Synthèse architecturale de circuits intégrés

Anne Mignotte

Thèse

Présentée par

Anne MIGNOTTE

Pour obtenir le grade de

Docteur de l'Institut National Polytechnique de Grenoble
(arrêté ministériel du 30 Mars 1992)

(Spécialité **Microélectronique**)

Synthèse architecturale de circuits intégrés

Soutenue le 26 Novembre 1992
Composition du Jury:

Prof. Jacques Mossière	Président
Dr. Raul Camposano	Rapporteur
Dr. Roland Gerber	Rapporteur
Prof. Guy Mazaré	
Prof. Gerde Finke	
Prof Gabrièle Saucier	

Thèse préparée au laboratoire de Conception de Sytèmes Intégrés à l'INPG.

PRESIDENT DE L'INSTITUT
Monsieur Maurice RENAUD

Année 1991/1992

PROFESSEURS DES UNIVERSITES

BARIBAUD	Michel	ENSERG
BARRAUD	Alain	ENSIEG
BAUDELET	Bernard	ENSPG
BAUDIN	Gérard	UFR PGP
BEAUFILS	Jean-Pierre	ENSIEG/ILL
BOIS	Philippe	ENSHMG
BONNET	Guy	ENSPG
BOUVIER	Gérard	ENSERG
BRISSONNEAU	Pierre	ENSIEG
BRUNET	Yves	CUEFA
CAILLERIE	Denis	ENSHMG
CAVAIGNAC	Jean-François	ENSPG
CHARTIER	Germain	ENSPG
CHENEVIER	Pierre	ENSERG
CHERADAME	Hervé	UFR/PGP
CHERUY	Arlette	ENSIEG
CHOYET	Alain	ENSERG
COGNET	Gérard	ENSHMG
COLINET	Catherine	ENSEEG
COMMAULT	Christian	ENSIEG
CORNUT	Bruno	ENSIEG
COULOMB	Jean-Louis	ENSIEG
CROWLEY	James	ENSIMAG
DALARD	Francis	ENSEEG
DARVE	Félix	ENSHMG
DELLA DORA	Jean	ENSIMAG
DEPEY	Maurice	ENSERG
DEPORTES	Jacques	ENSPG
DEROO	Daniel	ENSEEG
DESRE	Pierre	ENSEEG
DIARD	Jean-Paul	ENSEEG
DOLMAZON	Jean-Marc	ENSERG
DURAND	Francis	ENSEEG
DURAND	Jean-Louis	ENSPG
FAUTRELLE	Yves	ENSHMG
FOGGIA	Albert	ENSIEG
FONLUPT	Jean	ENSIMAG
FOULARD	Claude	ENSIEG
GALERIE	Alain	ENSEEG
GANDINI	Alessandro	UFR/PGP
GAUBERT	Claude	ENSPG
GENTIL	Pierre	ENSERG
GENTIL	Sylviane	ENSIEG
GREVEN	Hélène	CUEFA
GUERIN	Bernard	ENSERG
GUYOT	Pierre	ENSEEG
IVANES	Marcel	ENSIEG
JALLUT	Christian	ENSEEG

JANOT	Marie-Thérese	ENSERG
JAUSSAUD	Pierre	ENSIEG
JOST	Rémy	ENSPG
JOUBERT	Jean-Claude	ENSPG
JOURDAIN	Geneviéve	ENSIEG
KUENY	Jean-Louis	ENSHMG
LACHENAL	Dominique	UFR PGP
LACOUME	Jean-Louis	ENSIEG
LADET	Pierre	ENSIEG
LESIEUR	Marcel	ENSHMG
LESPINARD	Georges	ENSHMG
LIENARD	Joël	ENSIEG
LONGEQUEUE	Jean-Pierre	ENSPG
LORET	Benjamin	ENSHMG
LOUCHET	François	ENSEEG
LUCAZEAU	Guy	ENSEEG
LUX	Augustin	ENSIMAG
MASSE	Philippe	ENSIEG
MASSELOT	Christian	ENSIEG
MAZARE	Guy	ENSIMAG
MICHEL	Gérard	ENSIMAG
MOHR	Roger	ENSIMAG
MOREAU	René	ENSHMG
MORET	Roger	ENSIEG
MOSSIERE	Jacques	ENSIMAG
OBLED	Charles	ENSHMG
OZIL	Patrick	ENSEEG
PANANAKAKIS	Georges	ENSERG
PAULEAU	Yves	ENSEEG
PERRET	Robert	ENSIEG
PIAU	Jean-Michel	ENSHMG
PIC	Etienne	ENSERG
PLATEAU	Brigitte	ENSIMAG
POUPOT	Christian	ENSERG
RAMEAU	Jean-Jacques	ENSEEG
REINISCH	Raymond	ENSPG
RENAUD	Maurice	UFR/PGP
ROBERT	François	ENSIMAG
ROSSIGNOL	Michel	ENSPG
ROYE	Daniel	ENSIEG
SABONNADIERE	Jean-Claude	ENSIEG
SAGUET	Pierre	ENSERG
SAUCIER	Gabriéle	ENSIMAG
SCHLENKER	Claire	ENSPG
SCHLENKER	Michel	ENSPG
SILVY	Jacques	UFR/PGP
SIRIEYS	Pierre	ENSHMG
SOHM	Jean-Claude	ENSEEG
SOLER	Jean-Louis	ENSIMAG
SOUQUET	Jean-Louis	ENSEEG
TICHKIEWITCH	Serge	ENSHMG
TROMPETTE	Philippe	ENSHMG
TRYSTRAM	Denis	ENSGI
VEILLON	Gérard	ENSIMAG
VERJUS	Jean-Pierre	ENSIMAG
VINCENT	Henri	ENSPG
ZADWORNY	François	ENSERG

SITUATION PARTICULIERE

PROFESSEURS D'UNIVERSITE

DETACHEMENT

ENSERG	BLIMAN	Samuel	Mutation	
ENSPG	BLOCH	Daniel	Recteur	21/12/93
ENSIMAG	LATOMBE	Jean-Claude	Détachement	01/05/93
ENSHMG	PIERRARD	Jean-Marie	Disponible	

RETRAITE

ENSEEG	BONNETAIN	Lucien

DIRECTEURS DE RECHERCHE CNRS

ABELLO	Louis
ALDEBERT	Pierre
ALEMANY	Antoine
ALLIBERT	Colette
ALLIBERT	Michel
ANSARA	Ibrahim
ARMAND	Michel
AUDIER	Marc
AUGOYARD	Jean-François
AVIGNON	Michel
BERNARD	Claude
BINDER	Gilbert
BLAISING	Jean-Jacques
BONNET	Roland
BORNARD	Guy
BOUCHERLE	Jean-Xavier
CAILLET	Marcel
CARRE	René
CHASSERY	Jean-Marc
CHATILLON	Christian
CIBERT	Joël
CLERMONT	Jean-Robert
COURTOIS	Bernard
CRIQUI	Patrick
CRISTOLOVEANU	Sorin
DAVID	René
DION	Jean-Michel
DOUSSIERE	Jacques
DRIOLE	Jean
DUCHET	Pierre
DUGARD	Luc
DURAND	Robert
ESCUDIER	Pierre
EUSTATHOPOULOS	Nicolas
FINON	Dominique
FRUCHARD	Robert
GARNIER	Marcel
GIROD	Jacques
GLANGEAUD	François
GUELIN	Pierre
HOPFINGER	Emil
JORRAND	Philippe
JOUD	Jean-Charles
KAMARINOS	Georges
KLEITZ	Michel
KOFMAN	Walter
LACROIX	Claudine
LANDAU	Ioan
LAULHERE	Jean-Pierre
LEGRAND	Michel
LEJEUNE	Gérard
LEPROVOST	Christian
MADAR	Roland
MARTIN	Jean-Marie
MERMET	Jean
MEUNIER	Gérard
MICHEL	Jean-Marie
NAYROLLES	Bernard
PASTUREL	Alain
PEUZIN	Jean-Claude
PHAM	Antoine
PIAU	Monique
PIQUE	Jean-Paul
POINSIGNON	Christiane
PREJEAN	Jean-Jacques
RENOUARD	Dominique
SENATEUR	Jean-Pierre
SIFAKIS	Joseph
SIMON	Jean-Paul
SUERY	Michel
TEODOSIU	Christian
VACHAUD	Georges
VAUCLIN	Michel
WACK	Bernard
YAYARI	Ali-Reza
YONNET	Jean-Paul

PERSONNES AYANT OBTENU LE DIPLOME

D'HABILITATION A DIRIGER DES RECHERCHES

BALESTRA	Francis
BALME	Louis
BECKER	Monique
BIGEON	Jean
BINDER	Zdeneck
BOE	Louis-Jean
CANUDAS DE WIT	Carlos
CHOLLET	Jean-Pierre
COEY	Jean-Pierre
CORNUEJOLS	Gerard
COURNIL	Michel
CRASTES DE PAULET	Michel
DALLERY	Yves
DESCOTES-GENON	Bernard
DUGARD	Luc
DURAND	Madeleine
FERRIEUX	Jean-Paul
FEUILLET	René
FREIN	Yannick
GAUTHIER	Jean-Paul
GHIBAUDO	Gérard
GUILLEMOT	Nadine
GUYOT	Alain
HAMAR	Sylviane
HAMAR	Roger
HORAUD	Patrice
JACQUET	Paul
LATOMBE	Claudine
LE HUY	Hoang
LE GORREC	Bernard
LOZANO-LEAL	Rogelio
MACOYSCHI	Mihaïl
MAHEY	Philippe
METAIS	Olivier
MONMUSSON-PICQ	Georgette
MORY	Mathieu
MULLER	Jean
MULLER	Jean-Michel
NGUYEN TRONG	Bernadette
NIEZ	Jean-Jacques
PERRIER	Pascal
PLA	Fernand
RECHENMANN	François
ROGNON	Jean-Pierre
ROUGER	Jean
ROUX	Jean-Claude
TCHUENT	Maurice

PERSONNES AYANT OBTENU LE DIPLOME

DE DOCTEUR D'ETAT INPG

ABDEL-RAZEK	Adel
AKSAS	Haris
ALLA	Hassane
AMER	Ahmed
ANCELLE	Bernard
ANGENIEUX	Gilbert
ATMANI	Hamid
AYEDI	Hassine Feri
A.BADR	Osman
BACHIR	Aziz
BALANZAT	Emmanuel
BALTER	Roland
BARDEL	Robert
BARRAL	Gérard
BAUDON	Yves
BAUSSAND	Patrick
BEAUX	Jacques
BEGUINOT	Jean
BELLISSENT née FUNEZ	Marie-Claire
BELLON	Catherine
BEN RAIS	Abdejettah
BERGER-SABBATEL	Gilles
BERNACHE-ASSOLANT	Didier
BEROVAL	Abderrahmane
BERTHOD	Jacques
BILLARD	Dominique
BLANC épouse FOULETIER	Mireille
BOCHU	Bernard
BOJO	Gilles
BOKSENBAUM	Claude
BOLOPION	Alain
BONNARD	Bernard
BORRIONE	Dominique
BOUCHACOURT	Michel
BRINI	Jean
BRION	Bernard
CAIRE	Jean-Pierre
CAMEL	Denis
CAPERAN	Philippe
CAPLAIN	Michel
CAPOLINO	Gérard
CASPI	Paul
CHAN-TUNG	Nam
CHASSANDE	Jean-Pierre
CHATAIN	Dominique
CHEHIKIAN	Alain
CHIRAMELLA	Yves
CHILO	Jean
CHUPIN	Jean-Claude
COLONNA	Jean-François
COMITI	Jacques
CORDET	Christian
COUDURIER	Lucien

COUTAZ	Jean-Louis
CREUTIN	Jean-Dominique
DAO	Trongtich
DARONDEAU	Philippe
DAVID	Bertrand
DE LA SEN	Manuel
DELACHAUME	Jean-Claude
DENAT	André
DESCHIZEAUX née CHERUY	Marie-Noëlle
DIJON	Jean
DOREMUS	Pierre
DUPEUX	Michel
EL ADHAM	Karim
EL OMAR	Fovaz
EL-HENNAWY	Adel
ETAY	Jacqueline
FABRE épouse MAXIMOVITCH	Suzanne
FAURE-BONTE épouse MARET	Mireille
FAVIER	Denis
FAVIER	Jean-Jacques
FELIACHI	Movloud
FERYAL	Haj Hassan
FLANDRIN	Patrick
FOREST	Bernard
FORESTIER	Michel
FOSTER	Panayotis
FRANC	Jean-Pierre
GADELLE	Patrice
GARDAN	Yvon
GENIN	Jacques
GERVASON	Georges
GILORMINI	Pierre
GINOUX	Jean-Louis
GOUMIRI	Louis
GROC	Bernard
GROSJEAN	André
GUEDON	Jean-Yves
GUERIN	Jean-Claude
GUESSOUS	Anas
GUIBOUD-RIBAUD	Serge
HALBWACHS	Nicolas
HAMMOURI	Hassan
HEDEIROS SILIVEIRA	Hamilton
HERAULT	Jeanny
HONER	Claude
HUECKEL	Tomasz
IGNAT	Michel
ILIADIS	Athananios
JANIN	Gérard
JERRAYA	Ahmed Amine
JUTTEN	Christian
KAHIL	Hassan
KHUONGQUANG	Dong
KILLIS	Andreas
KONE	Ali
LABEAU	Michel
LACAZE	Alain
LACROIX	Jean-Claude
LANG	Jean-Claude
LATHUILLERE	Chantal

LATY	Pierre
LAUGIER	Christian
LE CADRE	Jean-Pierre
LE GARDEYR	René
LE NEST	Jean-François
LE THIESSE	Jean-Claude
LEMAIGNAN	Clement
LEMUET	Daniel
LEVEQUE	Jean-Luc
LONDICHE	Henry
L'HERITIER	Philippe
MAGNIN	Thierry
MAISON	François
MAMWI	Abdullah
MANTEL épouse SIEBERT	Elisabeth
MARCON	Guy
MARTINEZ	Francis
MARTIN-GARIN	Lionel
MASSE	Dominique
MAZER	Emmanuel
MERCKEL	Gérard
MEUNIER	Jean
MILI	Ali
MOALLA	Mohamed
MODE	Jean-Michel
MONLLOR	Christian
MONTELLA	Claude
MORET	Frédéric
MRAYATI	Mohammed
M'SAAD	Mohammed
M'SIRDI	Kouider Nace
NEPOMIASTCHY	Pierre
NGUYEN	Trong Khoi
NGUYEN-XUAN-DANG	Michel
ORANIER	Bernard
ORTEGA MARTINEZ	Roméo
PAIDASSI	Serge
PASSERONE	Alberto
PEGON	Pierre
PIJOLAT	Christophe
POGGI	Yves
POIGNET	Jean-Claude
PONS	Michel
POU	Tong Eck
RAFINEJAD	Paiviz
RAGAIE	Harie Fikri
RAHAL	Salah
RAMA SEABRA SANTOS	Fernando
RAYAINE	Denis
RAZBAN-HAGHIGHI	Tchanguiz
RAZZOUK	Micham
REGAZZONI	Gilles
RIQUET	Jean-Pierre
ROBACH	Chantal
ROBERT	Yves
ROGEZ	Jacques
ROHMER	Jean
ROUSSEL	Claude
SAAD	Abdallah
SAAD	Youcef

SABRY	Mohamed Nabi
SALON	Marie-Christine
SAUBAT épouse MARCUS	Bernadette
SCHMITT	Jean-Hubert
SCHOELLKOPF	Jean-Pierre
SCHOLL	Michel
SCHOLL	Pierre-Claude
SCHOULER	Edmond
SCHWARTZ	Jean-Luc
SEGUIN	Jean
SIWY	Jacques
SKALLI	Abdellatif
SKALLI HOUSSEYNI	Abdelali
SOUCHON	Alain
SUETRY	Jean
TALLAJ	Nizar
TEDJAR	Farouk
TEDJINI	Smail
TEYSSANDIER	Francis
THEVENODFOSSE	Pascale
TMAR	Mohamed
TRIOLLIER	Michel
TUFFELIT	Denis
TZIRITAS	Georges
VALLIN	Didier
VELAZCO	Raoul
VERDILLON	André
VERMANDE	Alain
VIKTOROVITCH	Pierre
VITRANT	Guy
WEISS	François
YAZAMI	Rachid

Remerciements

Je remercie Jacques Mossière, Professeur et Directeur de l'ENSIMAG, qui m'a fait l'honneur de bien vouloir présider ce jury.

Je remercie mes rapporteurs Roland Gerger, chef de la division CCI du CNET, et Raoul Camposano, Directeur de la division SET de GMD; et tout particulièrement Raoul Camposano qui a eu à affronter la barrière de la langue Française. Je le remercie aussi pour ses conseils éclairés.

Je remercie les membres du Jury Guy Mazaré, Professeur à l'ENSIMAG et Directeur du LGI, et Gerde Finke, Professeur à l'UJF, d' avoir bien voulu se pencher sur mon travail et en juger la qualité.

Je remercie Gabrièle Saucier, Professeur à l'ENSIMAG et Directeur du laboratoire CSI, qui a contribué à l'épuration de cette thèse. Je la remercie aussi de m'avoir donné l'occasion de ne pas rester confinée au sein du laboratoire en m'intéressant à des projets d'envergure européenne.

Je remercie Pierre Paulin, Wim Verhaegh et Jean-Christophe Gariel pour leur présence par le biais des courriers électroniques, leur soutien et leurs suggestions sur les problèmes d'ordonnancement.

Je remercie les membres du CSI pour les moments de détente partagés et spécialement Eric Chotin qui a su m'accueillir tous les matins avec la même sympathie.

Je remercie Sophie Robert pour son appui tant sur la forme de cette thèse que sur la mienne.

Je remercie Monique et Pierre, Loïc et Christine, Efix et François-Xavier Marie Jacques Emile, Laurent et Juliette, Claire et Stéphanie, Métro, Pepe et Pascal, Marcel, LaMouche et Gaston, Alain, Fodil et Toufik, Napo et Gège, Hervé et Violaine, Pascale, et tant d'autres, pour leur bonne humeur ou leur mauvaise ...

