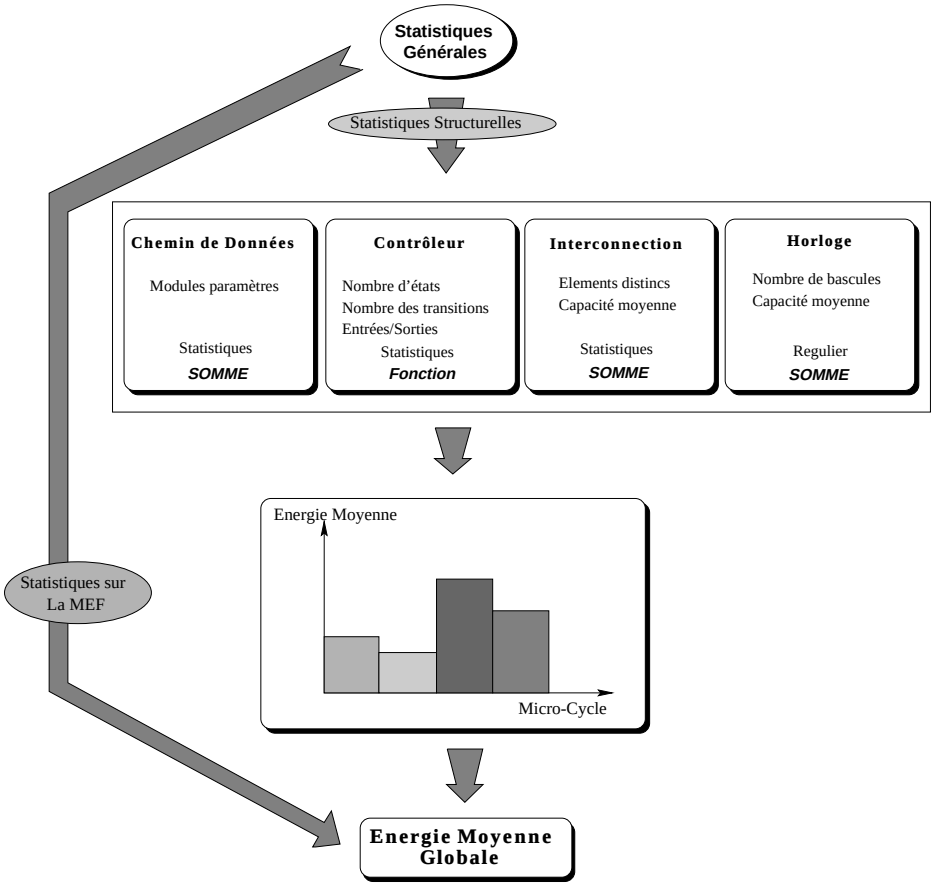


Figure 5.4: Vue globale de l'estimation de la consommation

Pour chaque partie distincte, et pour un cycle i donné, on obtient l'énergie consommée par l'intermédiaire de sa capacité moyenne commutée ou C_{SW}. Pour obtenir l'énergie

consommée totale par cycle élémentaire, il suffit d'ajouter ces divers résultats, sachant que les quatre parties distinguées pour simplifier l'estimation agissent ensembles au cours d'un cycle.

On aura donc :

$$C_{SW_{Totale}}(i) = C_{SW_{PO}}(i) + C_{SW_{Cont}}(i) + C_{SW_{INT}}(i) + C_{SW_{Clock}} \tag{5.16}$$

où:

$C_{SW_{PO}}$ est la capacité moyenne commutée pour la partie opérative

$C_{SW_{Cont}}$ est la capacité moyenne commutée pour la partie contrôle

$C_{SW_{INT}}$ est la capacité moyenne commutée correspondant au réseau d'interconnexion

$C_{SW_{Clock}}$ est la capacité moyenne commutée liée au fonctionnement de l'horloge

Chacune de ces donnés est obtenue pour chaque cycle élémentaire de la machine d'états finis.

Le schéma de la figure 5.5 donne une vision tangible des mesures obtenues.

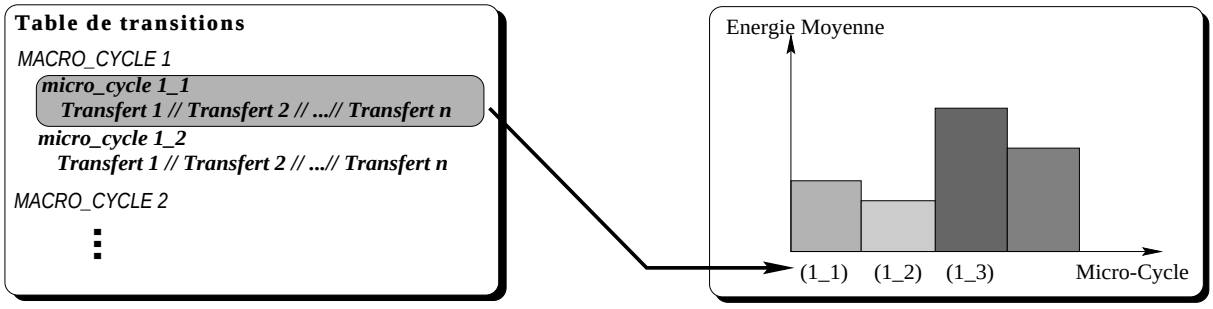


Figure 5.5: Energie par cycle élémentaire du circuit

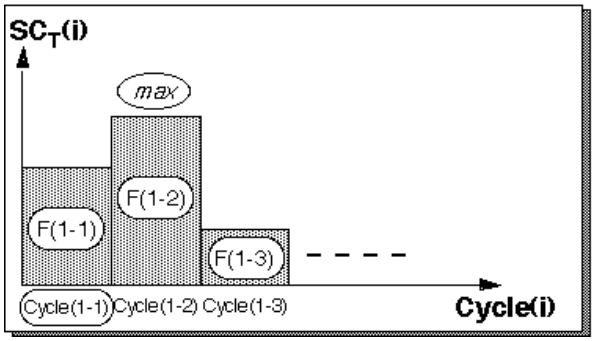
Cet ensemble de mesures peut déjà fournir une aide précieuse, notamment par la connaissance des cycles consommant le plus, sur lesquels on peut décider d'agir.

Un cycle présentant une consommation élevée nécessitera d'autant plus d'être modifié que sa probabilité d'être exécuté dans un fonctionnement typique sera grande. Il est donc nécessaire d'avoir des probabilités d'exécution de chaque transition ou micro_cycle de la machine d'états. Celles-ci sont susceptibles d'être obtenues de deux façons :

1. En résolvant les équations de Chapman-Kolmogorov [TPD94, MD94];

2. Par simulation du circuit complet, en même temps que les statistiques structurelles[Inc93].

Cette dernière solution semble être la plus appropriée. Il s'agit en fait de compter combien de fois un micro_cycle est sollicité, et de diviser par le nombre total de cycles mis en jeu au cours de la simulation. La somme de ces probabilités est bien entendu égale à 1. Ceci est illustré par la figure 5.6.



Cycle(i)	$SC_T(i)$	F(i)
Cycle(1)	$SC_T(1)$	F(1)
Cycle(2)	$SC_T(2)$	F(2)
Cycle(3)	$SC_T(3)$	F(3)
⋮	⋮	⋮
		$\sum F(i) = 1$

Figure 5.6: Illustration de la fréquence d'exécution associée à chaque micro cycle

5.3.4 Résultats Expérimentaux

Comparant les résultats donnés par AMICAL avec ceux obtenu grâce au outils de synthèse RTL, la table 5.1 montre que les résultats calculés sont relativement précis pour ce qui est de la surface. Les valeurs indiquées donnent le nombre de transistors.

Circuit	*Modèle à base de bus*		*Modèle à base de multiplexeurs*	
	Surface		Surface	
	Estimation	Réel	Estimation	Réel
Opérator	21052 (-25%)	28132	18235 (-19%)	22512
Pid	50133 (-14%)	57816	43040 (-16%)	51168

Table 5.1: Comparaison entre résultats calculés et obtenus après synthèse

Les deux autres modèles n'ont pas été complètement implémentés et donc testés.

Les trois modèles d'estimation présentés au cours de cette section fourniront à terme à l'utilisateur un ensemble de mesures approchées dont l'utilisation sera abordée au cours

de la section suivante, qui traite de la méthodologie basée sur l'interactivité et autorisée par l'outil faisant l'objet de ce travail de thèse : AMICAL.

5.4 Synthèse interactive

L'interactivité est une des forces du système AMICAL. Cette section discute de l'art et la manière d'exploiter ce concept.

AMICAL permet à l'utilisateur de réaliser la synthèse en trois modes différents: automatique, interactif et manuel.

En mode automatique, les étapes de synthèse sont enchaînées sans intervention d'utilisateur. En mode interactif, les tâches sont exécutées automatiquement en mode pas à pas. Les cadences sont contrôles par l'utilisateur. Ce mode permet à l'utilisateur à la fois de suivre l'évolution du processus de synthèse et d'intervenir pour modifier les décisions des algorithmes de synthèse. En mode manuel, l'utilisateur prend en charge tous les processus de synthèse. Dans ce cas, le système agit en tant qu'assistant : il vérifie la cohérence des décisions prises et ne permet que les opérations correctes.

Au cours du processus de synthèse, certaines étapes autorisent certains modes à l'exclusion de tout autre comme l'illustre la figure 5.7.

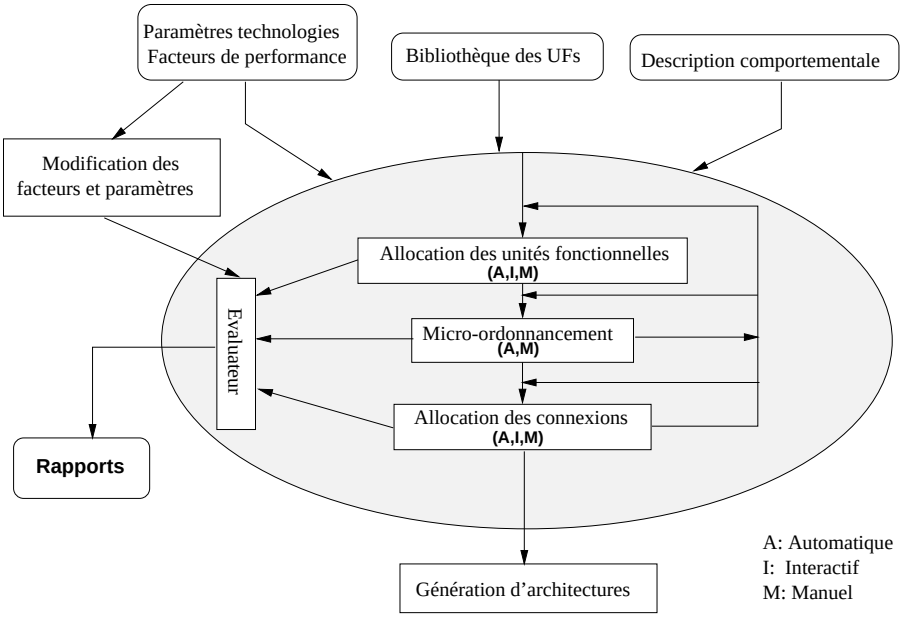


Figure 5.7: *Différents modes supportés par le système AMICAL.*

L'allocation des unités fonctionnelles et l'allocation des connexions supportent les trois

modes, tandis que le micro-ordonnancement s'effectue de manière exclusivement automatique. Cependant, la possibilité est offerte au concepteur de modifier manuellement ce dernier si besoin est, par exemple en séquentialisant les transferts d'un micro-cycle si le parallélisme de la solution n'est pas optimal selon lui.

L'introduction des modes interactif et manuel impose un certain nombre de contraintes liées notamment à la vérification de la cohérence des actions du concepteur. Ces modes introduisent cependant une souplesse non négligeable puisqu'à tout moment au cours de la synthèse, le concepteur pourra revenir en arrière en se basant, par exemple, sur les résultats fournis par l'estimateur.