Résumé

La synthèse architecturale consiste à obtenir automatiquement la description d'un circuit en termes de blocs interconnectés à partir de sa description comportementale. Pour la synthèse dite de haut niveau cette description comportementale est de type algorithmique. A partir d'une description algorithmique une structure interne représentant un graphe de dépendance de données est extraite. Cette structure définie les contraintes de précédence entre les opérations de la spécification initiale. Les deux étapes principales de la synthèse sont l'ordonnancement et l'allocation de ressources. Chaque étape de la synthèse de haut niveau est un problème combinatoire NP-Complet. Afin de diminuer la complexité de l'espace de solutions à explorer, ces deux étapes sont généralement réalisées séparément. Dans un souci d'universalité, des architectures cibles et des heuristiques flexibles par rapport à ces architectures ont été définies.

Dans un premier temps, l'ordonnancement affecte les tâches à des unités de temps et détermine le nombre d'opérateurs. La méthode d'ordonnancement choisie est une amélioration de la méthode orientée par les forces.

Dans un second temps, l'allocation de ressources assigne des opérateurs physiques aux opérations, des registres aux variables et des connexions aux transferts de données de la description initiale. Cette dernière est également scindée en deux étapes l'allocation de registres et d'opérateurs est réalisée en premier lieu avec comme objectif de préparer l'étape suivante d'allocation des connexions.

Une méthode d'allocation de registres et d'opérateurs originale est définie. Celle-ci a comme objectif d'être flexible par rapport à l'architecture cible du circuit. Cette architecture pourra utiliser des multiplexeurs ou des bus, des registres dédiés aux opérateurs ou des registres communs. Une phase d'alignement des opérandes exploite la commutativité des opérateurs dans le but de minimiser les interconnexions. La phase d'allocation des interconnexions est définie spécifiquement pour chaque architecture cible.

Toutes ces méthodes sont implantées dans un système complet fait de deux parties : la synthèse de haut niveau et la synthèse RTL.

Mots clefs

Synthèse architecturale, Synthèse de haut niveau, Synthèse RTL, Compilateur de silicium, Ordonnancement, Allocation de ressources, Allocation de registres, Architecture cible

Abstract

Architectural synthesis consists in the automatic block netlist generation of a design from its behavioural description. For high level synthesis, this description is an algorithmic specification. from which is extracted an internal format representing data dependencies. It defines the precedence relations between operations. The two main steps of high level synthesis are scheduling and resource allocation. Each of these steps is a NP-Complete problem. To reduce the solution space exploration, scheduling and resource allocation are realised separately. To be as independent as possible from an application domain, architecture styles and flexible heuristics dedicated to these styles have been defined.

First of all, scheduling assigns each operation to a time unit. A force directed scheduling method has been chosen.

Then, the resource allocation assigns physical operators to operations, registers to variables and connections to data transfers. This step is divided into two tasks, register and operator allocation is performed first and prepares the next task namely the connection allocation.

An original method for resource allocation has been defined. Its main objective is to be flexible with respect to architecture styles which can be either multiplexer based or bus based, with dedicated or common registers. Then an operand alignment method exploits operator commutativity. At the end, connection allocation is dedicated to each architecture style.

All these methods are implemented in a complete system made up of two parts : high level synthesis and RTL synthesis.

Table des matières

Introduction

Depuis l'arrivée des circuits Very Large Scale Integration (VLSI), la complexité des circuits implantés n'est plus limitée par la technologie, mais par la phase de conception. Un concepteur ne peut plus réaliser un circuit sans l'aide d'outils d'automatisation tant la complexité du circuit 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 [FARL 88].
L'introduction massive d'outils de synthèse dans les environnements de conception assistée par ordinateur de circuits intégrés constitue le fait le plus marquant de la dernière décennie. Les outils introduits récemment réalisent la synthèse de fonctions Booléennes et de machines d'états finis sur des cibles technologiques variées, telles les cellules standards ou les réseaux programmables ... Des méthodes et outils plus ambitieux sont à l'étude. Ils ont comme objectif de générer l'architecture d'un circuit en termes de blocs élémentaires interconnectés à partir d'une spécification comportementale. Ce sont les outils de synthèse architecturale. Selon le niveau de description comportementale, on distingue deux niveaux de synthèse: la synthèse dite de haut niveau, qui part d'une description de type algorithmique, et la synthèse dite RTL (Register Transfer Level), qui part d'une description en termes de transferts de registres dans laquelle les barrières temporelles entre les opérations sont fixées.

Une spécification algorithmique décrit les opérations à réaliser par le circuit, ainsi que leur relation de dépendance. Les deux principales tâches réalisées par la synthèse de haut niveau sont l'ordonnancement des opérations et l'allocation de ressources. Les méthodes de synthèse diffèrent par les techniques d'ordonnancement et d'allocation de ressources, par l'ordre des étapes de synthèse, ainsi que par le style d'architecture défini qui dépend du domaine d'application ciblé.
L'ordonnancement affecte une unité de temps à chaque opération. L'allocation de ressources comprend plusieurs tâches: la sélection du type de ressources dans une bibliothèque, la détermination du nombre de ressources de chaque type et l'assignation de ressources (opérateurs physiques, registres et connexions) aux opérations. Tous ces problèmes sont connus pour être généralement NP-Complets. Leurs critères d'optimisation sont la surface du circuit et ses performances. Ces deux critères ne peuvent être qu'estimés lors de la synthèse de façon parfois très approximatives. En effet, la surface et les performances réelles d'un circuit ne peuvent être connues sans réaliser le placement et le routage définitifs. Par conséquent, les surfaces de routage et la topologie du circuit sont le plus souvent négligées. De plus, la partie de contrôle et la connexion de la partie de contrôle et de la partie opérative ne sont pas prises en compte.
En raison de l'estimation des critères et de la taille de l'espace de solutions à observer, il est certain qu'un système de synthèse ne peut pas donner de réponse convenable en une seule itération. Un

concepteur essayera donc plusieurs solutions, en changeant les contraintes initiales, l'architecture et la technologie cibles avant d'obtenir une solution satisfaisante. Un système de synthèse de haut niveau doit donc être hautement interactif. Une réponse rapide à des problèmes NP-Complets et interdépendants est nécessaire. Il parait donc évident que la synthèse doit être décomposée par étapes: l'ordonnancement et l'allocation de ressources. Selon les systèmes, les étapes d'ordonnancement et d'allocation sont effectuées soit séparément, soit simultanément, soit partiellement simultanément en ne réalisant qu'une partie de l'allocation de ressources.

Ainsi, le système MAHA [PARK 86] réalise simultanément l'assignation d'opérateurs et l'ordonnancement. FACET [TSEN 83], HAL [PAUL 89a], ASA [FONK 89] et EASY [STOK 91] définissent le nombre de ressources pendant l'ordonnancement. Le système ASA a recours à des techniques d'intelligence artificielle. CADDY [KRAM 88] et HYPER [CHU 89] déterminent le nombre de ressources avant l'ordonnancement. CADDY réalise simultanément l'ordonnancement et la sélection du type de ressource.

Les systèmes YSC [CAMP 88a], HIS [BERG 91a], GENARC [GOUL 90] et BUD [FARL 86a et b], intégré dans CMUDA, effectuent d'abord l'assignation d'opérateurs puis l'ordonnancement. MIMOLA réalise l'ordonnancement, la sélection du type d'opérateurs et l'assignation de ressources simultanément, en réalisant l'assignation pendant un ordonnancement par pas de contrôle.

Le système Cathedral II remet en cause l'ordonnancement à chaque étape de synthèse: l'allocation de bancs de registres, l'allocation des connexions.

Les approches réalisant toutes ces étapes simultanément par un recuit simulé, telles que [SAFI 90] [DEVA 87], ne semblent pas réalistes. Le recuit simulé, défini ultérieurement, est reconnu pour être un algorithme très long à trouver une solution.

Deux approches sont envisageables pour la synthèse RTL. Une première approche est dirigée par les données. Elle part d'une description RTL généralisée et implante les opérations au niveau du bit en les affectant à des cellules standards. A l'opposé, une deuxième approche est dirigée par le contrôle. Elle a pour architecture cible un circuit formé d'une partie opérative et d'une partie de contrôle. La description RTL représente alors un organigramme de contrôle, duquel une machine d'états finis est extraite et une partie opérative est générée par allocation des connexions. Ce type de synthèse a l'avantage d'offrir la possibilité d'observer un large espace de solutions. Il permet d'essayer différentes architectures et technologies cibles pour la synthèse de la machine d'états finis et différents types de synchronisation entre la partie opérative et la partie de contrôle. Nous nous intéressons, dans cette thèse, à la synthèse RTL dirigée par le contrôle qui correspond aux dernières étapes de la synthèse de haut niveau.

Nous utilisons une spécification initiale de haut niveau, donnée sous forme de graphe de dépendance de données entre les opérations. Bien que les phases d'ordonnancement et d'allocation de ressources soient interdépendantes, ces deux tâches sont traitées successivement. Dans un premier temps, l'ordonnancement affecte les tâches à des unités de temps et détermine le nombre d'opérateurs. Afin de diminuer la complexité de l'espace de solutions à explorer, l'allocation de ressources est réalisée

après la phase d'ordonnancement. Cette dernière phase est également scindée en deux étapes, à savoir l'allocation de registres et d'opérateurs et l'allocation des connexions. L'allocation de registres et d'opérateurs est réalisée en premier lieu avec comme objectif de préparer l'étape suivante d'allocation des connexions. L'originalité des méthodes proposées consiste en leur faible complexité.
Certains systèmes de synthèse se limitent volontairement à un type d'application spécifique. Dans ce but, un style d'architecture unique, répondant aux exigences de ce domaine d'application, est défini. L'objectif prioritaire des méthodes décrites dans cette thèse est la flexibilité, afin de prendre en compte les critères et les contraintes de cibles architecturales variées.

Dans un premier chapitre, la méthode d'ordonnancement choisie, qui est une amélioration de la méthode orientée par les forces [PAUL 87], est étudiée. Le deuxième chapitre expose une méthode originale d'allocation de ressources, flexible par rapport aux styles d'architecture. Le troisième chapitre présente une méthode originale d'alignement des opérandes exploitant la commutativité des opérations et l'allocation des connexions dédiée à chaque style d'architecture. Le dernier chapitre aborde le passage de la théorie à la pratique. Le choix d'un domaine d'application spécifique est discuté. Le système complet de synthèse architecturale implanté et utilisant les méthodes définies dans les chapitres précédents est décrit. Ce système se décompose en deux parties: la synthèse RTL et la synthèse de haut niveau.

Aides à la lecture

Afin d'aider un lecteur néophyte dans l'appréhension des méthodes décrites pour la résolution des problèmes de la synthèse architecturale, nous avons choisi de présenter d'abord, en détail, une méthode particulière, puis les autres méthodes envisageables pour le même problème. Chacun des chapitres suivants commence donc par la description de notre méthode et finit par un récapitulatif de l'état de l'art du domaine concerné.

Pour permettre une lecture rapide de cette thèse, certains paragraphes sont marqués par une bordure (comme l'est celui-ci). Ce sont des paragraphes que nous considérons comme importants.

Enfin, en annexe, se trouvent un index définissant aussi la traduction anglaise des termes les plus couramment utilisés, un glossaire et la liste des notations des variables définies.

Chapitre I

ORDONNANCEMENT

Les problèmes d'ordonnancement constituent un domaine important de la recherche opérationnelle. Ils concernent en effet de nombreux domaines industriels.

La méthode d'ordonnancement qui est étudiée ici est dédiée à la synthèse architecturale de circuits intégrés. C'est la méthode d'ordonnancement orienté par les forces, définie dans [PAUL 87]. Cette méthode réalise l'ordonnancement de manière séquentielle. Deux améliorations ont été proposées: la première permet de mieux anticiper l'effet de l'ordonnancement d'une opération sur les prochains pas de l'algorithme; la seconde permet d'optimiser la surface du circuit généré. Cette heuristique d'ordonnancement nécessite d'être accompagnée d'une méthode d'ordonnancement des boucles d'une spécification initiale.

Une première section de ce chapitre définit les données initiales de la synthèse de haut niveau, nommément un graphe de dépendance de données ou de précédence. La deuxième section définit brièvement les étapes principales de la synthèse de haut niveau ou synthèse architecturale.

Une troisième section présente les problèmes d'ordonnancement pour la synthèse de haut niveau et les classifie selon leurs critères d'optimisation et leur état de départ. La complexité de ces problèmes est ensuite discutée.

La méthode d'ordonnancement orienté par les forces et les améliorations que nous proposons sont détaillées en quatrième section.

Quelques autres méthodes d'ordonnancement pour la synthèse de haut niveau sont abordées en cinquième section.

Les résultats de l'algorithme de base, tel qu'il a été présenté par Paulin et Knigh [PAUL 89b], et de l'algorithme amélioré sont présentés dans la sixième section de ce chapitre.

Enfin une dernière section traite de l'ordonnancement de boucles imbriquées ou successives.

I.1. Spécifications initiales

La description initiale du circuit est supposée être donnée, par le concepteur, sous forme d'une liste d'opérations élémentaires pouvant être conditionnelles. Les opérations élémentaires correspondent à des fonctions réalisables sur des opérateurs physiques implantés ultérieurement dans le circuit. La liste d'opérations élémentaires décrite par une séquence peut être comparée à un programme exprimant un algorithme. Néanmoins, nous ne prétendons pas, ici, générer un circuit à partir d'un programme de haut niveau, mais nous nous contenterons de structures restrictives. Comme en programmation, cette description initiale est faite par une liste dont l'ordre est supposé respecter la dépendance de données; elle introduit une sérialisation abusive des opérations. Il convient d'analyser la réelle dépendance de données, afin de tirer parti du parallélisme potentiel et de proposer un ensemble intéressant de solutions architecturales. L'étude de cette dépendance de données conduit à un format intermédiaire classiquement utilisé en synthèse architecturale.

I.1.1. Opération élémentaire et branchement conditionnel

Définition I.1: Une *opération élémentaire* t est définie par $z = op\ (a_1, a_2, ..., a_n)$; l'opération t définit z et utilise les opérandes $a_1, a_2, ..., a_n$.

L'opération t correspond nécessairement à un opérateur *op* existant en bibliothèque. Les opérandes et les valeurs définies par une opération sont des données de taille fixée en nombre de bit; elles sont implicitement liées à des registres existant en bibliothèque.

Définition I.2: Un *branchement conditionnel* est une instruction particulière de la forme suivante "si P alors L_1 sinon L_2", où L_1 et L_2 sont des listes d'opérations élémentaires et P est une liste d'opérations élémentaires permettant le calcul d'un prédicat de branchement.

Définition I.3: Les listes d'opérations élémentaires L_1 et L_2 sont deux listes *mutuellement exclusives*. Les opérations élémentaires qui les définissent sont *mutuellement exclusives*, deux à deux, par extension.

La spécification initiale du circuit est donnée par une séquence d'opérations élémentaires et de branchements conditionnels.
Par exemple, la figure I.1 donne des listes d'opérations élémentaires contrôlées par un branchement conditionnel. L'opération (1) est une opération élémentaire s'exécutant sur un opérateur existant en bibliothèque et capable de réaliser une addition. L'opération (2) définit un prédicat de branchement conditionnel et est liée à un élément de bibliothèque capable de réaliser une comparaison. Le branchement conditionnel de cette description initiale définit deux listes d'opérations mutuellement exclusives: la liste formée de la seule opération (5) et celle formée des opérations (3) et (4).

```
(1)  A:=A+1
     Si
(2)    A =6
     alors
(3)    B:= B+C
(4)    E := B-C
     Sinon
(5)    B:= B-D
     FinSi
(6)  F:=A+B
(7)  G:=C+D
```

Figure I.1: Listes d'opérations élémentaires et de branchements conditionnels

I.1.2. Graphe de flot de contrôle

Un premier graphe est construit pour représenter le contrôle des opérations élémentaires; c'est un graphe de flot de contrôle [AHO 88]. Il est basé sur la définition de blocs de base.

Définition I.4: Une *opération de tête* est une opération élémentaire t qui vérifie l'une des quatre propriétés suivantes:

i) t est la première opération de la spécification initiale.

ii) t est la première instruction d'une liste d'opérations élémentaires atteinte par un branchement conditionnel.

iii) t est la première opération élémentaire suivant un branchement conditionnel.

iv) t est la première opération élémentaire d'un calcul de prédicat de branchement.

Dans l'exemple de la figure I.1, l'opération (1) vérifie la propriété (i), les opérations (3) et (5) vérifient la propriété (ii), l'opération (6) vérifie la propriété (iii) et l'opération (2) vérifie la propriété (iv). Toutes ces opérations élémentaires sont donc des opérations de tête.

Définition I.5: Un *bloc de base* est un bloc commençant par une opération de tête et est constitué de toutes les opérations suivantes, qui ne sont pas des opérations de tête.

Un branchement conditionnel définit donc trois blocs de base: un bloc de base de calcul du prédicat de branchement, un bloc de base atteint si le prédicat est vrai et un bloc de base atteint si le prédicat et faux.

Les blocs de bases de la figure I.1 sont au nombre de 5 et sont listés figure I.2.

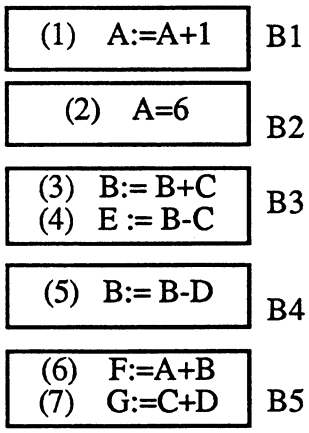


Figure I.2: Blocs de bases de la figure I.1

Définition I.6: Un *Graphe de Flot de Contrôle (GFC)* est un graphe orienté G(B∪BC,A), où B est l'ensemble des sommets représentant les blocs de base, BC est l'ensemble des sommets de branchement conditionnel et A est l'ensemble des arcs.

Un arc a_{ij} lie le bloc B_i au bloc B_j si B_i suit immédiatement B_j dans l'ordre de la spécification initiale.

Le bloc de base contenant la première opération élémentaire de la spécification initiale est le sommet racine du GFC.

Un branchement conditionnel est représenté par les trois sommets représentant les blocs de base qu'il définit, un sommet de branchement conditionnel et trois arcs: un arc lie le bloc de calcul de prédicat au sommet de branchement conditionnel, un arc lie le sommet de branchement conditionnel au bloc de base atteint si le prédicat est vrai (cet arc est étiqueté par la valeur *vrai*) et un arc lie le sommet de branchement conditionnel au bloc de base atteint si le prédicat est faux (cet arc est étiqueté par la valeur *faux*).