Les sections suivantes se proposent d'éclaircir ces concepts, en approfondissant les notions d'allocation interactive et en détaillant les mesures fournies par l'estimateur.

5.4.1 Allocation et interactivité

Cela concerne les deux types d'allocations du processus de synthèse : allocation des unités fonctionnelles et des connexions.

5.4.1.1 *Allocation des unités fonctionnelles*

Comme illustré par la figure 5.7, l'allocation d'unités fonctionnelles et l'allocation des connexions peuvent être exécutés en mode automatique, interactif ou manuel (voir la figure 5.8).

L'algorithme automatique d'allocation des unités fonctionnelles associe un coût aux opérations pour sélectionner la meilleure unité fonctionnelle parmi celles de la bibliothèque. Dans ce mode, l'utilisateur n'intervient pas.

Le mode interactif permet à l'utilisateur de suivre l'algorithme itération par itération. A chaque itération, une nouvelle unité fonctionnelle est allouée.

En mode manuel, l'utilisateur sélectionne une unité fonctionnelle et un ensemble d'opérations pouvant être liées à cette unité fonctionnelle. Cette sélection se fait de manière

interactive. A chaque fois que l'utilisateur sélectionne une unité fonctionnelle, AMICAL fait apparaître toutes les opérations qui peuvent lui être liées. De même, à chaque fois que l'utilisateur sélectionne un ensemble d'opérations, AMICAL montre les unités fonctionnelles pouvant exécuter ces opérations.

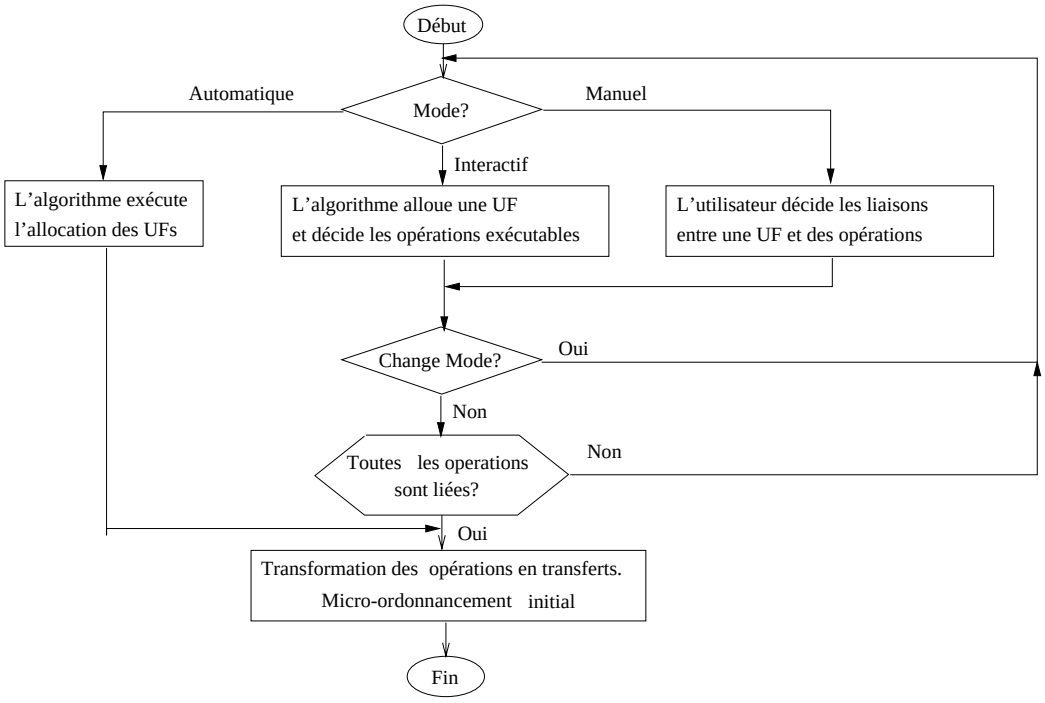


Figure 5.8: *Flot d'exécution en trois modes durant l'étape de l'allocation d'unités fonctionnelles.*

Durant l'allocation d'unités fonctionnelles en mode manuel, AMICAL assiste l'utilisateur à l'aide de plusieurs facilités graphiques.

Les transferts et les opérations déjà liées à une unité fonctionnelle disparaissent dans l'affichage de la description des macro-cycles. Cette facilité aide l'utilisateur à se concentrer sur les opérations restantes.

Pour décider manuellement les liaisons entre des opérations et une unité fonctionnelle, l'utilisateur peut commencer par sélectionner soit une unité fonctionnelle, soit des opérations. Dans le premier cas, AMICAL marque les opérations exécutables par l'unité fonctionnelle sélectionnée (les rectangles dans la figure 5.9). L'utilisateur doit sélectionner,

parmi les opérations marquées, celles qui seront liées à l'unité fonctionnelle. Les opérations sélectionnées sont alors marquées une seconde fois (en noir la figure 5.9). Dans le second cas, l'utilisateur sélectionne des opérations qu'il veut lier à une même unité fonctionnelle. Dans ce cas, AMICAL marque toutes les unités fonctionnelles qui sont capables d'exécuter les opérations sélectionnées. L'utilisateur doit sélectionner, parmi les unités fonctionnelles marquées, celle qui sera allouée et liée aux opérations sélectionnées.

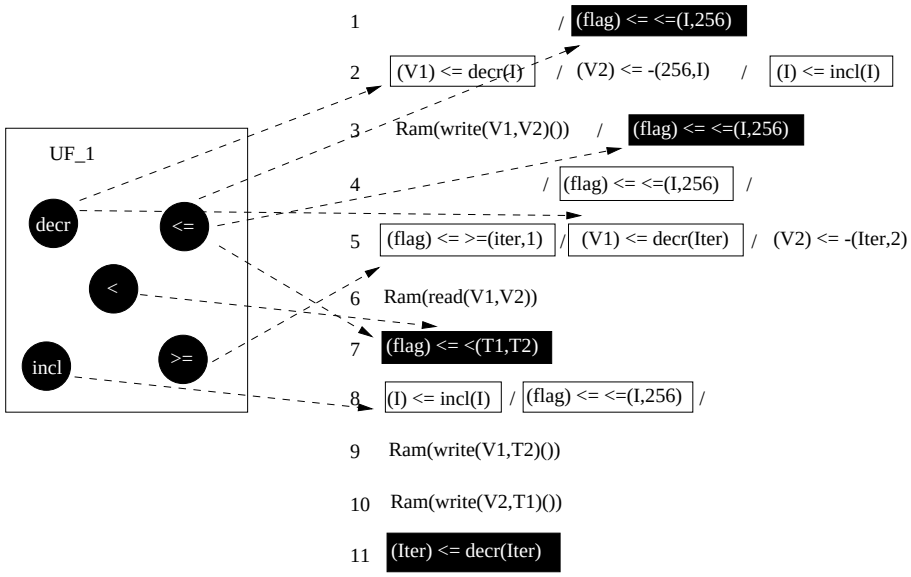


Figure 5.9: *Liaisons entre une unité fonctionnelle et des opérations en mode manuel.*

A chaque intervention de l'utilisateur, la vérification de la cohérence est nécessaire de manière à pouvoir combiner la conception automatique et la conception manuelle. Pour être acceptées, les interventions de l'utilisateur doivent respecter un certain nombre de règles de cohérence. A chaque étape du processus de synthèse, on associe un ensemble de règles permettant de vérifier la cohérence des interventions manuelles.

L'allocation d'unités fonctionnelles en mode manuel doit respecter les règles suivantes:

- Dans un macro-cycle, une unité fonctionnelle exécute une et une seule opération à la fois. Dans le cas où une unité fonctionnelle doit exécuter plusieurs opérations, celles-ci doivent être chaînées (micro-cycle différent).

- Toutes les unités fonctionnelles allouées sont disponibles pour les opérations parallèles d'un macro-cycle. Les opérations d'un macro-cycle doivent terminer leur

exécution avant que les opérations du macro-cycle suivant ne commencent à s'exécuter.

- Un appel/opération doit être lié à l'unité fonctionnelle désigné par l'utilisateur.
- Les transferts n'utilisent pas d'unités fonctionnelles.

5.4.1.2 *Allocation des connexions*

L'allocation des connexions peut être effectuée aussi en trois modes pour la solution basée sur les bus et les interrupteurs. Ces trois modes sont illustrés par la figure 5.10.

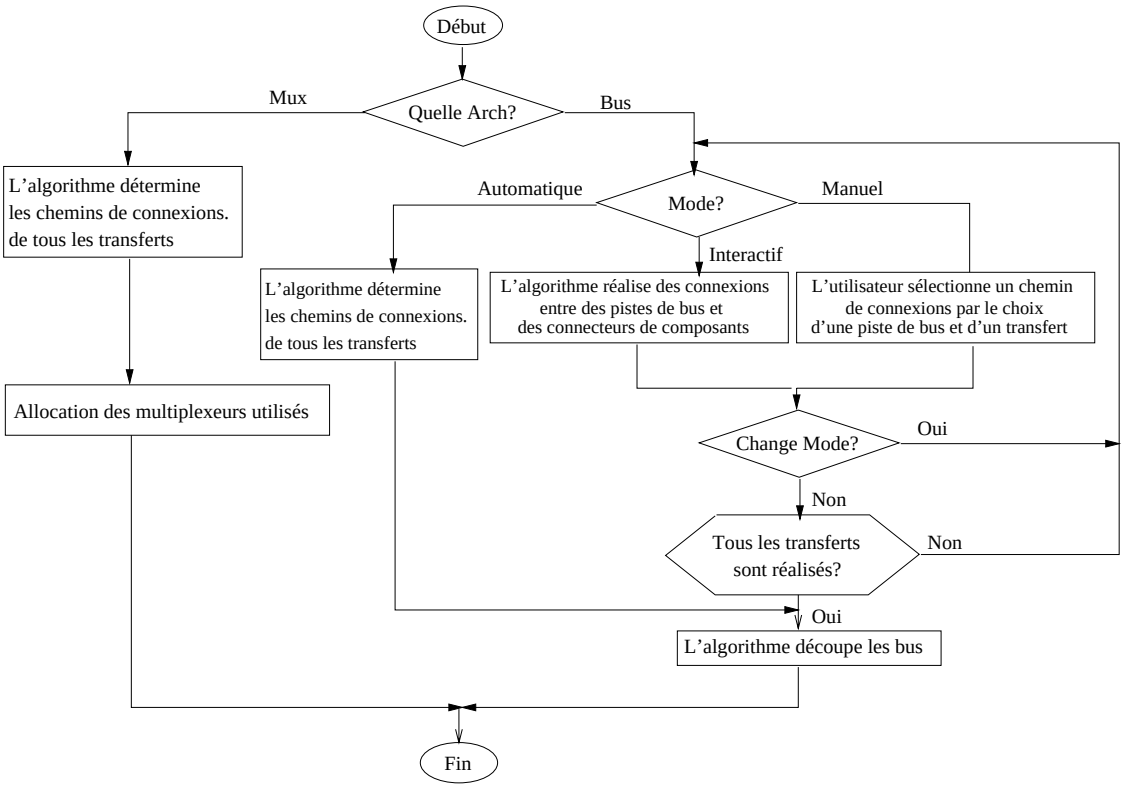


Figure 5.10: *Flot d'exécution en trois modes durant l'étape de l'allocation des connexions.*

En mode interactif, l'algorithme procède itérativement. A chaque intervalle d'itération, l'utilisateur peut intervenir, comme dans l'étape de l'allocation d'unités fonctionnelles, soit pour allouer les connexions en mode manuel, soit pour terminer l'allocation des connexions en mode automatique.

Lors de l'allocation de connexions en mode manuel, l'utilisateur doit sélectionner une piste de bus et un transfert pour définir un chemin de connexions. La sélection d'un

transfert implique que les composants de ce transfert deviennent les extrémités du chemin de connexions. AMICAL vérifie que ce chemin de connexions n'est pas utilisé par les autres transferts du même micro-cycle avant de l'accepter.

5.4.2 Evaluation à l'aide du modèle de performance

L'évaluateur analyse la structure synthétisée en générant quatre types d'informations: la surface du circuit, les délais de chaque micro-cycle, la consommation de chaque micro-cycle et diverses informations statistiques sur l'utilisations des composants. De plus, il vérifie si la structure synthétisée satisfait les contraintes demandées par l'utilisateur.

Le plan de masse et les facteurs de la surface sont utilisés pour estimer la surface totale de la partie opérative et la partie contrôle. Un exemple de résultat d'estimation de surface est montré sur la figure 5.11.

Comme mentionné dans ce chapitre, l'évaluation temporelle pour chaque micro-cycle permet de déterminer le délai maximal parmi les micro-cycle, i.e. le temps d'exécution maximal dans la partie opérative. En considérant le délai maximal dans la partie contrôle, le minima de temps pour l'horloge du système peut être déterminé. Un exemple de résultat est donné sur la figure 5.12(a).

Un exemple d'estimation de l'énergie consommée par micro-cycle pour la partie opérative est donné sur la figure 5.12(b).

Les statistiques d'utilisation des composants, dont un exemple est donné sur la figure 5.12(c), indiquent le taux d'utilisation de chaque composant. Ceci permet notamment d'optimiser la structure, en supprimant des composants peu utilisés, au cours d'une nouvelle synthèse partielle (modification du micro-ordonnancement, séquentialisation, réallocation).

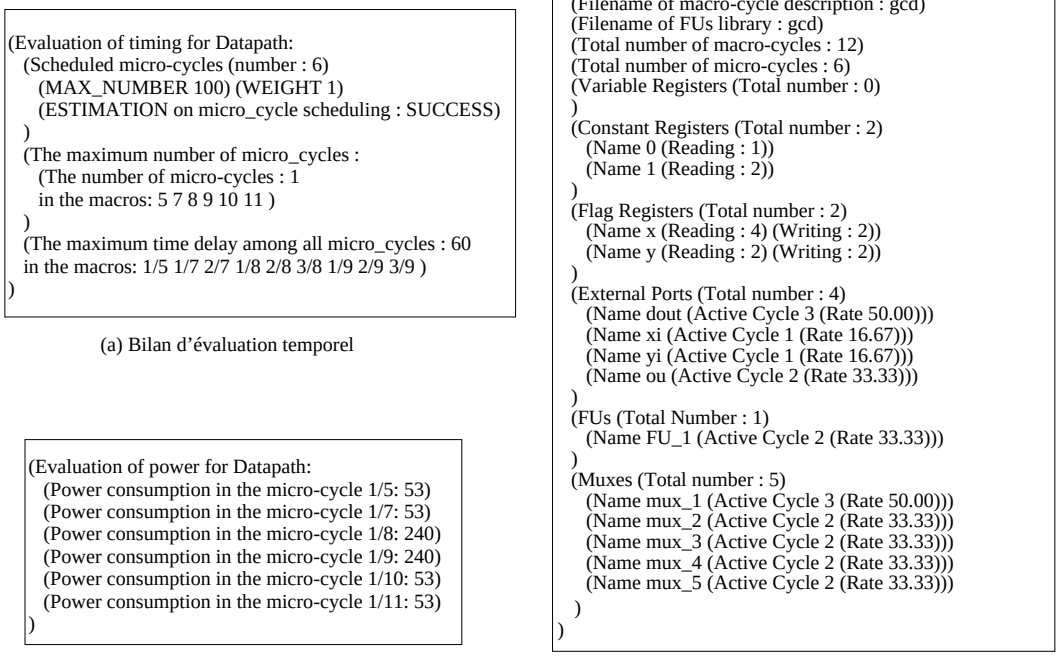
```
(Evaluation of the synthesized data path
  (Filename of macro-cycle description : gcd)
  (Filename of FUs library : gcd)
  (Allocated registers (number : 4) (NBTR : 2000)
    (Variable register :)
    (Constant register : <0> <1>)
    (Flag register : <x> <y>)
    (MAX_NUMBER illimited)
    (ESTIMATION on register allocation : SUCCESS)
  )
  (Allocated FUs (number : 1) (NBTR 1800)
    (Allocated FU <FU_1> == <SUB> (NBTR : 1800))
    (MAX_NUMBER 10) (WEIGHT 1)
    (ESTIMATION on FU allocation : SUCCESS)
  )
  (Allocated connections (NBTR 3975)
    (Allocated buses (number : 3)
      (BUS_1 : <BUS_1_1>)
      (BUS_2 : <BUS_2_1>)
      (BUS_3 : <BUS_3_1>)
      (MAX_NUMBER 9) (WEIGHT 10)
      (ESTIMATION on bus allocation : SUCCESS)
    )
    (Allocated switches (number : 15)
      (MAX_NUMBER 90) (WEIGHT 1)
      (ESTIMATION on switch allocation : SUCCESS)
    )
  )
  (Evaluation of the synthesized controller
    (Number of transistions 12)
    (Number of states 6)
    (Number of input-outputs 58 (unit bit))
    (FACTOR FCONTROLLER 0.44)
    (Controller NBTR: 500 transistors)
  )
  (Total area (NBTR : 8275 transistors)
          (The factor TR2AREA : 0.000200)
          (area : 1.66 mm2)
    (MAX_AREA 50000.00) (WEIGHT 10)
    (ESTIMATION on total area: SUCCESS)
  )
    (FINAL_ESTIMATION : SUCCESS)
)
```

(a) Solution BUS

```
(Evaluation of the synthesized data path
  (Filename of macro-cycle description : gcd)
  (Filename of FUs library : gcd)
  (Allocated registers (number : 4) (NBTR : 2000)
    (Variable register :)
    (Constant register : <0> <1>)
    (Flag register : <x> <y>)
    (MAX_NUMBER illimited)
    (ESTIMATION on register allocation : SUCCESS)
  )
  (Allocated FUs (number : 1) (NBTR 1800)
    (Allocated FU <FU_1> == <SUB> (NBTR : 1800))
    (MAX_NUMBER 10) (WEIGHT 1)
    (ESTIMATION on FU allocation : SUCCESS)
  )
  (Allocated connections (NBTR 1475)
    (Allocated muxes (number : 5)
      (MAX_NUMBER 50) (WEIGHT 1)
      (ESTIMATION on multiplexer allocation : SUCCESS)
    )
  )
)
(Evaluation of the synthesized controller
  (Number of transistions 12)
  (Number of states 6)
  (Number of input-outputs 48 (unit bit))
  (FACTOR FCONTROLLER 0.44)
  (Controller NBTR: 418 transistors)
)
(Total area (NBTR : 5693 transistors)
          (The factor TR2AREA : 0.000200)
          (area : 1.14 mm2)
  (MAX_AREA 50000.00) (WEIGHT 10)
  (ESTIMATION on total area: SUCCESS)
)
(FINAL_ESTIMATION : SUCCESS)
)
```

(b) Solution MUX

Figure 5.11: *Bilan d'évaluation de la surface.*



(a) Bilan d'évaluation temporel

(b) Bilan d'évaluation de la consommation

(c) Bilan statistique

Figure 5.12: *Bilans d'évaluation.*

5.5 Résultats Expérimentaux

5.5.1 Exemple d'Exploration Architecturale : Synthèse de Sous-Bandes de la Norme MPEG

La norme MPEG-audio est une norme audio pour la compression de signaux. Le décodeur MPEG-audio est un circuit tenant sur une puce; il peut être divisé en quatre parties majeures : le pré-analyseur, le décodeur, le contrôleur DRAM et l'interface PCM (figure 5.13). La partie décodeur est celle qui effectue la plus grande part de calcul algorithmique; elle réalise la quantisation inverse, l'échelonnage et la synthèse de sous-bandes.

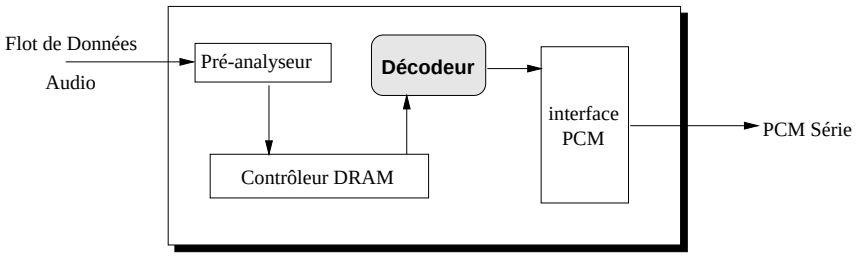


Figure 5.13: *Décodeur MPEG-audio.*

Trois partitionnements différents du module de synthèse de sous-bandes ont été réalisés. Chacune des trois descriptions comportementales a permis la génération des plusieurs solutions architecturales différentes, dont un condensé de la structure est donné (tables 5.2 et 5.3); cet exemple illustre bien l'intérêt de l'interactivité AMICAL-utilisateur différente d'un système "push button"; au fur et à mesure des versions developpées, le concepteur a pu agir sur la structure initiale ainsi que sur l'action de l'outil afin d'aboutir a un résultat satisfaisant les contraintes.

La table 5.3 donne l'évaluation qualitative des architectures engendrées. Pour chaque description, seules deux architectures sont détaillées : l'architecture résultat de la synthèse automatique et l'architecture obtenue après une réduction maximale de la surface. Les travaux relatifs à cet exemple sont détaillé dans [Kis96].

Description Comportementale	Bibliothèque d'Unités fonctionnelles	Evaluation du Résultat de Synthèse
1. Description "naïve" -Sans opérateur complexe	-Plusieurs RAMS et ROMs - 1 opération par unité fonctionnelle	Circuit de taille très grande car il n'existe aucun partage de ressources entre UFs
2. Description "réaliste" - Nombre réduit de tableaux - Transparence des constantes	- 1 RAM - 1 ROM - UFs pouvant exécuter plusieurs opérations	Circuit trop lent (indiquant la nécessité d'introduire du parallélisme lors de l'exécution)
3. Description "efficace" - Exploitation des effets de bords permis par les sous-programmes	- 1 RAM - 1 ROM - UFs pouvant exécuter plusieurs opérations	Circuit optimal (respectant les contraintes)

Table 5.2: Evolution des spécifications d'entrée et des résultats de synthèse

Architectures	1ère Version Automatique	1ère Version Optimisée	2ème Version Automatique	2ème Version Optimisée	3ème Version Automatique	3ème Version Optimisée
Nb. bus	6	4	5	4	5	3
Nb. UFs	17	14	12	6	12	6
Nb. interrup.	142	132	119	76	113	63
Nb. états	34	43	34	47	34	59
Nb. trans.	49	58	50	63	49	76

Table 5.3: Comparaison des architectures obtenues

5.5.2 Exemple industriel : L'estimateur de mouvement

L'estimateur de mouvement traité fait partie d'un vidéophone codec de la norme H261. Cet exemple industriel a été synthétisé par AMICAL et le résultat a été comparé au circuit obtenu par une équipe de concepteurs de SGS-Thomson à Crolles à partir d'une description au niveau transferts de registres[PFea95]. Les contraintes temporelles ont été satisfaites mais le résultat obtenu par la méthodologie comportementale est d'un coût additionnel en surface d'environ 7% (table 5.4) pour un gain important en termes de lignes de codes (facteur 5). Des résultats plus détaillés de cet exemple peuvent être trouvé dans [BKVaea96].

Paramètres	Nb lignes VHDL	cellules	bascules	surface(μm^2)
manuelle	668	1146	82	299640
AMICAL	136	1039	133	320650
Comparaison	- 80%	-9%	+ 62%	+7%

Table 5.4: Comparaison entre synthèse comportementale et synthèse RTL

5.6 Conclusion

La synthèse comportementale à l'aide d'outil logiciel est un progrès non négligeable dans le domaine de la conception d'ASIC, progrès dont l'industrie commence à réaliser les bienfaits avec l'apparition d'outil commerciaux et universitaires de plus en plus performants. Cependant, ce progrès reste restreint si le concepteur est absent des décisions prises au cours de la synthèse, sans avoir la possibilité d'orienter celle-ci selon des critères qui lui sont propres et qui sont propres à l'application visée.