Le GFC résultant de la spécification initiale de la figure I.1 est donné figure I.3.

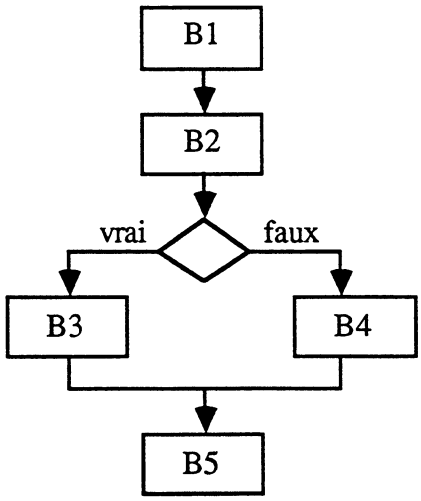


Figure I.3: GFC de la figure I.1

En supposant qu'une liste d'opérations donnée constitue le corps d'une boucle, le contrôle de cette boucle se fait de la façon classique suivante: "Si condition de boucle satisfaite alors répéter le corps de

la boucle sinon sortir de la boucle".

Une boucle définit un branchement conditionnel particulier. Elle est également représentée dans un GFC par un sommet représentant le bloc de base de calcul de la condition d'arrêt de la boucle, des blocs de base représentant le corps de la boucle et un sommet de branchement en cas de sortie de boucle. Un tel graphe de contrôle indique que le corps de la boucle et le calcul de la condition d'arrêt de la boucle sont potentiellement considérés en parallèle.

Si on suppose maintenant que la spécification initiale est celle de la figure I.4(a), le GFC résultant est celui de la figure I.4(b), où le bloc B_6 contient l'opération (8) et le bloc B_7 l'opération (9)

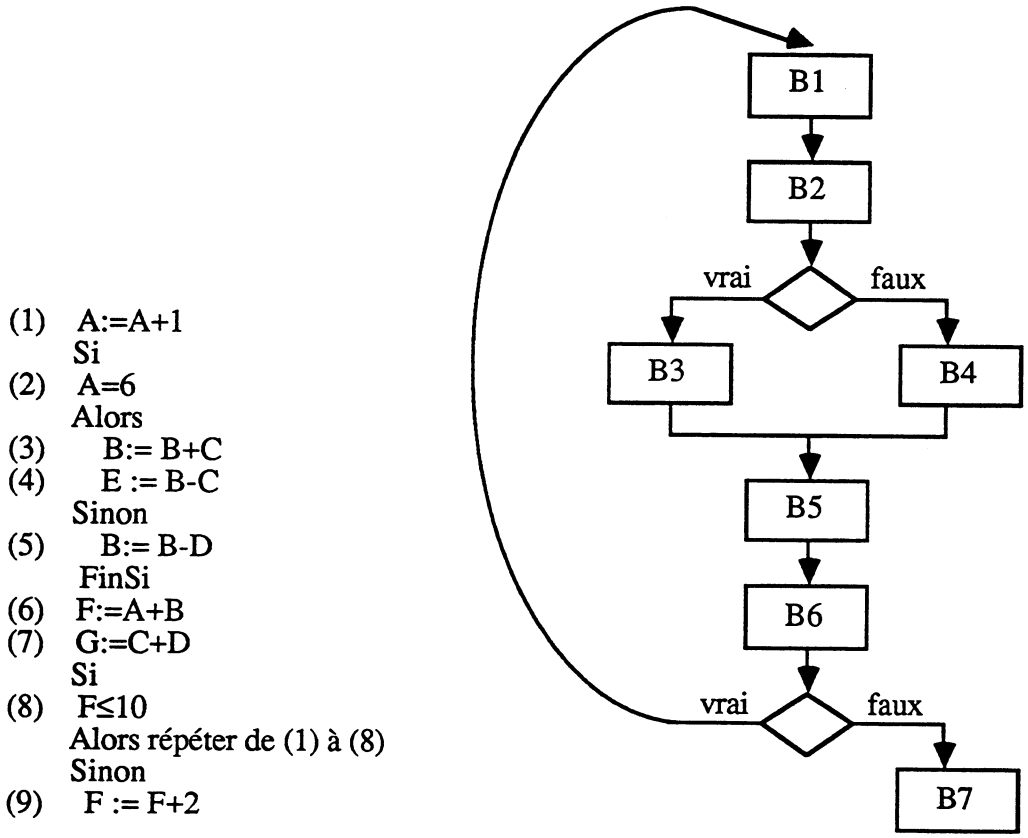


Figure I.4: Spécification initiale et GFC d'une boucle

Les boucles peuvent être identifiées dans un GFC grâce à la notion de sommet dominant [AHO 88].

Définition I.7: B_d est un *sommet dominant* B_i, si tout chemin du GFC, allant du sommet racine au sommet B_i, contient forcément B_d.

Définition I.8: un *arc de retour* est un arc dont le sommet de destination domine le sommet de départ.

Toutes les boucles d'un GFC peuvent être repérées en identifiant les arcs de retour du GFC. Une boucle est alors composée du sommet d'arrivée de l'arc de retour (sommet en-tête) et de l'ensemble des sommets atteints à partir du sommet en-tête, sans passer par le sommet de départ de l'arc de retour. Une boucle a deux propriétés principales: elle a un point d'entrée unique, dominant tous les autres sommets de la boucle, appelé l'en-tête de la boucle, et elle a un arc de retour allant vers l'en-

tête. Deux boucles ayant le même sommet en-tête sont regroupées et considérées comme une seule boucle.
Deux boucles, ainsi identifiées, sont soit disjointes, soit imbriquées.

Définition I.9: Une *boucle élémentaire* est une boucle qui n'en contient aucune autre.

Définition I.10: Une boucle Bl_1 est une *boucle imbriquée* dans une boucle Bl_2 si Bl_1 est entièrement contenue dans Bl_2.

Définition I.11: Des *boucles successives* sont imbriquées dans une même boucle.

I.1.3. Graphe de dépendance de données et graphe de précédence

Un bloc de base est associé à un graphe orienté acyclique exprimant la dépendance des données. En synthèse de haut niveau, un tel graphe est appelé un graphe de dépendance de données.

Définition I.12: Un *Graphe de Dépendance de Données (GDD)* d'un bloc de base est un graphe G(T,U), où T est l'ensemble des opérations élémentaires du bloc de base et U est l'ensemble des arcs de G. Un arc u_{ij} lie l'opération t_i à l'opération tj, si une donnée définie par t_i est une opérande de t_j. Les constantes sont représentées par des sommets particuliers.

Une boucle élémentaire est décrite par un ensemble de blocs de base sans arc de retour, contenant un sommet d'en-tête dominant tous les autres. Jusqu'à présent, la dépendance des données est exprimée, séparément pour chaque bloc de base, par un GDD. L'inconvénient de cette représentation est que deux instructions, inclues dans deux blocs de base différents et écrites l'une avant l'autre dans une description initiale, resteront dans cet ordre, lors de transformations à l'intérieur d'un bloc de base. Par exemple, dans la description de la figure I.4(a), l'instruction (7) est toujours après l'instruction (1) si des transformations sont réalisées à l'intérieur de chaque bloc de base séparément, alors que ces deux opérations ne sont pas liées par une dépendance de données.
Nous proposons donc d'associer à une boucle élémentaire un seul GDD.

Définition I.13: Le *GDD d'une boucle élémentaire*, décrite par les blocs de base B_1 et B_N, est le graphe G(T, U), où $T=T_1 \cup ... \cup T_N$ et $U= U_1 \cup ... \cup U_N \cup U_B$, formé des GDD $G_i(T_i,U_i)$ de chaque bloc de base B_i et des arcs U_B, tels que un arc u_{ij} existe entre une opération t_i d'un bloc de base B_i et une opération t_j d'un bloc de base B_j si et seulement si une valeur définie par t_i est utilisée par t_j.

Le GDD de la boucle de la figure I.4(b) est représenté par la figure I.5. Dans cette figure, chaque sommet est étiqueté par l'opération élémentaire qu'il représente.

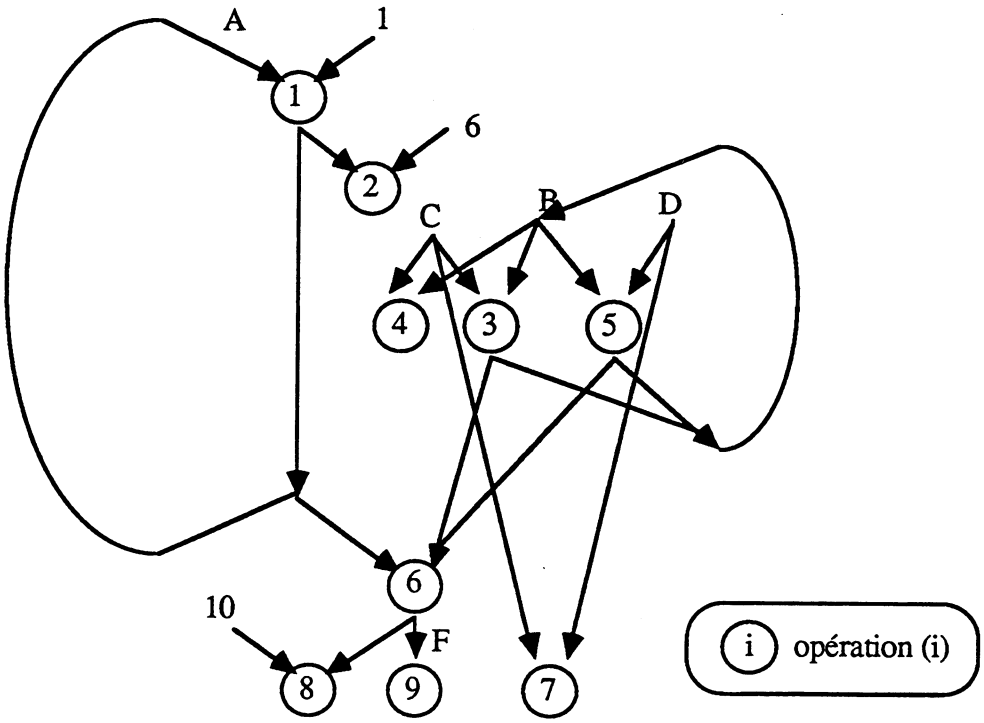


Figure I.5: GDD de la boucle de la figure I.4

A partir de la définition de GFC et de GDD, une relation de précédence peut être énoncée. Cette relation lie les opérations élémentaires de la spécification initiale.

Les *contraintes de précédence*, qu'induit cette représentation, sont de deux types: les contraintes de dépendance de données et les contraintes de branchement conditionnel. Ces dernières expriment le fait que l'opération élémentaire calculant le prédicat de branchement doit précéder les opérations élémentaires des blocs de base mutuellement exclusifs que le branchement définit.

Les notions de sommet dominant et d'arc de retour pour les GFC sont étendues aux GDD, en considérant comme sommets racines ceux représentant les premières opérations (dans l'ordre de la spécification initiale) définissant chaque donnée.

Définition I.14: Le *Graphe de Précédence (GP)* d'une boucle élémentaire est le graphe $G(T, U_P)$, où T, l'ensemble des sommets, représente l'ensemble des opérations élémentaires et U_P, l'ensemble des arcs, représente l'ensemble des *contraintes de précédence*. Les contraintes de précédence sont de deux types: les contraintes de dépendance de données (représentées dans U) et les contraintes de dépendance de contrôle (représentées dans U_C). U est l'ensemble des arcs de dépendance de données du GDD de la boucle élémentaire (définition I.13) en supprimant les arcs de retour de dépendance de données. U_C représente les relations de précédence entre l'opération de calcul de prédicat de branchement conditionnel et les opérations mutuellement exclusives que le branchement conditionnel définit. $U_P=U \cup U_C$.

La relation de précédence est transitive. Cependant, seules les relations de précédence directes, induites par la dépendance de données sont représentées.

Le GP du corps de la boucle de la figure I.4(b) est représenté figure I.6. Sur cette figure, les arcs en gras représentent les contraintes de précédence définies par le branchement conditionnel.

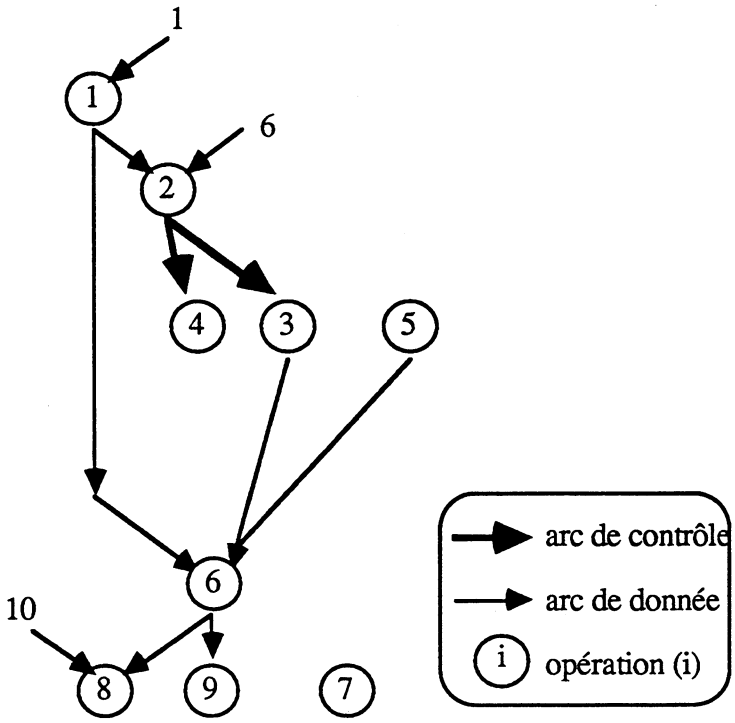


Figure I.6: GP de la figure I.4(b).

Les graphes de représentation du comportement du circuit, décrits dans cette section, sont des modèles utilisés en synthèse de haut niveau comme points de départ des différentes phases de synthèse. Les contraintes du graphe de précédence (GP définition I.14) sont naturellement respectées lors de l'ordonnancement des opérations élémentaires. Le graphe de dépendance de données (GDD définition I.13) définit entre autre les transferts de données entre ces opérations. Il est analysé pour l'allocation d'opérateurs, de registres et de connexions. Le graphe de flot de contrôle (GFC définition I.6) est utilisé pour connaître, à chaque étape, les opérations mutuellement exclusives et, en fin de synthèse, pour générer une machine d'états finis.

I.2. Présentation générale de la synthèse de haut niveau

La synthèse de haut niveau consiste à ordonnancer les opérations et à allouer des ressources physiques à ces opérations. Le résultat sera un circuit réalisé par une Partie Opérative (PO) connectant les ressources allouées et une Partie de Contrôle (PC) contrôlant le séquencement des opérations. La partie de contrôle est communément représentée par une machine d'états finis.
La première étape de la synthèse de haut niveau sélectionne, pour chaque opération, un opérateur en bibliothèque capable de l'exécuter. Cette étape est habituellement appelée la *sélection des types d'opérateurs*. Après cette phase, chaque sommet d'un GDD ou d'un GP est associé à un opérateur existant en bibliothèque.
Pour l'ordonnancement, l'échelle de temps est discrétisée en *pas de contrôle*. Un pas de contrôle est un cycle de base de la machine d'états finis. C'est, le plus souvent, un *cycle d'horloge*.

Définition I.15: Une étape de la synthèse de haut niveau, appelée l'*ordonnancement* affecte, à chaque opération d'une description initiale, un pas de contrôle en respectant les contraintes de précédence de la spécification initiale.

Les opérateurs sont parfois appelés des *unités fonctionnelles*. Un opérateur est caractérisé par les opérations qu'il réalise. Chaque opération a un délai qui correspond au temps d'exécution de cette opération sur l'opérateur.
Les opérateurs *multicycles* sont des opérateurs nécessitant plus d'un pas de contrôle pour réaliser une opération.

Définition I.16: Une opération ayant une *durée d(t)* en nano-secondes sera réalisée en *l(t)* pas de contrôle, si la durée d'un pas de contrôle est *c*; *l(t)*est défini par $l(t) = \lceil d(t)/c \rceil$. La durée en pas de contrôle est appelée le *délai* d'une opération.

L'ordonnancement doit respecter les contraintes de précédence d'une spécification initiale et le délai de chaque opération. Cette étape affecte les opérations élémentaires à des pas de contrôle en respectant les contraintes de précédence définies par un GP.

Définition I.17: L'*allocation de ressources* consiste en la sélection des types d'opérateurs, la définition du nombre de ressources de la PO et l'assignation de ressources.

Une fois le nombre de ressources de chaque type déterminé, la PO du circuit est partiellement construite en instanciant le nombre de ressources de chaque type dans la PO.

Définition I.18: L'*assignation d'opérateurs* consiste à assigner une instance d'opérateur physique à chaque opération. L'*assignation de registres* assigne des registres aux données et l'*assignation des connexions* assigne des connexions (multiplexeurs, bus et interconnexions) aux transferts de données.

I.3. Présentation de l'ordonnancement

Pour le problème d'ordonnancement, nous considérons que l'étape de sélection du type d'opérateur a été réalisée et, dans un premier temps, que l'ordonnancement sera accompli pour une boucle élémentaire. Nous verrons dans la section I.7 le cas d'ordonnancement de boucles successives ou imbriquées.

En effet, les boucles, dans une description initiale, impliquent parfois qu'une valeur, définie en fin de boucle, soit réutilisée au début du corps de la boucle. En représentant cette dépendance dans le GDD d'une boucle, celui-ci contiendra des circuits. Les circuits, dans un graphe de dépendance, ne peuvent pas être ordonnancés, puisque, dans ce cas, la relation de précédence n'est pas antisymétrique.

Le problème d'ordonnancement d'une boucle élémentaire consiste à assigner chaque opération d'une description initiale à un pas de contrôle, en respectant les contraintes de précédence, représentées dans le GP de la boucle. Un GP G(T,U) est sans circuit; les arcs de U définissent donc un ordre partiel $\succcurlyeq$ sur les opérations.

Pour une durée donnée de pas de contrôle, chaque opération t de T est associée à son délai l(t) et à un opérateur existant en bibliothèque. Soient O l'ensemble des opérateurs op et, pour chaque opérateur op de O, la fonction $SELEC_{op}$ de T dans {0, 1} qui vaut 1 si l'opérateur op a été sélectionné pour l'opération p et 0 sinon. La phase de sélection du type d'opérateur a déterminé, pour chaque opérateur, la valeur d'une fonction $SELEC_{op}(t)$.

Définition I.19: Soit P un ensemble ordonné d'entiers numérotés de 1 à L représentant les pas de contrôle, un *ordonnancement Ord* est une application de T vers P qui associe à chaque t de T un pas de contrôle p de P.

Définition I.20: La durée ou *date limite de fin d'ordonnancement* L(Ord) et la *largeur d'un ordonnancement* m(Ord) sont définies par:

$L(Ord) = \text{Max } t \in T \{ (Ord(t)+l(t)) \}$

$m(Ord) = \text{Max } p \in P \{ \| \{t / Ord(t) \leq p < Ord(t)+l(t)\} \| \}$

En synthèse de circuits intégrés, toutes les opérations ne s'exécutent pas sur le même opérateur. Le critère à optimiser n'est pas la largeur d'un ordonnancement mais une estimation de la surface du circuit par la somme des largeurs de chaque type d'opérateur.

Pour chaque opération, un seul opérateur est sélectionné, ce qui implique la contrainte suivante:

$$\sum_{op \in O} SELEC_{op}(t)=1$$

Pour chaque pas de contrôle p, soit $S_{op}(p)$ le sous-ensemble des opérations t de T en cours d'exécution à l'instant p, $S_{op}(p)$ est l'ensemble suivant:

$S_{op}(p) = \{t \in T / Ord(t) \leq p < Ord(t)+l(t)\}$

Nb(op), le nombre d'instances de l'opérateur op nécessaires, est défini par:

$$Nb(op) = \text{Max } p \in P\Big(\sum_{S_{op}(p)} SELEC_{op}(t) \Big)$$

Soit la fonction Surf, de l'ensemble O des opérateurs vers l'ensemble des entiers, représentant le coût en surface de chaque type d'opérateurs,
alors $m_O(Ord)$ est défini par: $m_O(Ord) = \sum_{op \in O} Surf(op) * Nb(op)$.

Définition I.21: La *largeur en opérateurs* d'un ordonnancement est $m_O(Ord)$, définie par:

$$m_O(Ord) = \sum_{op \in O} Surf(op) * Max\ p \in P\{ \| \{t / Ord(t) \leq p < Ord(t)+l(t)\} \text{ et } SELEC_{op}(t)=1 \| \}.$$

Le nombre d'opérateurs de chaque type Nb(op) est défini par:

$$Nb(op) = Max\ p \in P(\sum_{S_{op}(p)} SELEC_{op}(t))$$

Pour respecter l'ordre partiel, et donc la dépendance de données entre les opérations et les branchements conditionnels, l'ordonnancement Ord devra respecter les contraintes de précédence.

Définition I.22: Les *contraintes de précédence* d'un ordonnancement Ord, par rapport à un ordre partiel sur les opérations $\succcurlyeq$, sont respectées si et seulement si:

$\forall\ t_1$ et $t_2 \in T / t_1 \succcurlyeq t_2$, $Ord(t_1) > Ord(t_2)$

Les critères de l'ordonnancement sont donc les minimisations de $m_O(Ord)$ et de L(Ord). La seule contrainte de l'ordonnancement est celle de précédence. $m_O(Ord)$ ne prend en compte que le coût des opérateurs. Une fonction coût, estimant la surface du circuit, peut également prendre en compte le nombre de registres et le nombre de connexions.

Dans le cadre de la synthèse de haut niveau, l'assignation des ressources est parfois déterminée avant ou durant l'ordonnancement. Dans ce cas, $m_O(Ord)$ est fixée; cependant, des contraintes sont définies pour respecter l'assignation d'opérateurs.
Une fonction $ASSIGN_{ins}$ est définie de T vers {0, 1} pour chaque instance d'opérateur ins de I. L'ensemble I étant déterminé par le nombre d'opérateurs de chaque type Nb(op), Nb(op) instances d'opérateurs sont créées pour chaque type d'opérateur op. La fonction $ASSIGN_{ins}$ vaut 1 si l'opération t est assignée à l'instance d'opérateur ins et 0 sinon. L'assignation de ressources implique une contrainte sur l'ordonnancement qui se formule de la manière suivante:

$$\forall\ ins \in I,\ Max\ p \in P\ (\sum_{S_{op}(p)} ASSIGN_{ins}(t)) \leq 1.$$

Cette contrainte définit qu'à chaque instant, une instance de ressource *ins* ne peut être utilisée que par une seule opération t.
Chaque opération est allouée à une ressource et assignée à une instance de cette ressource et une seule, ce qui implique la contrainte suivante:

$$\sum_{ins \in I} ASSIGN_{ins}(t)=1$$

La définition du problème d'ordonnancement, donnée dans cette section, est une formulation simplifiée qui n'intègre pas certaines notions telles que les branchements conditionnels, qui seront développés lors de la présentation de la méthode d'ordonnancement que nous proposons.

I.3.1. Classification des problèmes d'ordonnancement

On distingue généralement deux classes de problèmes d'ordonnancement: l'Ordonnancement sous Contrainte Matérielle (OCM) et l'Ordonnancement sous Contrainte de Temps (OCT).
L'OCM consiste à minimiser le temps globale d'exécution des opérations élémentaires du GP, L(Ord), avec une contrainte sur le nombre d'instances d'opérateurs Nb(op), celui-ci étant fixé pour chaque type d'opérateur op. La fonction coût, pour les problèmes de la classe OCM, est donc L(Ord), donnée définition I.20.
L'OCT fixe la durée d'exécution globale du GP, L(Ord). Le critère à minimiser est le coût matériel m_O(Ord), donné définition I.21.
Selon les cas, l'allocation de ressources peut être partiellement réalisée avant l'ordonnancement. Par conséquent, le nombre d'opérateurs de chaque type peut être déterminé, ou même l'assignation d'opérateurs peut être effectuée avant l'étape d'ordonnancement. Pour chaque classe, différents états de départ et d'arrivée sont donc possibles, selon que les fonctions Nb, SELEC et ASSIGN sont définies avant, pendant ou après l'ordonnancement.
Si généralement le problème d'ordonnancement en synthèse est considéré comme un problème NP-Complet, cela n'a jamais été démontré de manière formelle, à notre connaissance.
Dans un premier temps, nous définirons donc quelques termes, liés à la théorie de la complexité des problèmes combinatoires. Dans un second temps, nous décrirons les liens entre les différentes classes de problèmes d'ordonnancement, du point de vue de la complexité. Enfin, nous démontrerons la complexité du problème d'ordonnancement dans le cadre de la synthèse architecturale.

I.3.2. Définitions liées à la théorie de la complexité

Tout d'abord, nous allons rappeler quelques définitions liées à la complexité des problèmes. [GARE 79] est un ouvrage très complet sur la théorie de la complexité des problèmes.
Un problème appartient à la classe *P*, s'il se résout en temps polynomial sur une machine déterministe. Un problème appartient à la classe *NP*, s'il peut se résoudre en un temps Polynomial sur une machine Non déterministe. Il est clair que P est inclus dans NP, mais personne n'a encore réussi à démontrer que P était différent de NP. La théorie des problèmes NP-Complets repose donc sur l'hypothèse "si NP≠P". Un problème est dit *NP-Complet* s'il appartient à NP et que tout autre problème de NP peut se transformer, en un temps polynomial, en ce problème. Intuitivement, la classe des problèmes NP-Complets, inclue dans NP-P, est la classe des problèmes les plus complexes de NP. Si un seul problème NP-Complet peut se résoudre en temps polynomial, alors tous les problèmes de NP peuvent se résoudre en temps polynomial.
Pour démontrer qu'un problème π est NP-Complet, il faut donc démontrer que ce problème appartient à la classe NP, sélectionner un problème NP-Complet connu et construire une transformation polynomiale du problème connu vers une instance du problème π.
Toute démonstration de complexité sur un problème d'optimisation combinatoire se ramène à une démonstration sur un problème de décision. Un problème se définissant de la manière suivante "(a): Trouver une solution au problème X telle que f soit minimale" est ramené au problème suivant "(b):

Existe-t-il une solution au problème X telle que f<F?". Une solution au problème (a) permet facilement de vérifier que F est supérieure au minimum. Par conséquent, si le problème de décision (b) est NP-Complet alors le problème d'optimisation combinatoire (a) est NP-Complet.
A partir de cet ensemble de propriétés, une dépendance entre les classes d'ordonnancement, pour la démonstration de la complexité, est définie.

I.3.3. Dépendance entre les classes d'ordonnancement

En réduisant le problème d'optimisation à un problème de décision, les critères (de temps et de ressources) sont ramenés à des contraintes. Il n'y a donc pas de distinction entre la classe OCT et la classe OCM pour la détermination de la complexité de l'ordonnancement. Une nouvelle classe de problèmes d'ordonnancement apparaît: c'est l'ordonnancement sous contraintes de décision (OCD). Cette formulation remplace les critères par des contraintes sur une borne supérieure de $m_O(Ord)$ et L(Ord).
Les différents points de départ du problème OCD se représentent par deux sous-classes: la classe OCD_SELEC et la classe OCD_ASSIGN. En effet, tous les critères s'expriment en tant que contraintes pour la classe OCD. En se ramenant à un problème de décision, le problème d'ordonnancement avec $SELEC_{op}$ déterminée et le problème d'ordonnancement déterminant $SELEC_{op}$ se formulent de la même manière: c'est le problème OCD_SELEC. Enfin, les problèmes d'ordonnancement partant d'une assignation fixée d'opérateurs et ceux réalisant l'assignation d'opérateurs se ramènent tous à une restriction du problème OCD_SELEC: c'est le problème OCD_ASSIGN.

Définition I.23: Le problème OCD_SELEC s'exprime donc de la manière suivante:

Soient T un ensemble d'opérations t de longueur l(t) entière et un ordre partiel $\gg$ sur T; soient un ensemble O d'opérateurs op de surface Surf(op) et une fonction de sélection du type d'opérateur $SELEC_{op}$ de T vers {0,1}; soient une date limite de fin entière L et une surface limite des opérateurs entière m_{op}. Existe-t-il un ordonnancement Ord de T vers P={1, 2, ..., L} tel que :

1) $\sum_{op \in O} Surf(op) * Max\ p \in P\ \{\ ||\{t / Ord(t) \leq p < Ord(t)+l(t)\}\ et\ SELEC_{op}(t)=1||\ \} \leq m_{op}$

2) $\forall\ t_1\ et\ t_2 \in T / t_1 \gg t_2,\ Ord(t_1) > Ord(t_2)$

3) $\forall\ t \in T,\ \sum_{op \in O} SELEC_{op}(t)=1$

Définition I.24: Le problème OCD_ASSIGN s'exprime de la manière suivante:

Soient T un ensemble d'opérations t de longueur l(t) entière et un ordre partiel $\succcurlyeq$ sur T; soient un ensemble I d'instances d'opérateurs *ins* et une fonction d'assignation des opérateurs $ASSIGN_{ins}$ de T vers {0,1}; soit une date limite de fin entière L. Existe-t-il un ordonnancement Ord de T vers P={1, 2, ..., L} tel que :

1') $\forall$ ins$\in$ I, $\| \{t / Ord(t) \leq p < Ord(t)+l(t)\}$ et $ASSIGN_{ins}(t)=1 \| \leq 1$

2) $\forall\ t_1$ et $t_2 \in T / t_1 \succcurlyeq t_2$, $Ord(t_1) > Ord(t_2)$,

3') $\forall\ t \in T$, $\sum_{ins \in I} ASSIGN_{ins}(t)=1$

I.3.4. Démonstration de la complexité du problème d'ordonnancement

[LENS 78] donne une classification des problèmes d'ordonnancement avec pour objectif de déterminer leur complexité. Les classes sont définies en fonction des critères à optimiser, du nombre de processeurs et du type de contraintes sur les tâches (préemption, contraintes potentielles, ...). Dans cette classification, le problème d'ordonnancement sous contraintes de précédence est noté $m|prec||C_{Max}$. Il représente l'ordonnancement sur m processeurs identiques avec des contraintes de précédence arbitraires, en n'autorisant pas la préemption et en minimisant la durée totale de l'ordonnancement. Cette classification ne permet pas de représenter la notion de ressources différentes utilisée en synthèse architecturale.

Nous allons démontrer que le problème d'ordonnancement avec sélection du type d'opérateur et détermination du nombre d'opérateurs (OCD_SELEC définition I.23) est un problème NP-Complet par réduction du problème d'ordonnancement sous contraintes de précédence, décrit par [GARE 79], page 239. Nous démontrerons ensuite que le problème d'ordonnancement avec assignation d'opérateurs (OCD_ASSIGN définition I.24) est aussi un problème NP-Complet par réduction du problème de "Job-Shop".

Le problème d'ordonnancement sous contraintes de précédence est de savoir s'il existe un ordonnancement sur m processeurs respectant une date de fin d'exécution et des contraintes de précédence. C'est un problème NP-Complet [GARE 79]. Ce problème se définit de la manière suivante:

Soient un ensemble de tâches T toutes de longueur 1, un ordre partiel $\succcurlyeq$ sur T, un nombre m de processeurs et une date limite de fin L. Existe-t-il un ordonnancement Ord de T vers {1, 2, ..., L} tel que :

4) $\forall\ i \in \{1, 2, ..., L\}, \| \{t \in T / Ord(t)=i\} \| \leq m$

5) $\forall\ t_1$ et $t_2 \in T / t_1 \succcurlyeq t_2$, $Ord(t_1) > Ord(t_2)$

Nous allons montrer que toute instance du problème ci-dessus se transforme en temps polynomial en une instance du problème OCD_SELEC. En effet, définissons une fonction $SELEC_{op}(t)$ égale à 1 pour toutes les tâches de T, la contrainte 3) de OCD_SELEC est donc vérifiée et la contrainte 4) peut se noter:

$\forall\ i \in \{1, 2, ..., L\}, \| \{t \in T / Ord(t)=i$ et $SELEC_{op}(t)=1\} \| \leq m$

Etant donné que Ord(t) est un entier, Ord(t)=i peut aussi se noter $Ord(t) \leq i < Ord(t)+1$

Soit maintenant Surf une constante égale à 1, alors 4) est équivalent à:

$\forall\ i \in \{1, 2, ..., L\}$, $Surf * \| \{t \in T / Ord(t) \leq i < Ord(t)+1 \text{ et } SELEC_{op}(t)=1\} \| \leq m$

Soit une fonction f, définie par:

$f(t,i) = Surf * \| \{t \in T / Ord(t) \leq i < Ord(t)+1 \text{ et } SELEC_{op}(t)=1\} \|$

f(t,i) est inférieure à m pour tout $i \in \{1, 2, ..., L\}$ si et seulement si le maximum sur l'ensemble {1, 2, ..., L} de f(t,i) est inférieur à m. Par conséquent, la contrainte 4) précédente est équivalente à 4'):

4') $Surf * Max\ i \in \{1, 2, ..., L\} \{ \| \{t / Ord(t) \leq i \leq Ord(t)+1\} \text{ et } SELEC_{op}(t)=1 \| \} \leq m$

Les contraintes 4'), 5) et 6) définissent une instance du problème OCD_SELEC dans le cas particulier d'un seul opérateur de surface 1 et de durée 1. Toutes les instances du problème d'ordonnancement sous contraintes de précédence sont donc transformables en temps polynomial en une instance d'OCD_SELEC. Le problème OCD_SELEC est donc un problème NP-Complet.

La démonstration de la complexité du problème OCD_ASSIGN est réalisée par transformation polynomiale du problème de "Job-Shop", tel qu'il est défini dans [GARE 79]. Cette définition est la suivante:

Soient m processeurs, numérotés de 1 à m, et J un ensemble de travaux; soient les contraintes suivantes:

6) $\forall\ j \in J$, un ordre sur les tâches $t_k[j]$ où $1 \leq k \leq n_j$

7) Soit op(t) le processeur affecté à t, $\forall\ j \in J$ et $\forall\ k / 1 \leq k < n_j$, $op(t_k[j]) \neq op(t_{k+1}[j])$

Soit l(t) la longueur de chaque tâche t, alors existe-t-il un "Job-Shop" de J respectant une date limite de fin L?

Ceci peut aussi se dire: existe-t-il une collection d'ordonnancements sur un processeur σ_i, chacun d'eux affectant les tâches t telles que op(t)=i pour tout i tel que $1 \leq i \leq m$, vérifiant les contraintes suivantes:

8) $\forall\ i$, si $\sigma_i(t) > \sigma_i(t')$ alors $\sigma_i(t) \geq \sigma_i(t')+l(t')$

9) $\forall\ j \in J$ et $\forall\ k / 1 \leq k < n_j$, $\sigma(t_{k+1}[j]) \geq \sigma(t_k[j])+l(t_k[j])$

10) $\forall\ j \in J$, $\sigma(t_{nj}[j])+l(t_{nj}[j]) \leq L$

Pour la transformation du problème de "Job-Shop", l'ordre sur les tâches est représenté par la relation d'ordre partiel $\succcurlyeq$ sur l'ensemble des travaux. La contrainte 6) s'exprime de manière équivalente par la contrainte 6') suivante:

6') $\forall\ j \in J$, $t_{k+1}[j] \succcurlyeq t_k[j]$ où $1 \leq k < n_j$

Soient m fonctions $ASSIGN_{ins}$ définies par:

$\forall\ ins \in \{1, ..., m\}$, $ASSIGN_{ins}(t_k) = 1$ si $op(t_k)=ins$ et 0 sinon.