Cette interactivité, dont les composantes sont susceptibles d'amélioration et de développement, fait partie intégrante de la philosophie initiale AMICAL.

De part la souplesse qu'elle autorise, elle facilite l'exploration architecturale qui est un des principaux intérêts de la synthèse de haut niveau : plus le niveau d'abstraction est haut, plus les décisions prises auront une influence sur la qualité du circuit final.

Ainsi, la possibilité de pouvoir influer sur les décisions au cours de la synthèse assure une qualité potentielle du résultat de la compilation de silicium, potentialité absente si l'on considère une automatisation totale.

Le modèle d'estimation, point clé de l'interactivité puisqu'il permet au concepteur d'obtenir des mesures relatives des circuits conçus au cours des diverses modifications effectuées, est en cours d'amélioration et d'extension. La paramétrisation de la bibliothèque de composants génériques, nécessaire au processus de synthèse, est le pivot autour duquel gravite toutes les estimations effectuées et qui conditionne la précision et la fiabilité des mesures fournies. Cette paramétrisation, ainsi que l'obtention de mesures statistiques permettant d'ajouter un caractère dynamique au mesures fournies (concernant vitesse et consommation notamment), représentent les principales sources d'améliorations envisagées.

CHAPITRE 6

Conclusion

Chapitre 6

Conclusion

6.1 Synthèse

L'objectif de ce travail de thèse était de contribuer à l'amélioration et à l'extention de l'outil de synthèse de haut niveau AMICAL. Les divers points exposés ici ont permis de présenter l'éventail des travaux réalisés dans cette optique, travaux qui peuvent se résumer en deux axes majeurs : synthèse multi-cible et interactivité concepteur-outil.

Proposer à un concepteur plusieurs styles architecturaux à l'issu de la synthèse comportementale est un atout important, favorisant l'exploration architecturale et donc la flexibilité de conception. Les possibilités d'architectures cibles, initialement limitées à une architecture bus, ont été étendues à une architecture à base de multiplexeurs, validée par le biais d'une expérimentation industrielle (Motion Estimator, SGS-THOMSON), et à une architecture basée sur l'emploi d'un microcontrôleur, en cours d'implantation. Ces deux extentions ont été présentées en détail au cours de cette thèse (cf. chapitre 4). Une partie de l'étude sur l'extention à une synthèse multi-cible s'est concentrée sur une

formalisation des problèmes liés à la génération de multiples architectures, sur la base d'un format de description commun à chaque style : SOLAR. De plus, chaque modèle intermédiaire de description au cours de la synthèse a fait l'objet d'une analyse détaillée; partant d'un modèle comportemental et débouchant sur plusieurs modèles au niveau RTL, le processus de synthèse passe par la création de modèles architecturaux intermédiaires correspondant à chaque étape de raffinement (cf. chapitre 3).

L'interactivité est une notion primordiale en CAO. Elle permet au concepteur de gérer le processus de synthèse et d'y participer de manière active, par opposition à une conception passive ou "push button" associée à un outil entièrement automatique. Cette notion représente le second axe d'étude au cours de cette thèse. Cette étude s'est focalisée sur deux aspects complémentaires de l'interactivité : l'interface outil-utilisateur et les mesures de performances, qui, permettant de qualifier la conception, donnent à l'utilisateur des moyens d'agir de manière efficace. L'interface d'AMICAL permet à l'utilisateur de suivre étape par étape le processus de synthèse. La possibilité lui est fournie de passer à tout instant en mode manuel pour infléchir la direction prise par le synthétiseur, qui joue alors le rôle d'assistant de conception en vérifiant la cohérence des modifications apportées. Au fur et à mesure de l'avancement du processus, le concepteur dirige et contrôle la poursuite de la synthèse en fournissant les informations adéquates. A l'issu des étapes de raffinement, la possibilité est offerte à l'utilisateur d'obtenir une première approximation des performances de l'architecture obtenue par l'intermédiaire de l'estimateur. Celui-ci fourni des informations sur le taux d'utilisation des ressources mise en jeu, ainsi que des mesures approchées concernant vitesse, surface et puissance. Cet outil de qualification de conception, susceptible d'être affiné, est indispensable pour d'une part avoir des mesures comparatives de la qualité d'une solution par rapport à une autre, et d'autre part pour être en mesure d'infléchir d'une manière efficace le processus de synthèse en explorant les diverses solutions de conception; l'exploration architecturale est en effet un atout majeur de la synthèse comportementale, atout qui ne peut être exploité que si l'on possède des éléments de comparaison lors de cette exploration (cf. chapitre 5).

6.2 Perspectives

Un certain nombre de points restent à améliorer, voire à explorer.

Tout d'abord, un certains nombre de travaux sont en cours autour du système AMICAL et concerne principalement le modèle de performance. Ce dernier est en phase de développement et d'amélioration. Les estimations de surface et de vitesse ont été implémentées mais nécessitent une amélioration au niveau de la précision. Le problème de la dépendance vis à vis de la technologie et les problèmes liés à la paramétrisation des éléments de la bibliothèque sont à l'étude. L'estimation de la consommation est en cours de développement; le modèle en cours d'implémentation tient compte de l'aspect dynamique lié à la consommation et nécessite, là aussi, une paramétrisation spécifique de la bibliothèque, ainsi qu'une prise en compte de mesures statistiques traduisant un comportement typique du circuit et issues si possible d'une simulation de la description comportementale préalable à la synthèse.

Un certain nombre d'améliorations, pouvant faire l'objet de travaux ultérieurs, sont envisageables :

- La synthèse avec AMICAL nécessite l'utilisation d'une bibliothèque dans laquelle l'utilisateur pourra trouver tout ou partie des unités fonctionnelles dont il a besoin. Dans le cas où certaines unités spécifiques au circuit visé doivent être rajoutées, ces unités doivent être instanciées, c'est à dire abstraites (boîtes noires) de manière à être lisible pour le système, et cela indépendamment de toute technologie ou implémentation finale. Cette étape, manuelle à ce jour, pourrait avantageusement être réalisée de manière automatique, du moins en partie, à partir de la description VHDL de ces unités spécifiques. Ce problème recoupe celui, largement rencontré, de la gestion des bibliothèques en CAO. L'automatisation de cette étape présenterait notamment l'avantage d'assurer une certaine cohérence dans ce processus d'instanciation, cohérence entre la fonctionnalité réelle du composant et la fonctionnalité traduite lors de son abstraction, aisément sujette aux erreurs humaines. Cette automation concerne aussi la caractérisation de chaque composant

de la bibliothèque afin de posséder des mesures réalistes, donc dépendantes de la technologie, lors du processus d'estimation des performances.

- Une architecture mixte multiplexeurs/bus représente un compromis entre les deux styles architecturaux, regroupant leurs avantages respectifs : rapidité et surface réduite. L'étude de la génération par AMICAL d'une telle architecture présente une perspective trop intéressante pour être négligée.

- Le partage des ressources, notamment au niveau des registres, n'est pas optimisé au cours de la synthèse. Par exemple, à chaque variable de la description comportementale va correspondre un registre de la description RTL. Cela présente l'avantage d'avoir une correspondance directe entre les deux descriptions, et donc une lisibilité accrue, mais le désavantage d'une majoration en terme de surface, majoration que l'étude de la durée de vie des variables et du partage de certains registres pourrait éviter.

En résumé, les travaux gravitant autour de la synthèse de haut niveau représentent un ensemble vaste et non-entièrement exploré. Cette exploration est rendu plus motivante encore par l'engouement actuel, notamment au niveau industriel, engendré par l'apparition d'outils commerciaux (Synopsys Behavioral Compiler) proposant une synthèse comportementale, et favorisé par le partenariat université-industrie dans ce domaine spécifique.

Bibliographie

Bibliographie

[AKRJ94] M. Aichouchi, P. Kission, P. Vijay Raghavan, and A.A. Jerraya. Lien entre la synthèse architecturale et la synthèse au niveau transfert de registres. *TSI (Technique et Science Informatiques)*, 1994.

[All90] J. Allen. Performance-directed synthesis of VLSI systems. *IEEE*, 78(2), February 1990.

[AP89] P. Anderson and L. Philpson. Movie - an interactive environment for silicon compilation tools. *IEEE trans. on CAD*, 6, June 1989.

[Arm88] J.R. Armstrong. Chip-level modeling with HDLs. *IEEE Design and Test of Computers*, pages 8–18, February 1988.

[BC84] G. Berry and L. Cosserat. The Esterel synchronous programming language and its mathematical semantics. Technical report, Ecole National Superieure de Mines de Paris, 1984.

[BCM+88] R.K. Brayton, R. Camposano, G. De Micheli, R.H.J.M. Otten, and J. Van Eijndhoven. *The Yorktown Silicon Compiler System*, pages 204–310. Ed. D.D. Gajski Addison-Wesley Publishing Company, 1988.

[Ber87] V. Berstis. The V compiler: Automating hardware design. *IEEE Design and Test of Computers*, pages 8–17, 1987.

[BFR85] T. Blackman, J. Fox, and C. Rosebrugh. The silc^{TM} silicon compiler: Language and features. In *Proceedings of the Design Automation Conference*, pages 232–237, Las Vegas, June 1985.

[BG87] F.D. Brewer and D.D. Gajski. Knowledge based control in micro-architecture design. In *Proceedings of the Design Automation Conference*, pages 203–209, Miami, USA., 1987.

[BKVaea96] E. Berrebi, P. Kission, S. Vernalde, and S. De Troch ans et al. Combined cntrol flow dominated and data flow dominated high-level synthesis. In *Proceedings of the Design Automation Conference*, 1996.

[BL90] J. Bhasker and H.C. Lee. An imizer for hardware synthesis. *IEEE Design and Test of Computers*, pages 20–36, 1990.

[Cam90] R. Camposano. From behavior to structure: High-level synthesis. *IEEE Design and Test of Computers*, pages 8–19, October 1990.

[Cam91] R. Camposano. Path-based scheduling for synthesis. *IEEE trans. on CAD*, 10(1):85–93, January 1991.

[Cas89] A.Z. Casavant. A synthesis environment for designing DSP systems. *IEEE Design and Test of Computers*, pages 35–43, 1989.

[CPTR89] C. Chu, M. Pontkonjak, M. Thaler, and J. Rabaey. HYPER: An interactive synthesis environment for high performance real time applications. In *Proceeding ICCD'89*, pages 432–435, Massachusetts, 1989.

[CR89] R. Camposano and W. Rosenstiel. Synthesizing circuits from behavioral descriptions. *IEEE trans. on CAD*, 8(2):171–180, February 1989.

[CST91] R. Camposano, L.F. Saunders, and R.M. Tabet. VHDL as input for high-level synthesis. *IEEE Design and Test of Computers*, pages 43–49, March 1991.

[CT90] R.J. Cloutier and D.E. Thomas. The combination of scheduling, allocation, and mapping in a single algorithm. In *Proceedings of the Design Automation Conference*, 1990.

[DCG+90] H. DeMan, F. Catthoor, G. Goossens, J. Van Meerbergen, S. Note, and J. Huisken. Architecture-driven synthesis techniques for VLSI implementation of DSP algorithms. *Proc. IEEE*, 78(2):319–355, February 1990.

[dPDL+89] M. Crates de Poulet, C. Duff, R. Leveugle, F. Poirot, G. Saucier, and P. Sicard. ASYL: A logic and architecture design automation system. In *Proceedings of EURO-ASIC*, pages 183–209, Grenoble, January 1989.

[DPST81] S.W. Director, A.C. Parker, D.P. Siewiorek, and D.E. Thomas. A design methodology and computer aids for digital VLSI systems. *IEEE trans. on Circuits System*, CAS-28(7), July 1981.

[Fis81] J.A. Fisher. Trace scheduling: A technique for global microcode compaction. *IEEE trans. on Computers*, C-30(7):478–490, July 1981.

[FY69] T.D. Friedman and S.C. Yang. Methods used in an automatic design generator (ALERT). *IEEE trans. on Computers*, C-18:593–614, 1969.

[GBK85] E.F. Girczyc, R.J.A. Buhr, and J.P. Knight. Applicability of a subset of ada as an algorithmic hardware description language for graph-based hardware compilation. *IEEE trans. on CAD*, pages 134–142, April 1985.

[GDP85] J. Granacki, D.Knapp, and A.C. Parker. The ADAM advanced design automation system: Overview, planner and natural language interface. In *Proceedings of the Design Automation Conference*, June 1985.

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

[Gui95] Philippe Guillaume. Modèle d'estimation de la consommation d'un circuit synthétisé avec AMICAL. Technical report, TIMA/INPG, 1995.

[HCLH90] C. Huang, Y. Chen, Y. Lin, and Y. Hsu. Data path allocation based on bipartite weighted matching. In *Proceedings of the Design Automation Conference*, pages 499–504, Orlando, 1990.

[HE89] B.S. Haroun and M.I. Elmasry. Architectural synthesis for DSP compilers. *IEEE trans. on CAD*, 8(4):431–447, 1989.

[HT83] C. Y. Hitchcock and D. E. Thomas. A method of automatic data path synthesis. In *Proceedings of the Design Automation Conference*, 1983.

[Inc92] Synopsys Inc. *Synopsys Design Analyzer Reference Manual, version 3.0*, December 1992.

[Inc93] Synopsys Inc. *Synopsys VHDL System Simulator Tutorial, Version 3.0b*, June 1993.

[Inc94] Synopsys Inc. *Synopsys Behavioral Compiler User Guide, Version 3.2a*, October 1994.

[ISO87] ISO/DIS9074. *International Standard, ESTELLE (Formal description technique based on an extended state transition model)*, 1987.

[ISO89] ISO, IS 8807. *LOTOS a formal description technique based on the temporal ordering of observational behavior*, February 1989.

[JBD+95] A.A. Jerraya, E. Berrebi, H. Ding, P. Kission, M. Rahmouni, and P. Vijaya Raghavan. A pragmatic apprach to behavioural synthesis. *Electronic Engineering*, May 1995.

[JD93] P. K. Jha and N. D. Dutt. Rapid estimation for parameterized components in high-level synthesis. *IEEE trans. on VLSI*, 1(3), September 1993.

[Jer89] A.A. Jerraya. *Contribution à la Compilation de Silicium et au Compilateur SYCO.* Thèse d'état, INPG, TIMA Grenoble, Décembre 1989.

[JMGC88] A.A. Jerraya, N. Mhaya, J.P. Geronimi, and B. Courtois. SYCO - a silicon compiler for VLSI ASICs specified by algorithms. *Computer-Aided Engineering Journal*, pages 122–130, 1988.

[JO92] A.A. Jerraya and K. O'Brien. SOLAR: An intermediate format for system-level design and specification. In *IFIP Inter. Workshop on Hardware/software Co-Design*, Grassau, Allemagne, May 1992.

[KDJ94] P. Kission, H. Ding, and A. A. Jerraya. Structured design methodology for high-level design. In *Proceedings of the Design Automation Conference*, San Diego, USA., June 1994.

[KDJ95] P. Kission, H. Ding, and A.A. Jerraya. VHDL based design methodology for hierarchy and component re-use at the behavioral level. In *Proceedings of the European Design Automation Conference*, 1995.

[Kis96] Polen Kission. *Exploitation de la hierarchie et de la ré-utilisation de blocs existants par la synthèse de haut niveau.* PhD thesis, INPG, 1996.

[KNRR88] H. Krämer, M. Neher, G. Rietsche, and W. Rosenstiel. Data path and control synthesis in the CADDY system. In *International Workshop at INPG*, Grenoble, 1988. INPG.

[KSJ85] D.E. Krekelberg, G.E. Sobelman, and C.S. John. Yet another silicon compiler. In *Proceedings of the Design Automation Conference*, 1985.

[LCGM93] D. Lanneer, M. Cornero, G. Goossens, and H. De Man. An assignment technique for incompletely specified data-paths. In *Proceedings of the European Conference on Design Automation*, Paris, February 1993.

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

[LMdWV91] P.E.R. Lippens, J.L. Van Meerbergen, A. Van der Werf, and W.F.J. Verhaegh. PHIDEO, a silicon compiler for high speed algorithms. In *Proceedings of the European Conference on Design Automation*, pages 436–441, Amsterdam, March 1991.

[Mar79] P. Marwedel. The MIMOLA design system: Detailed description of the software system. In *Proceedings of the Design Automation Conference*, pages 59–63, 1979.

[Mar86] P. Marwedel. A new synthesis algorithm for the MIMOLA software system. In *Proceedings of the Design Automation Conference*, 1986.

[MD94] J. Monteiro and S. Devadas. A methodology for efficient estimation of switching activity in sequential circuits. In *Proceedings of the Design Automation Conference*, 1994.

[MK88] G. De Micheli and D.C. Ku. HERCULES - a system for high-level synthesis. In *Proceedings of the Design Automation Conference*, 1988.

[MLD92] P. Michel, U. Lauther, and P. Duzy. *The Synthesis Approach to Digitial System Design.* Kluwer Academic Publishers, 1992.

[Moh94] Aichouchi Mohamed. *Etude des liens entre la synthèse architecturale et la synthèse au niveau transferet de registres.* PhD thesis, INPG, 1994.

[MPC88] M.C. McFarland, A.C. Parker, and R. Camposano. Tutorial on high-level synthesis. In *Proceedings of the Design Automation Conference*, pages 330–336, 1988.

[MPC90] M.C. McFarland, A.C. Parker, and R. Camposano. The high-level synthesis of digital systems. *IEEE*, 78(2):301–318, February 1990.

[NG92] S. Narayan and D.D. Gajski. Area and performance estimation from system-level specifications. Technical Report ICS-92-16, UC Irvine, Dept. of ICS, 1992.

[O'B93] K. O'Brien. *Compilation de silicium: du circuit au système.* PhD thesis, INPG, Grenoble, Mars 1993.

[ORA93] K. O'Brien, M. Rahmouni, and A.A.Jerraya. DLS: A scheduling algorithm for high-level synthesis in VHDL. In *Proceedings of the European Conference on Design Automation*, Paris, France, February 1993.

[Pan88] B.M. Pangrle. SPLICER: A heuristic approach to connectivity binding. In *Proceedings of the Design Automation Conference*, 1988.

[Par92] I. Park. *AMICAL: Un assistant pour la synthèse et l'exploration architecturale des circuits de commande.* Thèse inpg, INPG, Grenoble, Juillet 1992.

[Pen86] Z. Peng. Synthesis of VLSI systems with the CAMAD design aid. In *Proceedings of the Design Automation Conference*, 1986.

[PFea95] P. Paulin, J. Frehel, and M. Harrand et al. High-level synthesis and codesign methods: An application to a Videophone Codec. In *Proceedings of the European Design Automation Conference*, Paris, France, September 1995.

[PG87] B.M. Pangrle and D.D. Gajski. Slicer: A state synthesizer for intelligent silicon compilation. In *Proceedings of the International Conference on Computer-Aided Design*, pages 42–45, 1987.

[PKG86] P.G. Paulin, J.P. Knight, and E.F. Girczyc. HAL: A multi-paradigm approach to automatic data path synthesis. In *Proceedings of the Design Automation Conference*, 1986.

[RBL+96] M. Rahmouni, M. BenMohammed, C. Liem, H. Ding, P. Kission, and A.A. Jerraya. The applications of synthesis techniques for the generation of instruction-set architectures. In *Proceedings of the Design Automation Conference*, 1996. Submitted.

[RMVea88] J. Rabaey, H. De Man, J. Vanhoof, and G. Goossens et al. *Silicon Compilation*, chapter CATHEDRAL-II: A Synthesis System for Multiprocessor DSP Systems, pages 311–360. Ed. D.D. Gajski Addison-Westley Publishing Company, 1988.

[ROA94] M. Rahmouni, K. O'Brien, and A.A.Jerraya. A loop-based scheduling algorithm for hardware description languages. *Parallel Processing Letters*, 4(3):351–364, 1994.

[RP91] V.K. Rai and C.S. Patwardhan. Automated data path synthesis to avoid global interconnects. In *Proceedings. IEEE*, pages 11–16, 1991.

[SJ93] Alok Sharma and Rajiv Jain. Estimating architectural resources and pefomance for high-level synthesis applications. *IEEE trans. on VLSI*, 1(2):175–190, June 1993.

[Sou83] J.R. Southard. MacPitts: An approach to silicon compilation. *IEEE on Computer*, 16(12), December 1983.

[Std87] IEEE Std.1076-1987. *IEEE Standard VHDL Langage Reference Manual.* IEEE, NY. USA., March 1987.

[TDW+88] D.E. Thomas, E.M. Dirkes, R.A. Walker, J.V. Rajan, J.A. Nestor, and R.L. Blackburn. The system architect's workbench. In *Proceedings of the Design Automation Conference*, pages 337–343, June 1988.

[TKk89] T. Tanaka, T. Kobayashi, and O. karatsu. HARP: Fortran to silicon. *IEEE trans. on CAD*, 8(6), 1989.

[TPD94] Chi-Ying Tsui, M. Pedram, and A. Despain. Exact and approximate methods for calculating signal and transition probabilities in FSMs. In *Proceedings of the Design Automation Conference*, 1994.

[Tri87] H. Trickey. Flamel: A high-level hardware compiler. *IEEE trans. on CAD*, pages 259–269, 1987.

[TWRT88] C.J. Tseng, R.S. Wei, S.G. Rothweiler, and M.M. Tong. Bridge: A versatile behavioral synthesis system. In *Proceedings of the Design Automation Conference*, pages 415–420, 1988.

[VRB+93] Vanhoof, K. V. Rompaey, I. Bolsens, G. Goossens, and H. De Man. *High-Level Synthesis for Real-Time Digitial Signal Processing.* Kluwer Academic Publishers, 1993.

[WHHY90] Y.G. Wu, Y.H. Hu, W.P.-C. Ho, and D.Y.Y. Yun. A model-based expert system for digital system design. *IEEE Design and Test of Computers*, December 1990.

[Zim79] G. Zimmermann. The MIMOLA design system: A computer aided digital processor design method. In *Proceedings of the Design Automation Conference*, 1979.

Annexes

Annexe A

Grammaires des diverses descriptions rencontrées

A.1 Grammaire de description d'une MEF comportementale

- macro_cycle_description
  ::= (**DESCRIPTION** identifier external_bus_list variable_constant_list macro_list)
- external_bus_list
  ::= **empty** | external_bus {external_bus}
- external_bus
  ::= (**EXTERNAL_BUS** identifier bit_list dir_list)
- bit_list
  ::= (**BIT** integer integer)
- dir_list
  ::= (**DIRECTION** dir_opt)
- dir_opt
  ::= **IN** | **OUT** | **INOUT**
- variable_constant_lists
  ::= variable_constant_list {variable_constant_list}
- variable_constant_list
  ::= constant_list | variable_list
- variable_list
  ::= **empty** | variable { variable}
- variable
  ::= (**VARIABLE** identifier bit_list)
- constant_list
  ::= **empty** | constant { constant}
- constant
  ::= (**CONSTANT** identifier type_parameter bit_list)
- type_parameter
  ::= (**TYPE** type initial_value)
- type
  ::= identifier

- initial_value
  ::= integer | identifier

- macro_list
  ::= macro { macro}

- macro
  ::= (**MACRO_CYCLE** integer state_list condition_list nextstate_list operation_transfer_access_call_lists)

- state_list
  ::= (**STATE** identifier)

- nextstate_list
  ::= (**NEXTSTATE** identifier)

- condition_list
  ::= **empty** | (**CONDITION** expression)

- expression
  ::= relation | (boolean_opeator relation relation) | (boolean_opeator relation)

- relation
  ::= (relational_operator simple_expression)

- relational_operator
  ::= **=** | / = | <= | >= | < | >

- boolean_operator
  ::= & | | | NOT

- simple_expression
  ::= (identifier_list)

- identifier_list
  ::= identifiers {identifiers}

- identifiers
  ::= identifier | integer

- operation_transfer_access_lists
  ::= **empty** | operation_transfer_access_call_list {operation_transfer_access_call_list}