Par conséquent, la contrainte 7) s'exprime de manière équivalente par la contrainte 7') suivante:

7') $\sum_{k=1}^{nj} ASSIGN_{ins}(t_k) = 1.$

Soit un ordonnancement Ord défini de T (l'ensemble des tâches pour l'ensemble des travaux) dans {1,... , L} par Ord(t) = σ(t) i parcourant tous les processeurs. Les contraintes 8) et 9) s'expriment respectivement par 8') et 9') suivantes:

8') ∀ ins ∈ {1, ..., m} si Ord(t)>Ord(t') et $ASSIGN_{ins}(t)=ASSIGN_{ins}(t')=1$ alors Ord(t)≥Ord(t')+l(t')

9') ∀ t_1 et $t_2 \in T$, si $t_1 \gg t_2$ alors $Ord(t_1) > Ord(t_2)$

La contrainte 8') peut aussi se noter:

8') ∀ ins∈ {1, ..., m} si Ord(t') <Ord(t) ≤Ord(t')+l(t') alors soit $ASSIGN_{ins}(t)\neq 1$ soit $ASSIGN_{ins}(t')\neq 1$

ou encore:

8') ∀ ins∈ {1, ..., m} si Ord(t') ≤Ord(t)+1 <Ord(t')+l(t') alors ||{t"∈ {t, t'}/ $ASSIGN_{ins}(t")=1$}|| ≤ 1

La contrainte 9') est la contrainte de précédence 2) de l'ordonnancement OCD_ASSIGN. La contrainte 7') est la contrainte 3') de OCD_ASSIGN. La contrainte 6') définit un ordre partiel particulier sur les tâches. La contrainte 8') est équivalente à la contrainte 1') de OCD_ASSIGN. La contrainte 10) est respectée par la définition de Ord sur {1, ..., L}.

Par conséquent, le problème de "Job-Shop" se transforme en temps polynomial en une instance du problème OCD_ASSIGN, vérifiant un ordre partiel particulier (représenté par un ensemble de chemins disjoints). Le problème de "Job-Shop" est un problème NP-Complet. Le problème OCD_ASSIGN est donc NP-Complet.

Nous avons démontré que le problème d'ordonnancement est un problème NP-Complet quelle que soit la classe de problèmes abordée. Des restrictions de ce problème se résolvent en temps polynomial; en voici quelques exemples.

Dans le cas de m processeurs identiques, si le graphe G est un arbre rentrant (ou une forêt d'arbres rentrants), [HU 61] donne un algorithme polynomial de résolution exacte de l'ordonnancement. Le cas d'un arbre rentrant, pour la représentation des contraintes de précédence, se traduit, en synthèse, par le fait que chaque donnée d'une description de haut niveau n'est utilisée que par une seule opération élémentaire. Le fait d'avoir m processeurs identiques se traduit par le fait d'avoir un seul type de ressource. Ces deux cas réunis sur une même application ne sont pas envisageables.

Lawer [LAWE 76] propose une résolution exacte en $0(n^2)$ du problème d'ordonnancement, où n est le nombre d'opérations, dans le cas d'un seul processeur. Cela correspond à l'ordonnancement avec une seule instance de ressource, dans le cas de la synthèse de haut niveau, ou à l'ordonnancement de plusieurs boucles sur la même partie opérative.

OCD_ASSIGN est un problème d'ordonnancement à contraintes disjonctives. [ROY 66] propose une approche de ce problème en ajoutant des paires de disjonctions pour chaque contrainte disjonctive. Ces paires sont des arcs qui viennent s'ajouter aux arcs modélisant les contraintes de précédence. Pour deux machines, l'algorithme de Johnson, repris par [FAUR 76], s'applique. Cet algorithme

résout le problème d'ordonnancement de manière optimale en temps polynomial. Le problème d'ordonnancement OCD_ASSIGN avec seulement deux instances de ressources n'est pas un problème réaliste en pratique. Cependant, ce type d'algorithme pourrait être applicable pour l'ordonnancement de boucles élémentaires sur deux PO distinctes.

La préemption (interruption d'une tâche par une autre tâche) autorise souvent une résolution polynomiale des problèmes d'ordonnancement [BLAZ 91] ou [GARE 79]. La préemption en synthèse de haut niveau serait applicable dans le cas de partitionnement. Le partitionnement est soit réalisé pour minimiser le nombre d'éléments de mémorisation [DEPU 90a, 90b], soit issu de la résolution de l'ordonnancement par une méthode exacte (prohibitive en temps de calcul) qui ne permet pas de traiter plus d'un nombre fixé d'opérations, telle que la résolution par programmation linéaire en nombres entiers [HWAN 91]. Pour le partitionnement, l'ordonnancement est décomposé en deux phases: une première ordonnance l'intérieur des sous-graphes, une deuxième ordonnance les sous-graphes les uns par rapport aux autres, en considérant chaque sous-graphe comme une macro-opération. Dans la deuxième phase d'ordonnancement, il est possible d'interrompre l'exécution d'une tâche, en retardant la fin de son exécution, puisqu'une tâche est en fait une suite d'opérations. Un algorithme de résolution exacte intéressant, en $0(n^2)$, développé dans [BLAZ 87], page 24, serait applicable pour l'ordonnancement de graphes hiérarchisés ne comportant pas de boucle.

En conclusion, notons que la recherche exhaustive d'une solution d'OCM a une complexité de $0(n^{n+2})$ [PARK 88]. Nous avons listé certains cas très particuliers d'algorithmes polynomiaux de résolution exacte du problème d'ordonnancement. Cependant, nous avons montré que ce problème est, en général, un problème NP-Complet. Nous allons donc présenter une heuristique de résolution du problème d'ordonnancement dans le cas général, par une méthode orientée par les forces.

I.4. L'ordonnancement Orienté Par les Forces

L'ordonnancement étant un problème NP-Complet, une heuristique de résolution du problème OCT_SELEC, qui définit simultanément le nombre d'opérateurs, est proposée. C'est l'*ordonnancement Orienté Par les Forces (OPF).*

L'OPF a initialement été défini par Paulin et Knigh [PAUL 86], [PAUL 87], [PAUL 88], [PAUL 89a, 89b et 89c], [PAUL 90]. C'est une méthode constructive résolvant l'ordonnancement sous contrainte de temps (OCT) à partir du graphe de précédence (GP définition I.14) d'une boucle élémentaire, les opérations élémentaires du GP étant associées à des opérateurs existant en bibliothèque.

Après présentation de l'algorithme de base, nous en proposerons des améliorations. En effet, deux points critiques de cet algorithme sont identifiés et contournés en section I.4.4: une meilleure anticipation des prochains pas de l'algorithme et une amélioration permettant d'intégrer les critères spécifiques dus à des cas pratiques d'implantation de la partie opérative.

Pour alléger le texte, "OPF" est parfois employé dans le sens de "ordonnancement OPF".

I.4.1. Algorithme de base

Dans un premier temps, nous supposerons que le GP et le GFC de la boucle élémentaire ne comportent pas de branchement conditionnel et que les opérations sont monocycles. Des extensions seront présentées en section I.4.2.

Pour illustrer la description de la méthode d'ordonnancement, nous utiliserons l'exemple dit *de l'équation différentielle* [PAUL 86], qui permet de résoudre l'équation différentielle de la figure I.7(a) et dont la spécification initiale est celle de la figure I.7(b). Le GP du corps de la boucle de la spécification initiale est donné figure I.8 et le GFC est donné figure I.9.

Cette méthode résout le problème OCT. L'utilisateur doit lui-même fixer la contrainte de temps. La limite inférieure de cette quantité est la longueur du chemin critique, qui est la longueur minimale nécessaire, en pas de contrôle, pour ordonnancer toutes les opérations.

La première phase de l'algorithme va consister à définir l' *intervalle de positionnement* de chaque opération, c'est-à-dire l'intervalle dans lequel l'opération peut être assignée sans contrevenir aux contraintes de dépendance de données.

(1) x=x0
(2) y=y0
(3) u=u0

(4) t1 = u* dx
(5) t2 = 3*x
(6) x = x+dx
(7) t3 = [t1]*[t2]
(8) t4 = u - [t3]
(9) t5 = 3*y
(10) t6 = [t5]*dx
(11) t8 = u*dx
(12) u = [t4]-[t6]
(13) y = y+[t8]
Si
(14) x≤a
alors répéter de (4) à (14)
sinon sortie

$y'' + 3xy' + 3y = 0$

(a) Equation différentielle **(b) spécification initiale**

Figure I.7: Équation différentielle

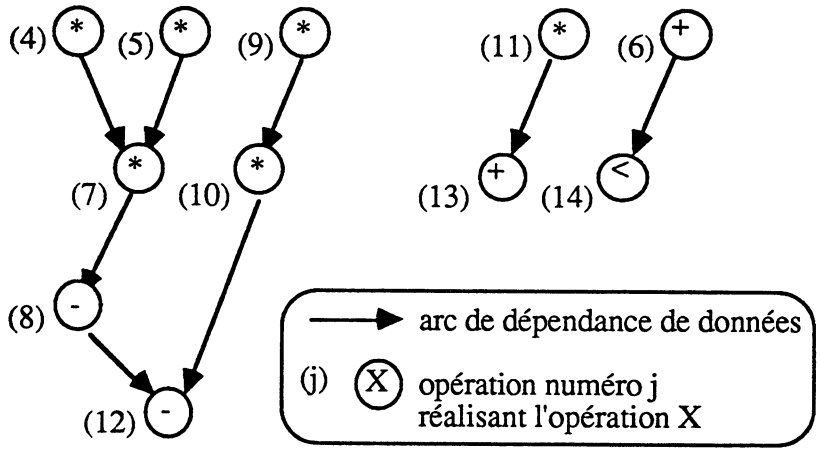


Figure I.8: GP du corps de la boucle de la figure I.7(b)

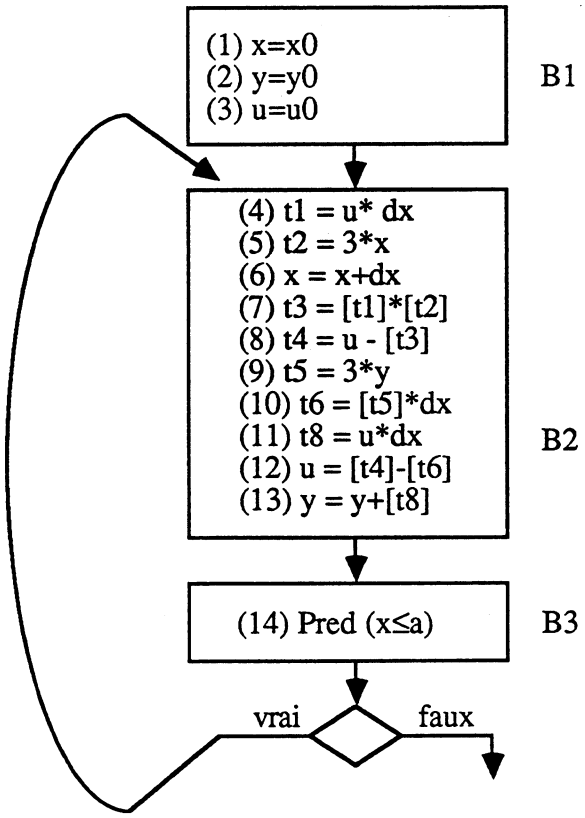


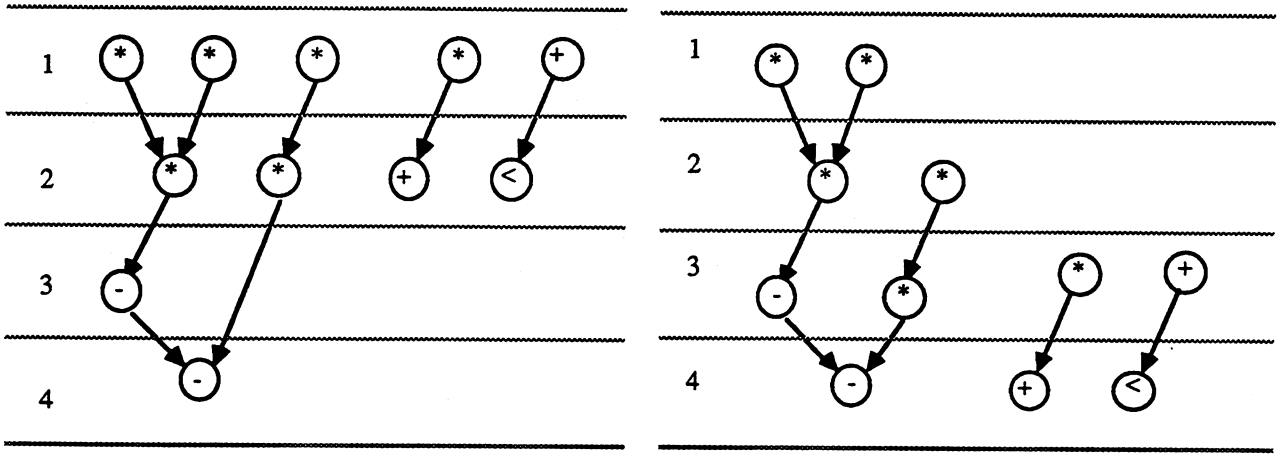
Figure I.9: GFC de la spécification initiale de la figure I.7(b)

Définition I.25: Une opération est *prête* dès que tous ses prédécesseurs sont ordonnancés.

L'ordonnancement Au Plus Tôt (*APTO*) ordonnance toutes les opérations sans prédécesseur dans le premier pas de contrôle. S'il reste des opérations à ordonnancer, l'ordonnancement APTO passe au pas de contrôle suivant en lui assignant toutes les opérations prêtes, et ainsi de suite jusqu'à ce que toutes les opérations soient ordonnancées. Un ordonnancement Au Plus TArd (*APTA*) est défini de manière similaire. Dans ce cas, il faut connaître la date limite de fin d'ordonnancement; les opérations sont ordonnancées en commençant par le dernier pas de contrôle.

Définition I.26: l'*intervalle de positionnement* d'une opération est compris entre les ordonnancements APTO et APTA de l'opération.

La figure I.10 représente ces deux ordonnancements pour le GP de la figure I.8, en prenant comme contrainte de temps, pour l'ordonnancement APTA, la longueur du chemin critique qui est égale à 4.



(a) APTO (b) APTA

Figure I.10: Ordonnancements APTO et APTA pour le GP de la figure I.8

A chaque opération t, est associée la probabilité Prob(t,p) d'être ordonnancée à un pas de contrôle donné p de son intervalle de positionnement. La loi de cette probabilité est supposée uniforme. Cette probabilité est donc égale à l'inverse de la longueur de l'intervalle de positionnement de l'opération. Les probabilités des opérations de la figure I.8 sont illustrées figure I.11.

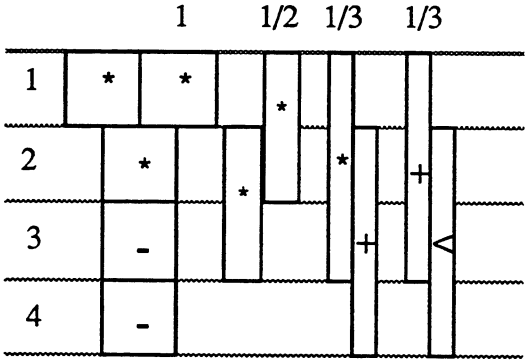


Figure I.11: Probabilités des opérations de la figure I.8

Ces probabilités permettent ensuite de définir le nombre moyen d'opérations devant s'exécuter sur un opérateur déterminé à chaque pas de contrôle.

Définition I.27: A chaque pas de contrôle p, le *Graphe de Distribution (GD)* de l'opérateur op est défini par:

$$GD_{op}(p) = \sum_{t/SELEC_{op}(t)=1} Prob(t,p)$$

La figure I.12 donne le graphe de distribution associé au multiplieur "*" et à l'additionneur "+", pour l'exemple exposé figure I.8.

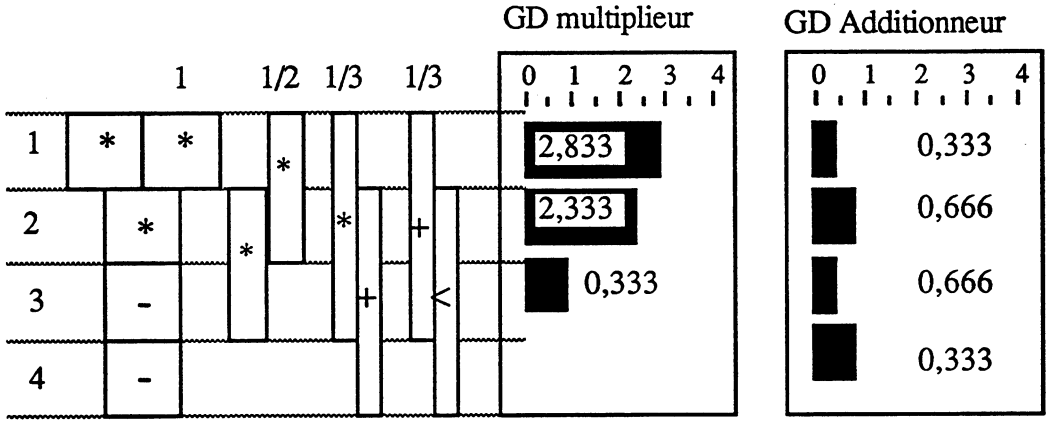


Figure I.12: Graphe de distribution du multiplieur et de l'additionneur

Le but de l'ordonnancement est de minimiser la fonction m_O(Ord) (définition I.21), en la bornant par une valeur maximale sur l'ensemble des pas de contrôle. Il s'agit donc d'équilibrer les graphes de distribution (GD définition I.27) de chaque opérateur par rapport aux pas de contrôle.

Dans ce but, une fonction va mesurer les variations des GD de chaque type d'opérateur (op) pour l'ordonnancement d'une opération à un pas de contrôle donné. Une variation minimisant le nombre d'opérateurs de type op (Nb(op) définition I.21) est une variation qui permet d'équilibrer la valeur des GD sur l'ensemble des pas de contrôle.

Il s'agit donc de diminuer la valeur de GD_{op} dans les pas de contrôle où elle est grande en augmentant sa valeur dans les pas de contrôle où celle-ci est petite. Une fonction F_t mesure la variation de la distribution sur l'ensemble des pas de contrôle pondérée par la valeur de GD_{op}, si l'opération t est assignée à un pas de contrôle p.

Définition I.28: La *fonction propre* $F_t(t,p)$, qui représente le coût de l'assignation d'une opération t à un pas de contrôle p, est définie par:

$$F_t(t,p) = \sum_{p'=APTO(t)}^{p'=APTA(t)} GD_{op(t)}(p') * x_t(p',t,p),$$

où op(t) est l'opérateur op tel que $SELEC_{op}(t)=1$,

$x_t(p',t,p)$ est la variation de probabilité au pas de contrôle p', due à l'assignation de t au pas de contrôle p.