- operation_transfer_access_call_list
  ::= transfert_list | operation_list | access_list | call_list

- access_list
  ::= (**ACCESS** identifier access_parameter output_parameter input_parameter line_parameter)

- access_parameter
  ::= (**BOP read/write**)

- call_list
  ::= (**CALL** identifier bop_parameter output_parameter input_parameter line_parameter)

- bop_parameter
  ::= (**BOP** identifier)

- operation_list
  ::= (**OPERATION** operational_operator output_parameter input_parameter line_parameter)

- transfert_list
  ::= (**TRANSFER** output_parameter input_parameter line_parameter)

- input_parameter
  ::= **empty** | (**INPUT** identifier_list)

- output_parameter
  ::= **empty** | (**OUTPUT** identifier_list)

- line_parameter
  ::= (**LINE** integer)

- operational_operator
  ::= + | − | ∗ | / | identifier | relational_operator | boolean_operator

- identifier
  ::= **letter** {[**underline**] **letter_or_digit**}

- integer
  ::= **digit** {**digit**}

A.2 Grammaire du fichier d'abstraction d'une unité fonctionnelle

- functional_unit_description
  ::= (**FUNCTIONAL_UNIT** identifier area_power_timing_list variable_list connector_list operator_list)

- area_power_timing_list
  ::= area_parameters power_parameters timing_parameters

- area_parameters
  ::= (**AREA** integer) (**WIDTH** integer) (**HEIGHT** integer)

- power_parameters
  ::= (**POWER** integer integer)

- timing_parameters
  ::= (**MAX_DELAY** integer)

- variable_list
  ::= (**PARAMETER** input_output_parameters)

- input_output_parameters
  ::= input_parameters | output_parameters

- input_parameters
  ::= **empty** | (**INPUT** identifiers)

- output_parameters
  ::= **empty** | (**OUTPUT** identifiers)

- connector_list
  ::= **empty** | data_input_output_list | control_input_output_list

- data_input_output_list
  ::= data_input_list | data_output_list

- data_input_list
  ::= **empty** | (**INPUT** identifier bit_list) (**INPUT** identifier bit_list)

- data_output_list
  ::= **empty** | (**OUTPUT** identifier bit_list) (**OUTPUT** identifier bit_list)

- control_input_output_list
  ::= control_input_list | control_output_list

- control_input_list
  ::= **empty** | (**CONTROL_INPUT** identifier bit_list) (**CONTROL_INPUT** identifier bit_list)

- control_output_list
  ::= **empty** | (**CONTROL_OUTPUT** identifier bit_list) (**CONTROL_OUTPUT** identifier bit_list)

- bit_list
  ::= (**BIT** integer integer)

- operator_list
  ::= operator { operator}

- operator
  :: =(**OPERATOR** operator_name [commutative_list] cycle_list)

- operator_name
  ::= identifier | symbol

- symobl
  ::= ! | | # | $ | % | & | * | + | -

- commutative_list
  ::= (**COMMUTATIVE** identifiers)

- cycle_list
  :: = cycle { cycle}

- cycle
  :: =(**CYCLE** integer transfer_valid_statements)

- transfer_valid_statements
  ::= transfer_valid_statement {transfer_valid_statement }

- transfer_valid_statements
  :: = transfer_statement | valid_statement

- transfer_statement
  :: = (**TRANSFER** identifiers)

- valid_statement
  :: = (**VALID** identifier value during)

- value
  :: = (**VALUE** integer)

- during
  :: = (**DURING** integer)

- identifiers
  ::= identifier { identifier}

- identifier
  ::= **letter** {[**underline**] **letter_or_digit**}

- integer
  ::= **digit** {**digit**}

A.3 Grammaire du fichier technologique

- technology_description
  ::= (**TECHNOLOGY_FILE** parameter_list constraint_list factor_list)

- parameter_list
  ::= (**PARAMETER** constant_register flag_register variable_register external_register switch mux bus)

- constant_register
  ::= (**CONSTANT_REGISTER** (**WIDTH** integer) (**HEIGHT** integer) **AREA** integer) (**POWER** integer integer) (**MAX_DELAY** integer))

- flag_register
  ::= (**FLAG_REGISTER** (**WIDTH** integer) (**HEIGHT** integer) **AREA** integer) (**POWER** integer integer) (**MAX_DELAY** integer))

- variable_register
  ::= (**VARIABLE_REGISTER** (**WIDTH** integer) (**HEIGHT** integer) **AREA** integer) (**POWER** integer integer) (**MAX_DELAY** integer))

- external_register
  ::= (**EXTERNAL_REGISTER** (**WIDTH** integer) (**HEIGHT** integer) **AREA** integer) (**POWER** integer integer) (**MAX_DELAY** integer))

- switch
  ::= (**SWITCH** (**WIDTH** integer) (**HEIGHT** integer) **AREA** integer) (**POWER** integer integer) (**MAX_DELAY** integer))

- mux
  ::= (**MUX** (**WIDTH** integer) (**HEIGHT** integer) **AREA** integer) (**POWER** integer integer) (**MAX_DELAY** integer))

- bus
  ::= (**BUS** (**HEIGHT** integer) (**POWER** integer integer) (**MAX_DELAY** integer))

- constraint_list
  ::= (**CONSTRAINT** max_mux max_micro max_bus max_fu max_switch max_width max_height max_area max_power)

- max_mux
  ::= (**MAX_MUX** integer (**WEIGHT** integer))

- max_micro
  ::= (**MAX_MICRO** integer (**WEIGHT** integer))

- max_bus
  ::= (**MAX_BUS_CHANNEL** integer (**WEIGHT** integer))

- max_fu
  ::= (**MAX_FU** integer (**WEIGHT** integer))

- max_switch
  ::= (**MAX_SWITCH** integer (**WEIGHT** integer))

- max_width
  ::= (**MAX_WIDTH** float (**WEIGHT** integer))

- max_height
  ::= (**MAX_HEIGHT** float (**WEIGHT** integer))

- max_area
  ::= (**MAX_AREA** integer (**WEIGHT** integer))

- max_power
  ::= (**MAX_POWER** integer (**WEIGHT** integer))

- factor_list
  ::= (**FACTOR** (**FDATAPATH** float) (**FCONTROLLER** float) (**FCIRCUIT** float) (**TR2AREA** float) (**SWPOWER** float))

- integer
  ::= **digit** {**digit**}

- float ::= **digit** {**digit**} . {**digit**}

Annexe B

Définitions détaillée des différents modèles architecturaux

---

1. **Modèle comportemental(avant compilation)**
   - Description: langage VHDL
   - Organisation: un seul process VHDL
   - Unité de temps: étape de calcul
   - Objets:
     - I/O
     - signaux, variables
     - procedures/fonctions
     - opérations sur les données
     - opérations de contrôle
   - Objets à générer lors de la prochaine étape:
     - arcs
     - nœuds
   - Modèle d'exécution:
     - synchronisation avec **wait**
     - exécution séquentielle
   - Obtention: la description
2. **Modèle de graphe de contrôle(CFG)(après compilation)**
   - Description: CFG
   - Organisation: graphe connecté (GC)
   - Unité de temps: état
   - Objets:
     - nœuds
       - opérations (op.)
       - conditions (cond.)
     - arcs
     - I/O
     - variables
   - Objets à générer lors de la prochaine étape:

  - états
  - registre (reg.)
  - transitions (trans.)
- Modèle d'exécution:
  - synchronisation avec les nœuds **wait**
  - exécution séquentielle
- Obtention: la compilation

3. **Modèle de MEF comportementale (après l'ordonnancement)**

- Description: MEF (Machine d'Etats Finis)
- Organisation: MEF
- Unité de temps: étape de contrôle (EC)
- Objets:
  - états
  - I/O
  - transitions
  - opérations
  - registre (reg.)
- Objets à générer lors de la prochaine étape:
  - unités fonctionnelles (UF) (à allouer)
  - micro-cycle
  - micro-états
- Modèle d'exécution:
  - synchronisation avec les signaux
  - un transition par chaque étape de contrôle
  - pipeline
- Obtention: l'ordonnancement

4. **Modèle de MEF comportementale reliée (avec ressources) (après allocation des UFs)**

- Description: MEF (Machine d'Etats Finis)
- Organisation: MEF
- Unité de temps: étape de contrôle (EC)
- Objets:
  - états
  - I/O
  - transitions
  - opérations (Unités Fonctionnelles)
  - registre (reg.)
- Objets à générer lors de la prochaine étape:
  - micro-cycle
  - micro-états
- Modèle d'exécution:
  - synchronisation avec les signaux
  - un transition par chaque étape de contrôle
  - pipeline
- Obtention: allocation des UFs

5. **Modèle de MEF au niveau transfert de registres (après micro-ordonnancement)**

- Description: MEF
- Organisation: MEF
- Unité de temps: micro-cycle
- Objets:
  - registres
  - I/O
  - micro-états
  - micro-transitions

  - transferts élémentaires
- Objets à générer lors de la prochaine étape:
  - unités de connexions(switch, bus, et mux) (UC) (à allouer)
- Modèle d'exécution:
  - synchronisation via le micro-cycle
  - MEF
- Obtention: micro-ordonnancement

6. **Le modèle d'architecture abstraite(après allocation des connexions)**

   - Description: SOLAR
   - Organisation: MEF + Partie Opérative
   - Unité de temps: micro-cycle
   - Objets:
     - I/O
     - registres
     - UFs
     - micro-états
     - micro-transitions
     - unités de connexions(switch, bus, et mux)
     - transferts élémentaires
   - Objets à générer lors de la prochaine étape:
     - bloc concurrents (bloc de synchronisation)
   - Modèle d'exécution:
     - synchronisation via le micro-cycle
     - MEF
   - Obtention: allocation des connexions

7. **Modèle PO-PC(après Génération d'Architecture et Personnalisation)**

   - Description: VHDL
   - Organisation: PC-PO synchronisées
   - Unité de temps: période d'horloge
   - Objets:
     - contrôleur
     - chemin de données
   - Modèle d'exécution:
     - synchronisation via l'horloge
     - MEF
   - Obtention: la génération de l'architecture

Annexe C

Les algorithmes de génération d'architectures

C.1 Génération de l'architecture à base des multiplexeurs

Il est nécessaires de considérer d'une part la génération de contrôleur, et d'autre part, la génération de la partie opérative.

C.1.1 Génération du contrôleur

La génération du contrôleur va s'effectuer en deux étapes, ainsi qu'il a été mentionné plus haut : interface et table d'états.

L'algorithme de la génération de l'interface est détaillé sur la figure C.1.

La génération des signaux de sélection des composants fonctionnels est basée sur un micro-cycle.

Un diagramme d'états contient un ensemble de micro-cycles dont seulement un est sélectionné en fonction du résultat de l'évaluation des conditions à un moment donné. Un

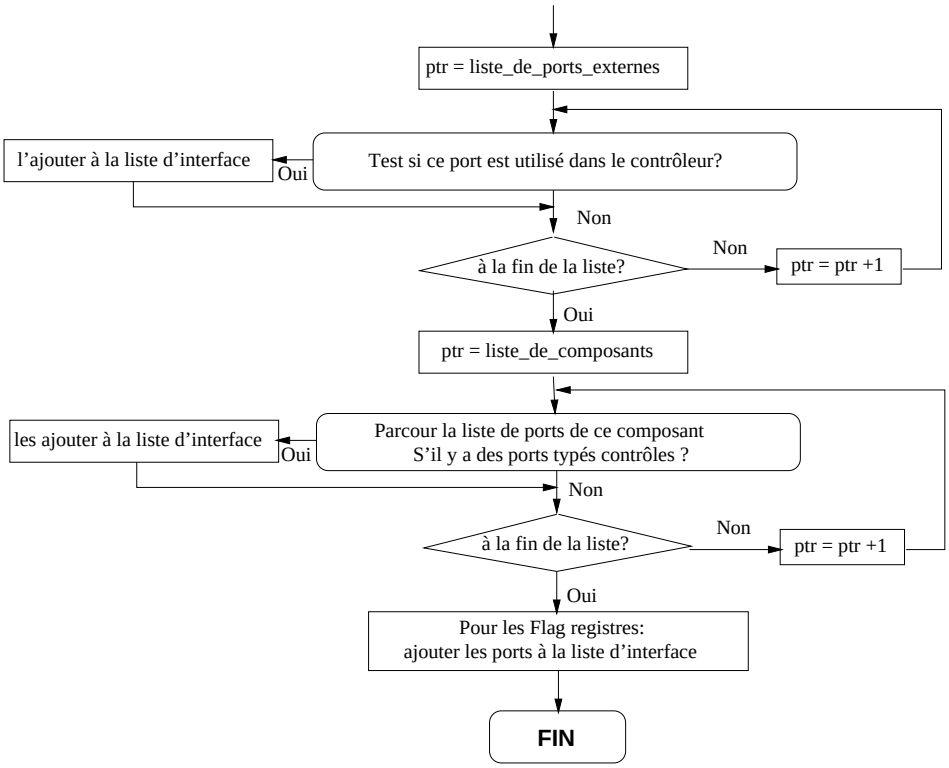


Figure C.1: *Algorithme de génération de l'interface du contrôleur.*

micro-cycle choisi, il est évident qu'au moins un sous-ensemble de composants doit être actif pour accomplir son action.

L'algorithme de cette étape est représenté dans la figure C.2.

Le principe qui conditionne la génération des signaux de contrôle des chemins de connexions est de trouver toutes les unités de connexions nécessaires et de générer les signaux de sélections.

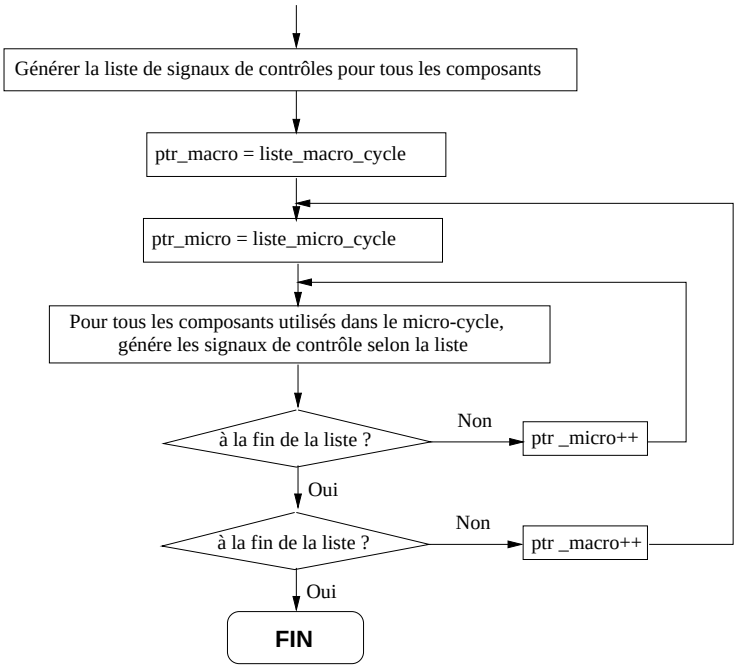


Figure C.2: *Algorithme de génération de signaux de contrôle pour les composants.*

C.1.2 Génération de la partie opérative

Pour générer la partie opérative, trois types d'informations sont nécessaires. Le premier type d'information porte sur la déclaration de l'interface, le second sur celle des composants utilisés, et le dernier sur celle des connexions entre les composants.

Selon l'architecture demandée, divers algorithmes sont utilisés dont la différence essentielle réside dans la génération du réseau de communication basé sur différents composants de connexion. En ce qui concerne l'algorithme de génération de la liste de connexions, deux différents algorithmes sont proposés en fonction de l'architecture choisie.

La méthodologie générale est résumée sur la figure C.3.

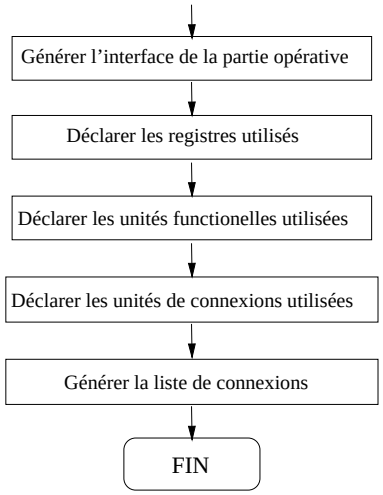


Figure C.3: *Algorithme de génération de la partie opérative*

La génération de la partie opérative comporte aussi plusieurs étapes. Plusieurs points vont être pris en compte :

- **l'interface de la partie opérative** est la partie connectée avec l'extérieur;
- **les composants fonctionnels** sont les unités fonctionnelles, les registres, les connecteurs externes, etc.;
- **le nombre de multiplexeurs** est déterminé par le nombre total des destinations des transferts;

- **le nombre d'entrées** d'un multiplexeur est déterminé par le nombre des sources connectées;

- **les connexions entre des composants** sont les signaux qui permettent de connecter entre les ports des composants.

La figure C.4 montre l'algorithme de la génération des multiplexeurs basé sur une méthode constructive.

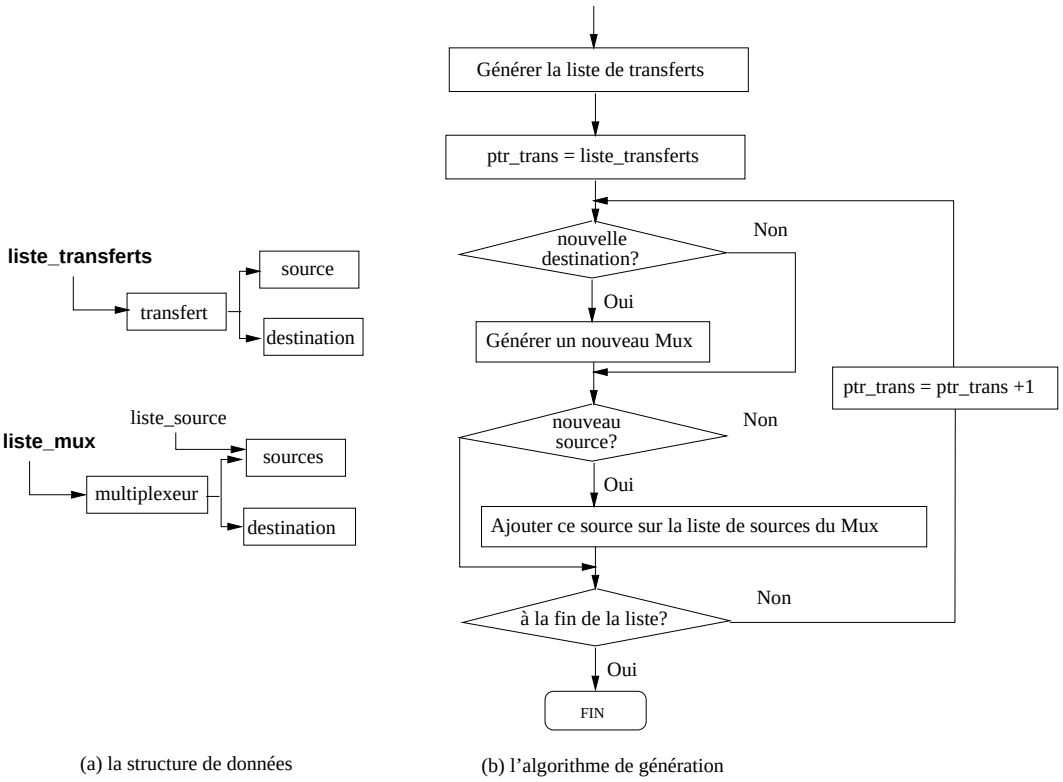


Figure C.4: *Algorithme de génération des multiplexeurs*

Une fois que la liste de multiplexeurs a été générée, les connexions du circuit sont fixées. Parcourant la liste de transferts, des connexions seront ajoutées de manière triviale entre source et multiplexeurs, et entre multiplexeur et destination à chaque fois qu'un multiplexeurs sera lu Il faudra aussi générer des connexions entre les signaux globaux et les ports internes. La figure C.5 illustre les étapes de la génération des connexions.

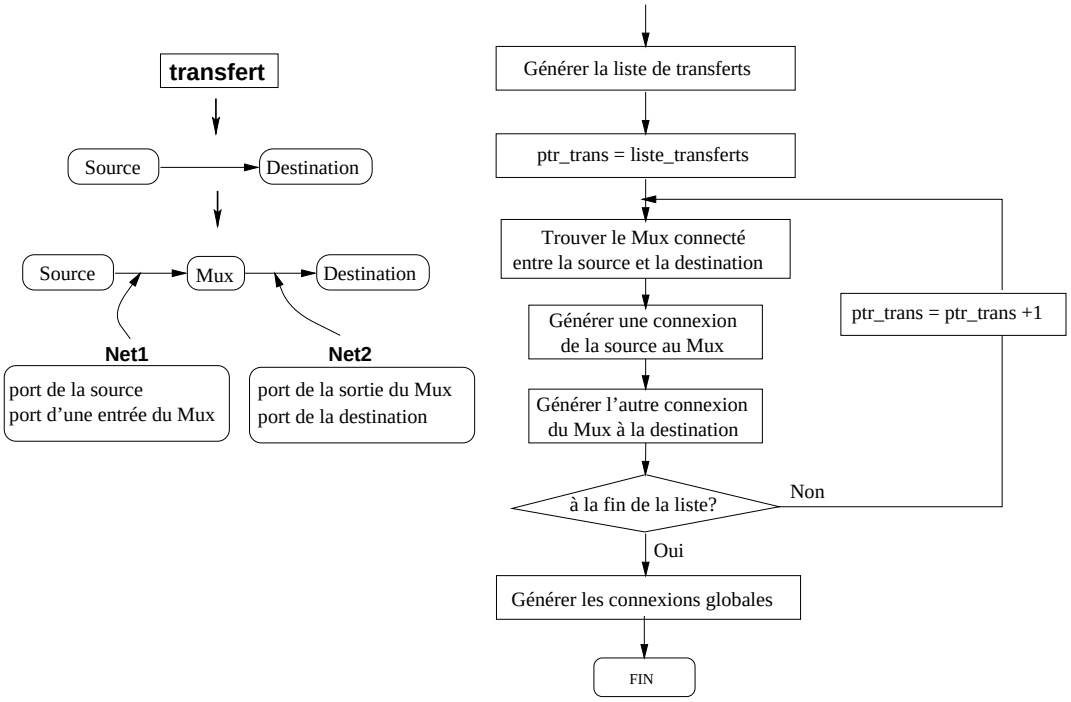


Figure C.5: *Algorithme de génération des connexions de la partie opérative*

C.2 Génération de l'architecture à base d'un micro-contrôleur

Le processus de génération du contrôleur est illustré sur la figure C.6.

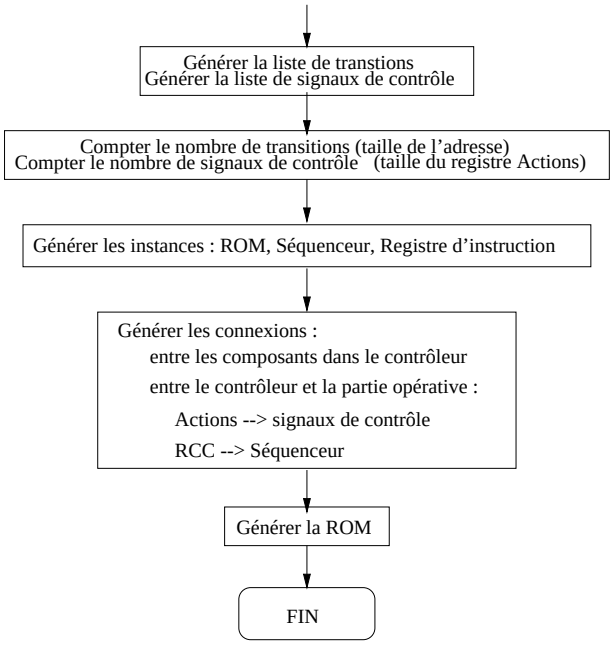


Figure C.6: *Le flot de la génération du contrôleur*

L'algorithme de génération de la ROM est présenté sur la figure C.7.