Remarque: En représentant les graphes de distribution par des ressorts, la fonction F_t représente la somme des forces de rappel des ressorts sur l'ensemble des pas de contrôle. Le fait d'ordonnancer une opération à un pas de contrôle p va comprimer le ressort correspondant à $GD_{op}(p)$ et au contraire décomprimer les autres ressorts.

Cette fonction ne reflète que l'assignation d'une opération à un pas de contrôle. Or, cette assignation fait varier les intervalles de positionnement des opérations, liées à l'opération ordonnancée par une relation de précédence. Cette fonction est donc additionnée aux fonctions précédentes F_{Prec} et suivantes F_{Suiv}, définies comme suit:

$$F_{Prec}(t,p) = \sum_{\{Prec(t)\}} F_{Prec(t)}(t,p),$$ où {Prec(t)} représente l'ensemble des prédécesseurs de t;

où $F_{Prec(t)}(t,p)$ représente l'effet de l'assignation de t à p sur l'opération précédente de t Prec(t) et est

définie de façon similaire à F_t par:

$$F_{Prec(t)}(t,p)= \sum_{p'=APTO(Prec(t))}^{p'=APTA(Prec(t))} GD_{op(Prec(t))}(p') * x_{Prec(t)}(p',t,p),$$

où $x_{Prec(t)}(p',t,p)$ est la variation de probabilité au pas de contrôle p', due à l'effet sur Prec(t) de l'assignation de t au pas de contrôle p.

Le même principe est appliqué pour le calcul de F_{Suiv}.

La fonction totale reflète l'influence de l'assignation de t au pas de contrôle p sur l'ensemble des pas de contrôle. Elle est définie par la somme de la fonction propre, de la fonction précédente et de la fonction suivante.

Définition I.29: la *fonction totale* F_{Tot} due à l'ordonnancement de t à p est définie par:

$$F_{Tot}(t,p) = \sum_{to \in T} F_{to}(t,p),$$

où F_{to} est définie par :

$$F_{to}(t,p) = \sum_{p'=APTO(t_o)}^{p'=APTA(t_o)} GD_{op(to)}(p') * x_{to}(p',t,p),$$

où $x_{to}(p',t,p)$ est la variation de probabilité au pas de contrôle p', due à l'effet sur t_o de l'assignation de t au pas de contrôle p.

L'algorithme d'ordonnancement est basé sur le calcul des fonctions totales (F_{Tot} définition I.29). Il commence par calculer les intervalles de positionnement et les graphes de distribution. Puis, pour chaque opération du Graphe de précédence (GP définition I.14), la fonction F_{Tot} est calculée pour toutes les assignations possibles de son intervalle de positionnement. L'assignation qui minimise la fonction F_{Tot} est sélectionnée. L'opération correspondante est assignée au pas de contrôle choisi. Les intervalles de positionnement et les graphes de distribution sont mis à jour et l'algorithme essaye, à nouveau, toutes les assignations possibles pour toutes les opérations non encore ordonnancées. Cet algorithme est résumé dans la figure I.13.

```
Tant qu'il reste des opérations non encore assignées faire
   Pour Chaque opération non encore assignée t faire
      Pour Chaque pas de contrôle p de son intervalle de positionnement faire
         Calculer Ft(t,p)
         Pour Chaque prédécesseur to de t faire
            Calculer Fto(t,p)
         FinPour
         Pour Chaque successeur to de t faire
            Calculer Fto(t,p)
         FinPour
      FinPour
      FTot (t,p) = Σ Fto
   FinPour
   Ordonnancer l'opération t au pas de contrôle p qui minimise FTot(t,p)
   Màj des GD
FinTantQue
```

Figure I.13: Algorithme d'ordonnancement OPF

I.4.2. Extensions

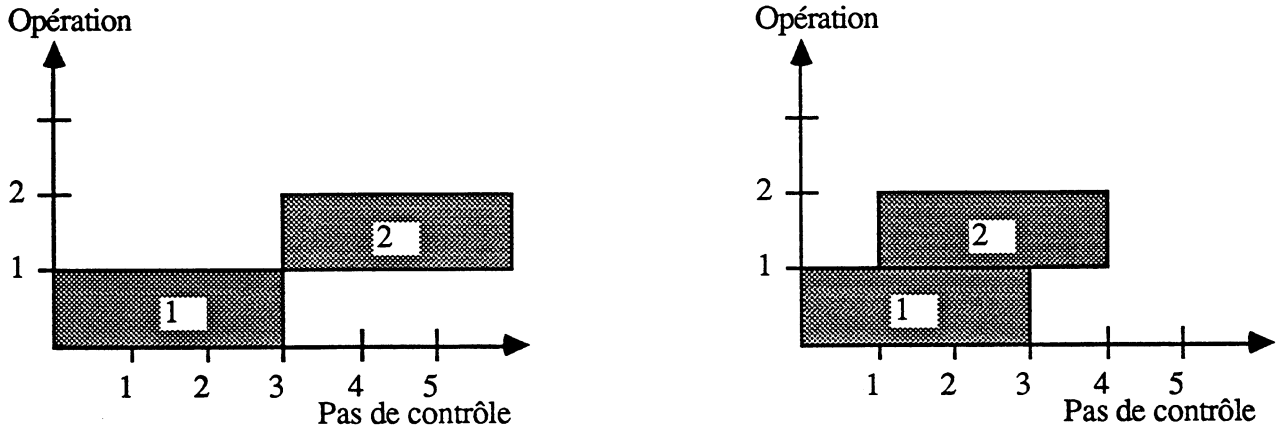
L'algorithme de base a été défini pour le GP d'une boucle élémentaire sans branchement conditionnel et avec des opérations utilisant des opérateurs monocycles. Cet algorithme est étendu aux opérateurs multicycles et pipelinés, aux opérations chaînées et aux cas de branchements conditionnels.

I.4.2.1. Opérateurs multicycles et pipelinés

Les opérateurs multicycles sont parfois des *opérateurs pipelinés*. Dans ce cas, après un temps dit de *latence*, l'opérateur est libre pour une nouvelle opération, bien que la première opération ne soit pas totalement exécutée. Le résultat de la première opération ne sera valide qu'après le délai nécessaire à son exécution. Ce délai est plus grand que le temps de latence. Les délais de l'opération et de latence d'une opération pipelinée (en nombre de pas de contrôle) peuvent être calculés en utilisant la définition I.1.16.

La figure I.14 (a) représente le temps d'occupation d'un opérateur multicycle simple et la figure I.14(b), celui d'un opérateur multicycle pipeliné, avec un délai d'opération de 3 pas de contrôle et une latence de 1 pas de contrôle. Pour simplifier, un opérateur multicycle simple est appelé opérateur multicycle tandis qu'un opérateur multicycle pipeliné est appelé opérateur pipeliné.

Par abus de langage, une opération dont l'opérateur est multicycle est appelée opération multicycle.



(a) Opérateur multicycle simple **(b) Opérateur pipeliné**

Figure I.14: Occupations d'opérateurs multicycles

Les modifications apportées à l'OPF sont de deux ordres: la modification du calcul des intervalles de positionnement des opérations et l'extension du calcul des probabilités.

Le calcul des intervalles de positionnement des opérations doit tenir compte de la longueur des opérations. Lors de l'ordonnancement APTO (respectivement APTA), une opération ne sera prête qu'après que toutes ses opérations précédentes (respectivement suivantes) auront été ordonnancées et qu'un délai supplémentaire valant le maximum des délais de ses opérations précédentes (respectivement suivantes) se sera écoulé.

Le calcul des probabilités est affecté par le délai de l'opération multicycle. Une opération multicycle, d'une durée de n pas de contrôle, peut alors être considérée comme n opérations d'une durée de 1 pas de contrôle. La probabilité de l'opération multicycle sera le cumul des probabilités des n opérations

monocycles. La figure I.15 illustre un calcul pour une opération définie sur 3 cycles avec un intervalle de positionnement de [1,3]. La probabilité est alors définie par étage: la probabilité 1/3 de trouver un étage de l'opération au pas de contrôle 1 est de 1/3, de 2/3 au pas de contrôle 2, etc. ...

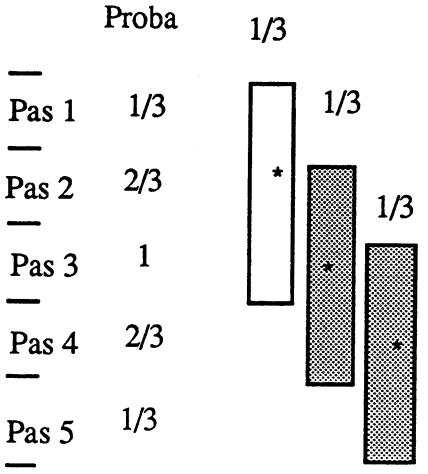


Figure I.15 : Calcul des probabilités pour une opération multicycle

Les opérations pipelinées sont ensuite prises en compte avec le même raisonnement. Le calcul des intervalles de positionnement prend en compte le délai de l'opérateur et le calcul des probabilités prend en compte le délai de latence.

I.4.2.2. Opérations chaînées

Le *chaînage* d'opérations est employé pour des opérateurs combinatoires. Plusieurs opérations arithmétiques, liées par une dépendance de données, sont exécutées pendant un même pas de contrôle.

Définition I.30: Soient deux opérations $t_1$et t_2, liées par une dépendance de données $t_1 \gg t_2$. $t_1$et t_2 pourront être *chaînées* si et seulement si elles vérifient $d(t_1)+d(t_2) \leq c$, où c est la durée d'un pas de contrôle et d(t) la durée de l'opération t en nano-secondes. Le chaînage de $t_1$et t_2 créera une macro-opération $t_1\_t_2$ de délai $d(t_1)+d(t_2)$.

Remarque: En fait, le délai de plusieurs opérations chaînées est souvent inférieur à la somme des délais des opérations. En effet, le chemin critique de plusieurs blocs combinatoires mis bout à bout n'a pas de raison d'être la suite des chemins critiques de chaque bloc. A moins de disposer d'une base de données contenant le délai de chaque type de chaînage, nous ferons l'hypothèse que les chemins critiques des opérations chaînées sont mis bout à bout. Le délai d'opérations chaînées est la somme des délais de chaque opération.

Tous les chaînages ne sont pas envisageables. Ainsi, dans la définition I.30, les cas de chaînages avec une opération multicycle et les cas de chaînages chevauchant la limite d'un pas de contrôle ont délibérément été écartés. Le chaînage d'opérations créera toujours une macro-opération monocycle.

Le chaînage d'opérations crée des macro-opérations qui sont réalisées sur de nouveaux opérateurs virtuels, appelés unités exécutives. Par exemple, à partir du GP de la figure I.16(a) (si on suppose que les arcs correspondent à des relations de dépendance de données), l'OPF réalise l'ordonnancement de la figure I.16(b). Cet ordonnancement crée donc une macro-opération de

décalage_addition et un opérateur virtuel qui peut être celui de la figure I.16(c). Cette unité exécutive est capable de réaliser soit le décalage_addition, soit le décalage, soit l'addition.

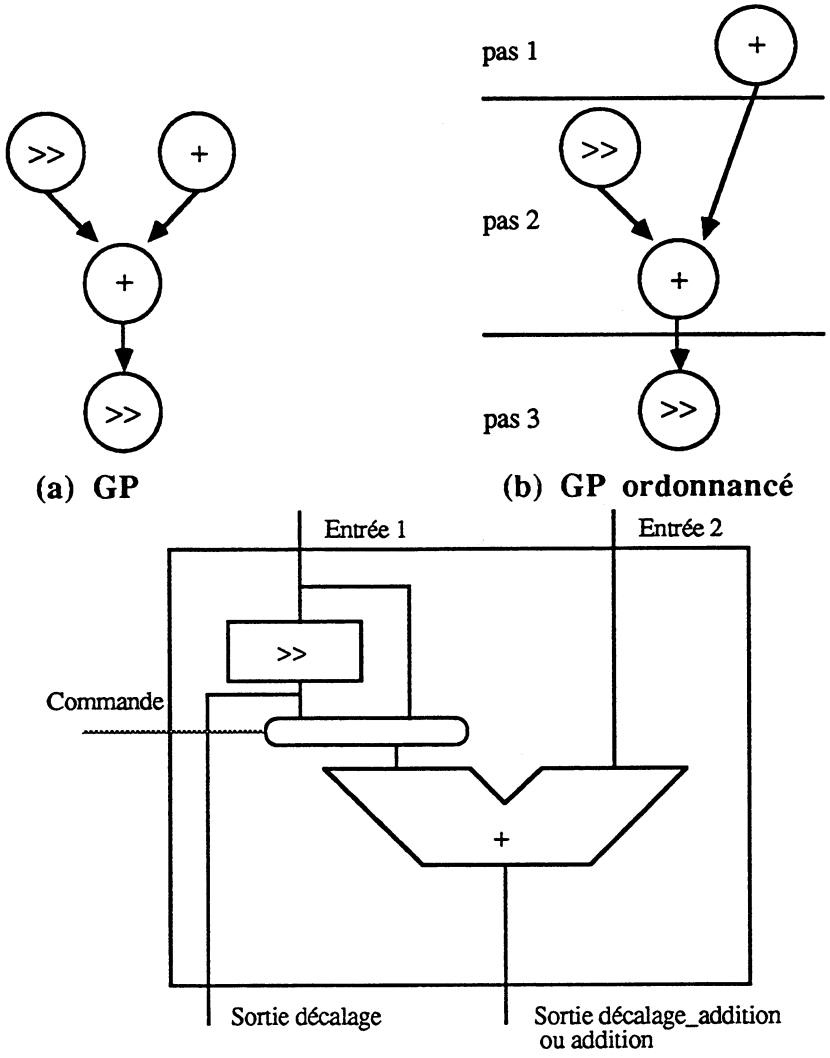


Figure I.16: Exemple de chaînage

Deux opérations chaînées doivent, bien entendu, utiliser des opérateurs différents. De ce fait, le calcul des probabilités n'est pas changé. Le chaînage n'intervient que dans le calcul des intervalles de positionnement de chaque opération. Deux étapes de l'algorithme sont modifiées: tout d'abord le calcul initial des ordonnancements APTO et APTA, puis, après assignation d'une opération à un pas de contrôle, les transformations sur intervalles de positionnement des opérations suivantes et précédentes.

Deux paramètres seront affectés à chaque opération: la durée nécessaire à son exécution et le temps restant dans le pas de contrôle après son ordonnancement. Lors du calcul de l'ordonnancement APTO (respectivement APTA), le passage au pas de contrôle suivant (respectivement précédent) n'est réalisé qu'après avoir vérifié qu'il ne reste pas assez de temps dans le pas de contrôle courant pour chaîner une opération devenue prête.

I.4.2.3. Branchements conditionnels

Dans le cas de branchements conditionnels, le GFC permet de déterminer si deux opérations sont mutuellement exclusives. Deux opérations mutuellement exclusives pourront partager les mêmes ressources: cela n'affecte pas le calcul des intervalles de positionnement des opérations, ni le calcul des probabilités. En revanche, cela affecte le calcul des graphes de distribution. La valeur du graphe de distribution d'une opération, d'un type donné, à un pas de contrôle donné, est égale au maximum des probabilités des opérations mutuellement exclusives (additionné des probabilités des opérations pouvant s'exécuter simultanément).
Nous remarquons que, en réalité, le calcul des forces précédentes et suivantes devrait également refléter la variation mutuellement exclusive en ne prenant pas la somme mais le maximum des forces dans le cas de branchement conditionnel.

I.4.3. Complexité de l'algorithme

La complexité de l'algorithme d'ordonnancement OPF, tel qu'il a été présenté initialement [PAUL 86 et 87] et précédemment dans cette thèse, est $0(m^2.n^3)$, où m est le nombre de pas de contrôle de la contrainte de temps globale et n est le nombre d'opérations.
Une autre formulation de cette méthode d'ordonnancement permet de réduire cette complexité à une complexité $0(m.n^2)$ [PAUL 88].
Cette formulation fait intervenir un nouveau calcul des fonctions propres, qui est plus rapide que celui de la définition I.28. Les nouvelles fonctions propres sont définies de la manière suivante:

Définition I.31: La *fonction propre* de la réduction de l'intervalle de positionnement d'une opération t à l'intervalle $[n_o, n_a]$ est F", définie par:

$$F'' = \sum_{p'=n_o}^{p'=n_a} GD_{op(t)}(p') * 1/n_h + \sum_{p'=APTO(t)}^{p'=APTA(t)} DG_{op(t)}(p') * 1/h,$$

où 1/h est la longueur de l'intervalle de positionnement initial de t (APTA(t) - APTO(t) -1) et n_h est la nouvelle longueur de l'intervalle de positionnement, après réduction, (n_a-n_o-1).

Ce nouveau calcul des fonctions propres permet déjà de réduire le temps de calcul effectif (temps cpu) du calcul des fonctions propres. En effet, la variation de probabilité moyenne ($2^{ème}$ terme du calcul de F" ci-dessus) peut être calculée, une fois pour toutes, au début de la recherche de l'ordonnancement minimisant F_{Tot}.
Pour permettre une réduction de la complexité de l'OPF, une nouvelle formulation de l'algorithme est proposée. Dans cette formulation, une première phase du calcul des fonctions totales calcule les fonctions propres de chaque opération pour:
- une affectation de cette opération à un pas de contrôle déterminé (et construit un vecteur V_1 associé à cette opération)
- une réduction par le bas de l'intervalle de positionnement de l'opération (et construit un vecteur V_2 de taille h -1)
- une réduction par le haut de l'intervalle de positionnement de l'opération (et construit un vecteur V_3 de taille h -1).

Cette première phase a une complexité de 0(m.n).

Une deuxième phase doit permettre de calculer les fonctions suivantes et précédentes pour chaque ordonnancement possible; deux solutions sont alors envisageables:

- la première consiste à calculer les fonctions totales (en mettant à jour successivement V1), sur chaque opération et chaque position possible de son intervalle de positionnement. Le calcul des fonctions suivantes (ou précédentes) est effectué par accumulation successive, sur tous les successeurs de l'opération traitée, des valeurs déjà calculées et enregistrées dans V_2 et V_3. Cette première solution permet de réduire la complexité de la phase de calcul des fonctions suivantes et précédentes à $0(m.n^2)$. Dans la formulation habituelle, cette phase a une complexité de $0(m^2.n^2)$.
- la deuxième consiste à calculer les fonctions suivantes et précédentes par accumulation successive des valeurs déjà calculées de V_1, V_2 ou V_3 (et mise à jour successive de V_1). Cette accumulation est réalisée en ne considérant que les successeurs directs, en traversant le graphe GP de bas en haut pour le calcul des forces suivantes et en procédant de haut en bas pour les fonctions précédentes. Cette deuxième solution permet de réduire la complexité de la phase de calcul des fonctions totales à 0(m.n). L'algorithme d'ordonnancement OPF a donc, grâce à cette formulation, une complexité de $0(m.n^2)$.

Leon Stok fait remarquer, dans [STOK 91], que la deuxième solution pour la reformulation de l'algorithme d'ordonnancement OPF peut être erronée dans le cas de chemins reconvergents. En effet, deux chemins reconvergents ont une racine et une destination communes. Après le calcul par accumulation en parcourant le graphe de haut en bas (par exemple), la fonction totale du nœud destination intégrera deux fois le calcul du sommet racine. Certaines fonctions sont calculées plusieurs fois pour les opérations ayant une racine commune.

Dans [PAUL 92], une solution pour contourner ce problème est proposée. Les chemins reconvergents sont recherchés avant le début de l'ordonnancement. Ces chemins permettent de définir une liste de situations de reconvergence formées de paires de sommets du GP. Pour chaque situation de reconvergence différente, un chemin fictif ne contenant qu'un seul sommet fictif est inséré, dans le GP, entre les deux sommets de la situation de reconvergence. A ce sommet fictif est attribuée une fonction fictive valant la valeur de la fonction propre calculée plusieurs fois (celle de la racine commune des chemins reconvergents). Lors de la mise à jour des fonctions totales (dans V_1) par accumulation successive des fonctions propres des sommets précédents, puis suivants, ces sommets fictifs permettent de rétablir l'erreur commise.

En résumé, l'OPF de base a une complexité de $0(m^2.n^3)$. En calculant, avant l'ordonnancement de chaque opération, la valeur des fonctions propres pour la réduction de l'intervalle de positionnement de chaque opération, cette complexité peut être réduite à $0(m^2.n^2)$ [PAUL 88].

Si, de plus, le calcul des fonctions totales est effectué par accumulation successive des fonctions propres, en parcourant le graphe de précédence de bas en haut puis de haut en bas, la complexité peut alors être réduite à $0(m.n^2)$ [PAUL 92], ceci en prenant en compte la remarque de Stok à propos des chemins reconvergents [STOK 91].

I.4.4. Améliorations

L'algorithme de base, décrit dans les sections précédentes, peut être amélioré. Les améliorations décrites dans cette section permettent d'anticiper sur l'évolution des graphes de distribution, de mieux estimer la surface et de tenir compte de la régularité des chaînages.

I.4.4.1. Amélioration par anticipation

Le calcul des fonctions F_{to} ne prend pas en compte l'évolution future des graphes de distribution. Dans [PAUL 89a], un facteur d'anticipation de l'évolution de ces graphes de distribution, fixé par expérience à environ 1/3 de la variation de probabilité de l'opération, a été proposé. Les fonctions F_{to} avec anticipation sont alors définies de la manière suivante:

Définition I.32: Le calcul des fonctions propres de la définition I.28 est étendu pour prendre en compte une anticipation locale de 0,30 de la manière suivante:

$$F_{to}(t,p) = \sum_{p'=APTO(to)}^{p'=APTA(to)} (GD_{op(to)}(p') + 0.30\ x_{to}(p',t,p)) * x_{to}(p',t,p)$$

L'inconvénient de l'anticipation locale sur la variation des graphes de distribution de la définition I.32 est qu'elle prend en compte l'anticipation, due à la variation de probabilité de chaque opération, de manière locale. Or, l'assignation d'une opération à un pas de contrôle fait varier les probabilités de plusieurs opérations. Les variations dues aux opérations précédentes ou suivantes peuvent annuler la variation due à l'opération t_o. De plus, dans le cas de branchements conditionnels, l'anticipation sur la variation du graphe de distribution, quelle que soit l'opération, n'est pas réaliste. Dans ce cas, le futur graphe de distribution vaudra, en effet, le maximum des changements de toutes les opérations mutuellement exclusives. Pour ces deux raisons, une définition locale de l'anticipation ne semble pas convenir.

Nous avons donc proposé un nouvel algorithme d'ordonnancement qui met à jour les variations de probabilités sur les graphes de distribution, au fur et à mesure du calcul de la fonction totale (F_{Tot} définition I.29). Cet algorithme est résumé dans la figure I.17.

La mise à jour peut également être effectuée avec un facteur de 1/3.

Cet algorithme a la même complexité que l'algorithme original d'ordonnancement par les forces, puisque la mise à jour des graphes de distribution a la même complexité que le calcul d'une fonction F_{to}. En revanche, l'amélioration de la complexité proposée dans [PAUL 92] n'est pas applicable.

Tant qu'il reste des opérations non encore assignées **faire**
  Pour Chaque opération non encore assignée t **faire**
    Pour Chaque pas de contrôle p de son intervalle de positionnement **faire**
      Calculer $F_t(t,p)$
      Imputer la variation de probabilité $x_t(p',t,p)$ aux $GD_{op(t)}(p')$
      Pour Chaque prédécesseur t_o de t **faire**
        Calculer $F_{to}(t,p)$
        Imputer la variation de probabilité $x_{to}(p',t,p)$ aux $GD_{op(to)}(p')$
      FinPour
      Pour Chaque successeur t_o de t **faire**
        Calculer $F_{to}(t,p)$
        Imputer la variation de probabilité $x_{to}(p',t,p)$ aux $GD_{op(to)}(p')$
      FinPour
    Remettre les GD dans leur état initial
    FinPour
    $F_{Tot} = \sum F_{to}$
  FinPour
  Assigner l'opération qui minimise F_{Tot}
  Màj des GD
FinTantQue

Figure I.17: Algorithme d'ordonnancement OPF avec mise à jour des GD

Pour cet algorithme, l'ordre de traitement des opérations suivantes et précédentes pourrait avoir une influence sur le résultat final. Cependant, nous allons montrer que ce n'est pas le cas.

Définition I.33: Le calcul des fonctions propres de la définition I.28 est étendu pour prendre en compte une anticipation globale par mise à jour de la manière suivante:

Soient t_1, ..., t_y et t_{y+1}, ..., t_n toutes les opérations s'exécutant sur le même opérateur op, dont l'intervalle de positionnement est modifié par l'assignation de t à p.

Supposons que les opérations soient traitées dans l'ordre de leur indice croissant, la fonction propre $F_{ty}(t,p)$ vaut donc:

$$F_{ty}(t,p) = \sum_{p'=APTO(ty)}^{p'=APTA(ty)} (GD_{op}(p') + x_{t1}(p',t,p) + \ldots + x_{ty-1}(p',t,p)) * x_{ty}(p',t,p)$$

Les fonctions $F_{ty}(t,p)$ et $F_{ty+1}(t,p)$ sont donc calculées de la manière suivante:

$$F_{ty}(t,p) = \sum_{p'=APTO(ty)}^{p'=APTA(ty)} (GD_{op}(p') + x_{t1}(p',t,p) + \ldots + x_{ty-1}(p',t,p)) * x_{ty}(p',t,p)$$

$$F_{ty+1}(t,p) = \sum_{p'=APTO(ty+1)}^{p'=APTA(ty+1)} (GD_{op}(p') + x_{t1}(p',t,p) + \ldots + x_{ty-1}(p',t,p) + x_{ty}(p',t,p)) * x_{ty+1}(p',t,p)$$

Si l'ordre de traitement des opérations t_y et t_{y+1} est inversé, les nouvelles fonctions $F'_{ty}(t,p)$ et $F'_{ty+1}(t,p)$, permettant de calculer la nouvelle fonction F'_{Tot}, sont définies par:

$$F'_{ty}(t,p) = \sum_{p'=APTO(ty)}^{p'=APTA(ty)} (GD_{op}(p') + x_{t1}(p',t,p) + \ldots + x_{ty-1}(p',t,p) + x_{ty+1}(p',t,p)) * x_{ty}(p',t,p)$$

$$F'_{ty+1}(t,p) = \sum_{p'=APTO(ty+1)}^{p'=APTA(ty+1)} (GD_{op}(p') + x_{t1}(p',t,p) + ... + x_{ty-1}(p',t,p)) * x_{ty+1}(p',t,p)$$

Pour toutes les opérations s'exécutant sur une autre ressource que op, ce changement d'ordre n'a pas d'influence, puisqu'il n'a pas d'influence sur leur graphe de distribution. Ce changement d'ordre n'a pas non plus d'influence sur toutes les opérations traitées avant t_y et celles traitées après t_{y+1}, puisque celles traitées avant ne prennent en compte aucune des variations de probabilité sur t_y et t_{y+1} et celles traitées après prennent en compte les deux variations de probabilité. Par conséquent, la différence entre F_{Tot} et F'_{Tot} est Diff, définie par:

$$Diff = (F_{ty}(t,p)+F_{ty+1}(t,p)) - (F'_{ty}(t,p)+F'_{ty+1}(t,p))$$

$$Diff = \sum_{p'=Min(APTO(ty),APTO(ty+1))}^{p'=Max(APTA(ty),APTA(ty+1))} (x_{ty}(p',t,p)*x_{ty+1}(p',t,p)) - (x_{ty}(p',t,p)*x_{ty+1}(p',t,p)) = 0$$

Or, tous les ordres sur $t_1, ..., t_n$ peuvent être obtenus par permutations successives, deux à deux, d'opérations se suivant dans un ordre initial. Par conséquent, quel que soit l'ordre de traitement des opérations, la fonction F_{Tot}, définie par l'algorithme de la figure I.17, a toujours la même valeur.

Verhaegh [VERH 91] propose aussi une amélioration de l'anticipation permettant un calcul global de l'anticipation. Cette amélioration consiste à faire la somme de l'ensemble des variations de probabilité sur toutes les opérations dont l'intervalle de positionnement change et dont la ressource allouée est la même. Dans ce cas non plus, l'amélioration de la complexité, proposée dans [PAUL 92], n'est pas applicable.

Définition I.34: Le calcul des fonctions propres de la définition I.28 est étendu pour prendre en compte une anticipation globale de α de la manière suivante:

$$Fto(t,p) = \sum_{p'=APTO(to)}^{p'=APTA(to)} (GD_{op}(to)(p') + \alpha\Delta x_{op}(p') * x_{to}(p',t,p)) * x_{to}(p',t,p)$$

où Δx_{op} est la somme des variations de probabilité des opérations exécutées sur l'opérateur op et vaut:

$$\Delta x_{op}(p') = \sum_{y=1}^{y=n} x_{ty}(p',t,p)$$

La différence entre l'anticipation globale de Verhaegh et l'anticipation globale par mise à jour proposée dans cette thèse est la valeur du coefficient de pondération de l'anticipation α. Dans notre cas, ce coefficient varie selon le nombre d'opérations impliquant une variation du graphe de distribution alors qu'il est fixe pour l'anticipation globale de Verhaegh.

I.4.4.2. Estimation de la surface

Une amélioration proposée dans [PAUL 89b] permet de prendre en compte la surface des opérateurs. En effet, chaque ressource n'a pas la même surface. Par conséquent, le calcul de la fonction totale F_{Tot} peut être pondéré par le poids de chaque type de ressource. La fonction F_{Tot} devient:

$$F_{Tot}(t,p) = \sum_{to \in T} F_{to}(t,p)*Surf(op(t_o)),$$

où $Surf(op(t_o))$ est le coût en surface de l'opérateur $op(t_o)$.
Il est également possible d'estimer le nombre de registres nécessaires, en considérant les intervalles de vie des variables, et d'estimer le nombre de connexions nécessaires.

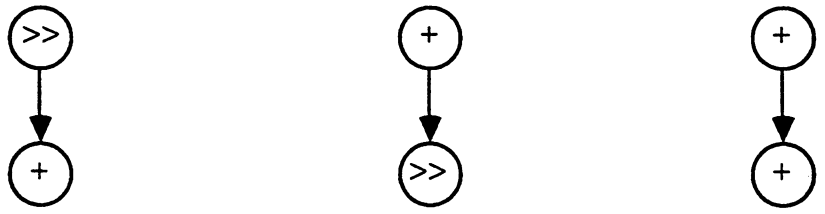
I.4.4.3. Chaînage

Le chaînage construit des macro-opérations implicitement allouées à des unités fonctionnelles complexes, appelées *unités exécutives* [DEMA 86] ou *unités à application spécifique* [NOTE 91]. Ces unités exécutives peuvent éventuellement être instanciées plusieurs fois dans une même partie opérative. Elles doivent donc être optimisées. Pour cela, il faut minimiser le nombre de connexions et le nombre d'opérateurs différents dans une unité exécutive. Il faut donc chercher à obtenir des chaînages identiques ou similaires. L'OPF ne prend pas en compte la régularité des chaînages. Une notion de régularité durant l'ordonnancement des opérations chaînées est donc introduite.
Le calcul d'une fonction F_{to} devient FCh_{to}, qui prend en compte les chaînages réalisés durant les pas précédents de l'algorithme.

Définition I.35: FCh_{to} est définie comme suit:

$$FCh_{to}(t,p) = F_{to}(t,p) + Ch(t_o,t,p)$$

Cette fonction $Ch(t_o,t,p)$ est un coût de chaînage sur l'opération t_o due à l'assignation de t à p. Elle est nulle si l'assignation de t au pas de contrôle p ne crée pas de nouveau chaînage sur l'opération t_o. Dans le cas inverse, la valeur de cette fonction dépend du chaînage ainsi créé.
Dans le cas de création de chaînage, trois éventualités peuvent se produire: dans la première, le chaînage créé fait intervenir des opérateurs déjà chaînés; dans la seconde, ce chaînage fait intervenir des opérateurs déjà chaînés, mais dans un autre sens, comme l'illustre la figure I.18(b) par rapport à la figure I.18(a); dans la troisième, ce chaînage fait intervenir des opérateurs qui n'ont encore jamais été chaînés, comme l'illustre la figure I.18(c) par rapport à la figure I.18(a). Le second cas de chaînage est appelé un chaînage inverse, le troisième un nouveau chaînage.



(a) Chaînage déjà existant (b) Chaînage inverse (c) Nouveau chaînage

Figure I.18: Chaînage existant et nouveaux chaînages

La création d'un chaînage inverse implique une connexion supplémentaire entre deux opérateurs déjà connectés qui forment déjà une unité exécutive. La création d'un nouveau chaînage implique, d'une part, une nouvelle connexion entre deux opérateurs non encore connectés et, d'autre part, soit la création d'une nouvelle unité exécutive (formée des deux opérateurs chaînés), soit l'accroissement

d'un opérateur sur une unité exécutive déjà existante (comprenant déjà l'un des opérateurs impliqués dans le nouveau chaînage). Le coût en connexions d'un nouveau chaînage sera forcément plus élevé que le coût d'un chaînage inverse.

Définition I.36: Soit ß le ratio entre le coût d'un nouveau chaînage et le coût d'un chaînage inverse (ß>1), alors la fonction Ch est définie de la manière suivante:

$Ch(t_o,t,p)$ = Surf(Conn) si le chaînage ainsi créé est un chaînage inverse,

$Ch(t_o,t,p)$ = ß * Surf(Conn) sinon,

où Surf(Conn) est le coût estimé d'une connexion par rapport au coût d'un opérateur.

Dans le premier cas de chaînage, un chaînage déjà existant est réutilisé. Un ordonnancement réalisant ce chaînage, par rapport à un ordonnancement ne le réalisant pas, économisera un registre entre les deux opérateurs chaînés et une connexion si ces opérateurs ne sont utilisés qu'en situation de chaînage. Par conséquent, la définition I.36 du coût d'un chaînage est étendue.

Définition I.37: Le coût d'un chaînage Ch est défini de la manière suivante:

$Ch(t_o,t,p)$ = - Surf(Conn) si le chaînage ainsi créé est un chaînage déjà existant,

$Ch(t_o,t,p)$ = Surf(Conn) si le chaînage ainsi créé est un chaînage inverse,

$Ch(t_o,t,p)$ = ß * Surf(Conn) si le chaînage ainsi créé est un nouveau chaînage,

où Surf(Conn) est le coût estimé d'une connexion par rapport au coût d'un opérateur.

L'algorithme d'ordonnancement OPF est séquentiel. Des décisions prises aux premiers pas de l'algorithme peuvent ne pas être intéressantes de manière globale. Un chaînage créé aux premiers pas de l'algorithme peut très bien ne plus jamais se reproduire. Un coefficient de fréquence probable d'apparition d'un chaîne (Fr) est introduit dans la fonction coût du chaînage.

Définition I.38: Le coût d'un chaînage Ch est définie de la manière suivante:

$Ch(t_o,t,p)$ = - Surf(Conn) si le chaînage ainsi créé est un chaînage déjà existant,

$Ch(t_o,t,p)$ = Surf(Conn) / (1+$Fr(t_o,t,p)$) si le chaînage créé est un chaînage inverse,

$Ch(t_o,t,p)$ = ß*Surf(Conn)/(1+$Fr(t_o,t,p)$) si le chaînage ainsi créé est un nouveau chaînage,

où $Fr(t_o,t,p)$ est la fréquence probable d'apparition du chaînage créé sur t_o par l'assignation de t à p.

Cette fréquence probable est calculée avant l'ordonnancement. Une approximation de la fonction Fr peut être donnée en énumérant tous les chaînages apparaissant lors du premier calcul des ordonnancements APTO et APTA.

Il va de soi que le calcul des fonctions $FCht_o$ se fait en mettant à jour, au fur et à mesure, la liste des chaînages existants, afin que, si l'assignation d'une opération à un pas de contrôle crée plusieurs fois le même chaînage, celui-ci ne soit pas compté plusieurs fois comme un nouveau chaînage.

I.4.4.4. Autres améliorations

Verhaegh [VERH 91] propose deux autres améliorations de l'ordonnancement OPF, l'une permettant

de rendre l'algorithme moins séquentiel, l'autre permettant de mieux optimiser le nombre de ressources.