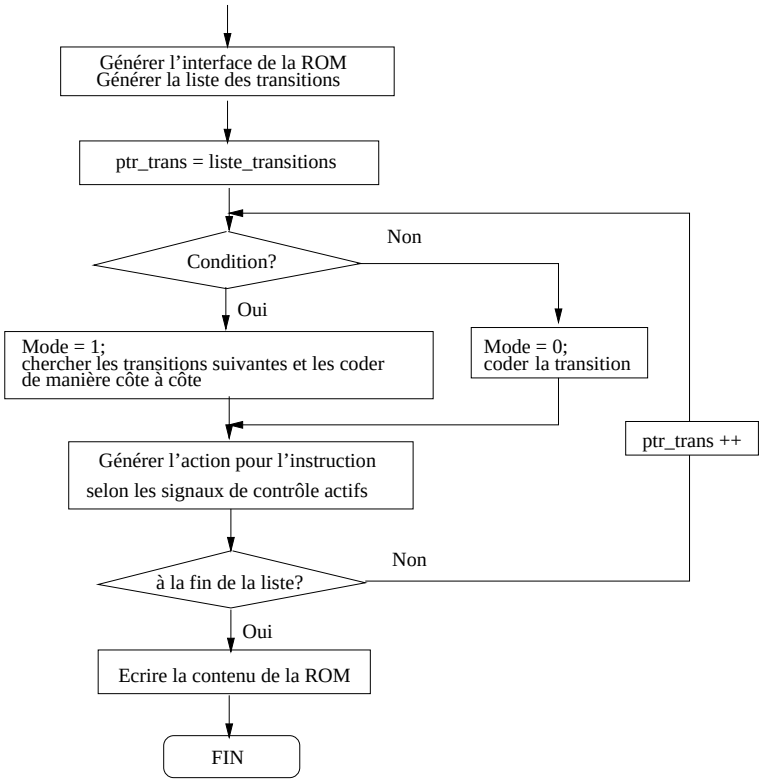


Figure C.7: *L'algorithme de génération de la ROM*

Annexe D

Définition du menu de commandes

Le menu de commandes d'AMICAL consiste en quatre niveaux hiérarchiques principaux, chacuns d'eux s'organisant en plusieurs parties et sous parties, ainsi qu'il est représenté sur la figure D.1.

Le premier niveau constitue le "front-end" d'AMICAL, i.e., la compilation d'une description en VHDL et l'ordonnancement (cf. figure D.1 (1)) . AMICAL permet d'utiliser plusieurs compilateurs de VHDL comme celui de CLSI (COMPASS), LVS (LEDA) ou Leap-frog (CADENCE). L'ordonnancement se fait à l'aide d'un programme indépendant, appelé "schedule", s'exécutant lors de la sélection de la commande correspondante.

Le second niveau représente les étapes principales du processus de synthèse (cf. figure D.1 (2)). Il contient plusieurs fonctionnalités. La figure D.2 montre l'organisation du menu et les tâches correspondantes.

Le troisième niveau (cf. figure D.1 (3)) correspond à diverses opérations : l'évaluation, l'information, le changement des paramètres, etc. (voir la figure D.3).

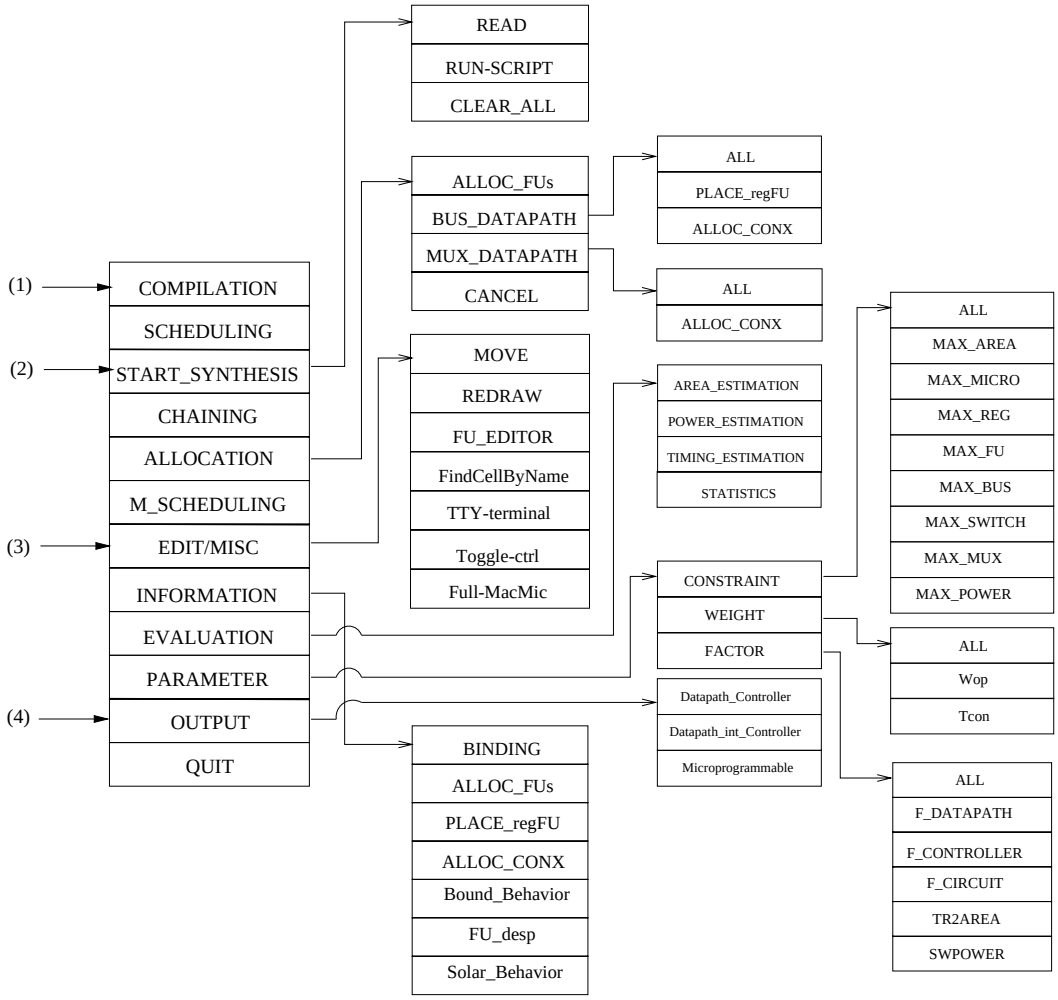


Figure D.1: *Les différents menus disponibles.*

Le dernier niveau donne les commandes permettant de générer les architectures (cf. figure D.1 (4)). Plusieurs architectures peuvent être générés par le système AMICAL. Le système vérifie automatiquement la cohérence de la génération. Par exemple, si la solution micro-programmable a été choisie, le système génère automatiquement la description de ROM et la description du contrôleur spécifique. (voir la figure D.4).

Menu principal	Menu secondaire	Menu tertiaire	Fonction
START_SYNTHESIS	READ		Lire les fichiers nécesaires à l'allocation
	RUN-SCRIPT		Lancer un script
	CLEAR_ALL		Effacer tout pour démarrer un autre exemple
CHAINING			Chaîner les opérations
ALLOCATION	ALLOC_FUs		Exécuter l'allocation d'UFs
	BUS_DATAPATH	ALL	Enchaîner toutes les étapes d'allocation (BUS)
		PLACE_regFU	Placer les composants
		ALLOC_CONX	Exécuter l'allocation des connexions
	MUX_DATAPATH	ALL	Enchaîner toutes les étapes d'allocation (MUX)
		ALLOC_CONX	Exécuter l'allocation des connexions
	CANCEL		Annuler la dernière étape de la synthèse
M_SCHEDULING			Modifier le micro-ordonnancement

Figure D.2: *Le menu des étapes d'allocation*

Menu principal	Menu secondaire	Menu tertiaire	Fonction
EDIT/MISC	MOVE		Déplacer la position d'un composant
	REDRAW		Rafraîchir l'écran
	FU_EDITOR		Ouvrir un texte editeur pour l'UF
	FindCellByName		Trouver une cellule par son nom
	TTY-terminal		Ouvrir une fenêtre UNIX
	Toggle-ctrl		Montrer les conditions/Cacher les conditions
	Full-MacMic		Montrer la description des micro-cycles Cacher la description des micro-cycles
INFORMATION	BINDING		Montrer les liaisons entre une opération et un composant, etc
	ALLOC_FUs		Documenter les états des opérations
	PLACE_regFU		Documenter les propositions de placement
	ALLOC_CONX		Documenter les chemins de connexions
	Bound_Behavior		Documenter les transferts liés avec un composant
	FU_desp		Montrer la description d'une UF
	Solar_Behavior		Montrer la description au niveau RT
EVALUATION	AREA_ESTIMATION		Evaluer la surface de l'architecture
	POWER_ESTIMATION		Evaluer la consommation de l'architecture
	TIMING_ESTIMATION		Evalue le timing de l'architecture
	STATISTICS		Statistique d'utilisation de composants

Figure D.3: *Le menu d'informations*

Menu principal	Menu secondaire	Fonction
OUTPUT	Datapath_Controller	Génération de l'architecture avec la partie opérative et la partie contrôle (Pour la solution Bus et la solution Mux)
	Datapath_int_Controller	Génération de l'architecture avec la partie opérative, la partie contrôle et l'interface entre ces deux parties (Pour la solution Bus et la solution Mux)
	Microprogrammable	Génération de l'architecture micro-programmable

Figure D.4: *Le menu de génération*

Résumé

Cette thèse présente plusieurs travaux visant à l'amélioration de la synthèse architecturale réalisée à l'aide de l'outil de synthèse de haut niveau AMICAL. Un point clé de ce travail est la notion d'interactivité. Le processus de synthèse se décompose en un ensemble de raffinements successifs. L'utilisateur a la possibilité d'intervenir au cours de ces différentes étapes et d'agir manuellement, ou au contraire de laisser se dérouler seules l'ensemble des étapes tout en gardant une vision claire des actions effectuées. Ce dernier a de plus le choix entre plusieurs styles architecturaux qu'il pourra implémenter à son gré, ce qui autorise une grande flexibilité.

Les points principaux abordés au cours de cette thèse sont les suivants :

- Les étapes et modèles successifs de raffinement au cours du processus de synthèse : chaque sous-tâche engendre un modèle architectural intermédiaire à partir duquel la sous-tâche suivante pourra agir.
- La notion d'interactivité : celle-ci inclue la mise au point d'un modèle de performance permettant d'estimer la qualité du circuit synthétisé, et permet au concepteur d'être le véritable acteur de la synthèse tout en l'assistant lors de la prise de décisions.
- La génération de plusieurs types d'architectures et les problèmes algorithmiques qui y sont liés.

Mots-Clefs: Synthèse architecturale, Synthèse interactive, Modèle architectural, Modèle de performance, Génération d'architectures.

Abstract

This thesis presents an interactive High Level Synthesis environment called AMICAL. The synthesis process is decomposed into a set of refinement steps. The user can execute these steps automatically, manually or in interactive mode when needed. The synthesis scheme is flexible; it allows several architectural models for the generated data-path (bus model, multiplexer model) and controller (hardwired, programmable).

The main issues developed in this thesis are:

- The models and steps used for refinements in a synthesis process. Several architectural models are defined for bridging gap between two synthesis steps.
- The interactive synthesis model. It includes a performance model allowing to estimate the synthesized results, and allows the designer to be a real actor of the synthesis process.
- The generation of different architectures and their algorithm issues. These architectures are usable as inputs for lower synthesis tools.

Keywords: Architectural synthesis, Architectural models, Algorithms, Functional unit allocation, Control-flow dominated circuits, Performance model, Interactive synthesis, Architecture generation.