La première amélioration propose de réduire progressivement les intervalles de positionnement des opérations plutôt que d'assigner les opérations à un pas de contrôle déterminé. Le calcul des fonctions se fait alors uniquement dans deux cas de réduction de l'intervalle de positionnement d'une opération t: d'un pas de contrôle vers le haut ou d'un pas de contrôle vers le bas. La complexité de l'algorithme de base n'est donc pas affectée.

La deuxième amélioration définit des constantes de raideur globales des ressorts. Cela permet de donner plus de poids à des déplacements lointains de la valeur maximale du graphe de distribution par rapport à des déplacements proches de cette valeur. Les constantes de raideur globales, à chaque pas de contrôle, sont définies par l'inverse de la valeur du graphe de distribution, additionnée de la valeur maximale du graphe de distribution sur l'ensemble des pas de contrôle (plus un epsilon pour garantir des divisions différentes de zéro).

Il semblerait, d'après les résultats de [VERH 91], que l'amélioration par constantes de raideur globales permette à elle seule de trouver de meilleurs résultats. Les deux améliorations conjuguées donnent des résultats optimaux sur l'exemple du filtre du 5ème ordre.

Remarque: Ces deux améliorations ne peuvent pas transformer avantageusement le chaînage, puisque les décisions sur le chaînage d'opérations restent locales et séquentielles.

I.5. Autres méthodes d'ordonnancement et étude comparative

Les techniques d'ordonnancement proposées dans la littérature utilisent des heuristiques, puisque les systèmes réalisent la synthèse de haut niveau de circuits intégrés de manière interactive. Il faut donc obtenir une solution rapide à un problème NP-Complet.

I.5.1. Ordonnancement Au Plus TOt

L'heuristique la plus simple est de négliger les critères de ressources et d'appliquer les méthodes exactes basées sur les potentiels [ERSC 79]. Ces méthodes garantissent l'exécution la plus rapide du graphe de dépendance de données, mais aux dépens de la quantité de ressources nécessaires. Ces méthodes résolvent le problème OCT sans allocation de ressources. Ce sont les ordonnancements APTO et APTA définis précédemment (définition section I.4.1).
Le système FACET [TSEN 83] utilise une telle approche.
Etant donné le nombre prohibitif de ressources qu'implique ce type d'ordonnancement, une extension de l'ordonnancement APTO permet de résoudre le problème OCM: c'est l'ordonnancement APTO sous contrainte de ressources. Cette méthode, utilisée dans le système MIMOLA, retarde de manière arbitraire l'ordonnancement de certaines opérations, si le nombre d'opérations prêtes dépasse un seuil donné [MARW 86].
Ces deux types d'algorithmes conduisent, de façon presque automatique, aux approches par liste.

I.5.2. Méthodes par liste

Ce sont des heuristiques dans lesquelles les tâches sont classées par ordre de priorité. Les opérations sont ordonnancées à partir de l'instant 0, en affectant à l'instant t les opérations prêtes, dont la priorité est la plus élevée [LAHR 82].
Différents critères sont utilisés pour retarder l'ordonnancement de certaines opérations. Habituellement, une borne supérieure pour le nombre de ressources disponibles limite le nombre d'opérations qui peuvent être ordonnancées dans un pas de contrôle. Un pas de contrôle supplémentaire est affecté lorsque cette contrainte n'a pas été respectée. Ce type de résolution s'applique donc à la classe des problèmes OCM. Il est utilisé par les systèmes EMUCS [HITC 83], ELF [GIRC 85], SPLICER [PANG 87], BUD [FARL 86a et 86b] et HAL [PAUL 89b].
Différentes fonctions de priorité ont été utilisées pour sélectionner les opérations à retarder en synthèse de haut niveau. Pour le système SPLICER, les ordonnancements APTO et APTA permettent de définir un critère de mobilité pour chaque opération. Le système ELF définit un critère d'urgence qui intègre une notion de mobilité de l'opération et de ces successeurs: c'est le nombre minimal de pas de contrôle requis pour exécuter la portion de GP entre l'opération traitée et la contrainte de temps la plus proche. L'ordonnancement par liste orienté par les forces (force directed list scheduling) du système HAL utilise la fonction force totale, définie précédemment pour l'ordonnancement OPF,

comme critère de priorité.

La fonction de priorité peut aussi être définie par:

- un degré de liberté local (égal à APTA(t) moins le pas de contrôle courant),
- une distance (égale à L(Ord)-APTO(t)),
- une prise en compte du nombre de successeurs,
- une combinaison de plusieurs de ces fonctions de priorité [HEIJ 90].

Une méthode utilisée dans HYPER [POTK 89] se rapproche de l'ordonnancement par liste: c'est l'ordonnancement par noyau. Ici, la fonction de priorité est recalculée après chaque assignation d'une opération à un pas de contrôle. Elle est basée sur deux ratios: un ratio local et un ratio global, mesurant la difficulté de l'ordonnancement pour chaque ressource (nombre de ressources d'un type donné sur le nombre de ressources de ce type nécessaires en moyenne).

D'autres approches sérielles utilisent la notion de degré de liberté. C'est le cas de MAHA [PARK 86], qui ordonnance et assigne d'abord les opérations du chemin critique, puis les autres opérations, en commençant par les opérations de plus petit degré de liberté. Celui-ci est défini par la mobilité de l'opération. Les opérations sont ordonnancées au premier pas de contrôle où il reste une instance de ressource libre pour l'opération.

I.5.3. Programmation en nombres entiers

Des méthodes de programmation en nombres entiers permettent de décrire le problème de façon exacte.

En général, la formulation de l'ordonnancement considère un ensemble de variables x_{tp} , où x_{tp} vaut 1 si l'opération t est ordonnancée dans le pas de contrôle p et 0 sinon. Pour la résolution de problèmes de la classe OCM, un autre ensemble de variables Nb_{op} permet de définir la quantité de ressources de type op autorisées.

Une formulation du problème OCM a été proposée dans [GOOS 87]. Elle permet de prendre en compte les contraintes de ressources et la précédence de données. Dans [GOOS 89a et 89b], la notion de pliage de boucles est aussi prise en compte. Cependant, cette formulation demande un temps de calcul trop important. Pour ce système (CATHEDRAL II), une solution par liste a donc été choisie.

[LEE 89] propose aussi une formulation du problème OCM.

[JANA 90] donne une formulation en deux phases: une première est réalisée sous contrainte de temps et définit le nombre de ressources nécessaires en uniformisant une fonction de distribution du nombre de ressources sur les pas de contrôle; une deuxième réalise l'ordonnancement sous contrainte de ressources. Malheureusement, [JANA 90] ne donne d'information sur les résultats obtenus ni en temps de calcul, ni en performance et en surface.

Une approche hybride est décrite par [PAPA 90]: la sélection des modules utilisés est réalisée en même temps que l'ordonnancement. Chaque module a un poids et l'affectation est modélisée par un programme linéaire en nombres entiers. Cela élargit l'espace d'exploration, le compromis surface-performances étant réalisé en tenant compte de la sélection du type de ressources. Cette approche est particulièrement utile lorsque l'on dispose d'une bibliothèque de modules riche en UAL (Unité Arithmétique et Logique).

En général, la résolution de ces formulations est longue, sachant que le problème général de programmation en nombres entiers est NP-Complet [GARE 79], même dans le cas de variables ne valant que 0 ou 1.

[HWAN 90] décrit une formulation plus avantageuse du problème OCT, en transformant la fonction objective en une fonction linéaire et en limitant l'espace de recherche, en définissant les intervalles de positionnement des opérations avant la formulation. [HWAN 91] propose aussi une formulation des problèmes OCT et OCD. Dans les trois formulations, les branchements conditionnels, les opérations multicycles, les opérations pipelinées, le pliage de boucles (exécution de la fin de la boucle et de son début en parallèle, en prolongeant certaines dépendances de données), le pipeline fonctionnel (pipeline sur le corps d'une boucle), la minimisation des intervalles de vie des variables et la minimisation du nombre de bus sont exprimables. Ce sont donc des formulations complètes permettant de traiter des boucles élémentaires. Ces trois formulations semblent donner des résultats en un temps acceptable pour l'exemple du filtre du $5^{ème}$ ordre qui contient 34 sommets. Cependant, Hwang conseille d'utiliser des heuristiques pour fixer le nombre de ressources et la contrainte de temps globale, avant d'utiliser la formulation de l'OCD, qui est la formulation la plus efficace.

La programmation linéaire en nombres entiers est donc applicable, en limitant la complexité du problème par partitionnement du problème d'ordonnancement initial.

I.5.4. Séparation/évaluation

La technique par séparation et évaluation est utilisée pour résoudre des problèmes de "Job-Shop" [GOND 79] et [CARL 89].

Pour ce qui concerne la synthèse, [GRAS 90] propose un algorithme de sérialisation d'un ordonnancement initial totalement parallèle. L'avantage d'une telle approche est qu'elle permet une exploration exhaustive de l'espace de solutions. De plus, des heuristiques peuvent être facilement dérivées d'une telle approche. Cependant, en pratique, les calculs sont souvent longs et coûteux en place mémoire.

I.5.5. Ordonnancement basé sur les distributions

Les méthodes par distributions se rapprochent des méthodes par lissage utilisées en recherche opérationnelle [FAUR 76]. Ces méthodes apparaissent lorsque l'ordonnancement se fait avec des contraintes cumulatives. L'heuristique est fondée sur l'homogénéité d'une fonction intensité, définie comme étant la quantité de moyens utilisés à un instant donné. Ces méthodes sont applicables à la classe de problèmes OCT.

Ce type d'approche a été utilisé pour la première fois en synthèse par le système HAL [PAUL 86] et [PAUL 87]. La loi de probabilité qui associe une opération à un pas de contrôle est alors considérée comme uniforme. Différentes fonctions d'intensité sont alors définies.

L'ordonnancement orienté par les forces, présenté en section précédente, est une première approche possible.

Une autre approche, utilisée dans CADDY [KRAM 90], consiste à calculer la valeur moyenne du

nombre d'opérations d'un certain type qui sont exécutées dans un pas de contrôle particulier.

I.5.6. Méthodes par transformations successives

Les méthodes par transformations successives partent d'un ordonnancement initial et améliorent cet ordonnancement.

Dans ce domaine, la méthode par redécoupage d'états [CAMP 90b] et le système SCHOLAR [BERG 91c] remettent en cause de manière locale l'ordonnancement d'opérations. Ces méthodes adressent la classe d'ordonnancement au delà des pas de contrôle, en regroupant des pas de contrôle.

Des méthodes plus récentes, comme celle du recuit simulé, ont également été appliquées dans le cadre de la synthèse de haut niveau [DEVA 87], [SAFI 90].

Le recuit simulé est une technique stochastique générale utilisée dans beaucoup de problèmes d'optimisation combinatoire [KIRK 83]. Il repose sur une analogie avec la cristallisation des solides. Le recuit simulé est une méthode d'amélioration successive et itérative. A partir d'une solution initiale, des transformations sont sélectionnées aléatoirement, puis essayées. Les modifications apportées à la disposition courante, lors d'une itération, doivent être de moins en moins importantes au fur et à mesure de l'algorithme. Pour cela, l'évolution de l'algorithme est paramètrée par une fonction température qui est abaissée selon une stratégie donnée. Pour chaque valeur de ce paramètre, suffisamment d'itérations doivent être réalisées, afin de permettre une stabilisation de la fonction coût, intégrant la surface et les performances. Enfin, l'algorithme doit permettre de s'extraire des minima locaux. Ceci est réalisé en acceptant, avec une certaine probabilité, les configurations qui dégradent la fonction coût. Cette probabilité doit décroître avec la baisse de la température. Les transformations essayées sur un ordonnancement initial sont le changement de pas de contrôle d'une opération ou la réallocation de ressources.

Le recuit simulé est applicable à toutes les classes de problèmes d'ordonnancement. Cependant, il est difficile à mettre au point, puisque plusieurs coefficients sont à déterminer (ceux de la fonction coût devant intégrer des critères antinomiques et ceux de la température). De plus, cette méthode est connue pour être une heuristique très longue en temps de calcul. Elle n'est en aucun cas applicable à un système interactif.

I.5.7. Méthode d'ordonnancement Au Plus VIte

Une approche originale du problème d'ordonnancement, pour des spécifications initiales avec des structures de contrôle importantes, est l'ordonnancement Au Plus VIte (APVI), utilisée dans le système Yorktown Silicon Compiler (YSC) [CAMP 89] et [CAMP 90a]. Cette approche résout le problème d'ordonnancement au delà des pas de contrôle, à partir d'une assignation de ressources déterminée. Les opérations ne sont plus affectées à un pas de contrôle, mais l'ordonnancement consiste à définir l'endroit où placer les changements de cycles.

Cette solution divise le GFC en sous-graphes, représentant chacun un chemin de contrôle, en découpant les blocs de base en chemins d'opérations et en supprimant les arcs de retour des boucles. L'ordre d'apparition des opérations sur chaque chemin est celui de la description comportementale

séquentielle initiale. Les contraintes de ressources, de temporisation ou de durée du cycle d'horloge sont modélisées par des intervalles entre deux opérations, sur lesquels il devra se produire un changement de pas de contrôle, pour respecter la contrainte.

Il s'agit alors de déterminer des coupures dans chaque chemin de contrôle, définissant la frontière entre les pas de contrôle. Un graphe de contraintes modélise les chevauchements entre intervalles sur un même chemin de contrôle. La minimisation du nombre de changements de cycles est réalisée par la couverture minimale en cliques de ce graphe. Ce partitionnement définit un ensemble de suites de sommets sur lesquelles une coupure doit être réalisée. Cependant, une même opération peut intervenir dans plusieurs chemins de contrôle. Il faut donc définir des coupures compatibles entre les différents chemins de contrôle. Il s'agit alors de trouver l'ensemble minimal de coupures ne se chevauchant pas, par un nouveau partitionnement en cliques.

Cette méthode part d'une assignation d'opérateurs fixée. Un algorithme d'assignation d'opérateurs permettant de réduire le nombre de pas de contrôle a donc été défini et implanté dans le système HIS [BERG 91a et b]. Cet algorithme traite chaque chemin de contrôle séparément. C'est un algorithme séquentiel sur les opérations, qui les classe dans l'ordre croissant du nombre d'instances pouvant leur être assignées. Par ailleurs, une amélioration d'APVI permet de minimiser le nombre de registres durant l'ordonnancement, en choisissant la coupe qui minimise le nombre d'intervalles de vie de variables coupés.

I.5.8. Récapitulatif

Les méthodes d'ordonnancement peuvent être classifiées selon plusieurs critères:

- l'utilisation d'une approche constructive (par pas de contrôle ou par opération), itérative ou globale,
- les paramètres à minimiser qui peuvent, selon les systèmes, intégrer les opérateurs, les registres et les connexions (Min. des opérateurs, Min. des registres, Min. des connexions),
- les étapes de l'allocation de ressources ($SELEC_{op}$ et/ou Nb(op) et/ou $ASSIGN_{ins}$ et/ou l'allocation de registres) qu'elles réalisent avant (Av), pendant (P) ou après (Ap) l'ordonnancement,
- la classe d'ordonnancements à laquelle elles appartiennent (OCM, OCT ou OCD).

Les caractéristiques de plusieurs méthodes d'ordonnancement sont récapitulées dans le tableau I.1 suivant.

OPF dénote l'ordonnancement orienté par les forces; *liste* dénote les ordonnancements par liste; *PL 0-1* dénote l'ordonnancement utilisant une formulation par un programme linéaire en 0 et 1. *MIMOLA* dénote l'approche utilisée par MIMOLA et *Rec Sim* dénote les ordonnancements par recuit simulé.

Tableau I.1 Récapitulatif des méthodes d'ordonnancement

(Légende ci-dessus)

Méthodes	OPF	Liste	MIMOLA	PL 0-1	APVI	Rec Sim	APTO
Min. des opérateurs	Oui	Fixé	Oui	Oui	Fixé	Oui	Non
Min. des registres	Oui	Non	Oui	Oui	Oui	Oui	Non
Min. des connexions	Oui	Non	Oui	Oui	Non	Oui	Non
OCM		Oui		Oui	Oui		
OCT	Oui			Oui			Oui
OCD				Oui			
Par opération	Oui						
Par pas de contrôle		Oui					Oui
Itérative						Oui	
Globale				Oui	Oui		
$SELEC_{op}$	Av	Av	P	Av	Av	P	Av
Nb(op)	P	Av	P		Av	P	P
$ASSIGN_{ins}$	Ap	Ap	P	Ap	Av	P	Ap
Allocation de Registres	Ap	Ap	P	Ap	Ap	P	Ap
Systèmes utilisant cette méthode	HAL EASY CADDY	CATHEDRAL BUD ELF SLICER EMUCS	MIMOLA	[LEE 89] [PAPA 90] [HWAN 91]	HIS YSC	[SAFI 90] [DEVA 87]	FACET

Toutes les méthodes d'ordonnancement intègrent le nombre d'opérateurs dans la fonction d'estimation de la surface du circuit, exceptées les méthodes d'ordonnancement au plus tôt (APTO) et au plus tard (APTA) qui ne tiennent pas compte du nombre de ressources pendant l'ordonnancement. Certaines méthodes, telles que la programmation linéaire en nombres entiers, l'ordonnancemnt orienté par les forces (OPF), l'ordonnancment au plus vite (APVI), et certaines méthodes par liste, intègrent aussi un coût sur les points de mémorisation soit en les estimant par le nombre d'intervalles de vie de variables, soit en déterminant leur nombre exact. En effet, les méthodes résolvant simultanément l'allocation de registres et l'ordonnancement, telles que celles fondées sur le recuit simulé, peuvent connaître le nombre exact de points de mémorisation lors de l'ordonnancement. Enfin, le nombre de connexions est pris en compte pour l'OPF, l'APVI, les méthodes fondées sur le recuit simulé ou la programmation linéaire en nombres entiers et certaines méthodes par liste. A notre connaissance, aucun système n'intègre un coût sur la partie de contrôle.

La plupart des méthodes d'ordonnancement présentées précédemment ne traitent pas les boucles imbriquées ou successives. Une méthode réalisant ce traitement est l'ordonnancement APVI. Cette méthode résout le problème d'ordonnancement avec assignation d'opérateurs déterminée

(OCM_ASSIGN section I.3.4.). La définition de l'assignation des ressources, lors des premières étapes de la synthèse, restreint le problème d'ordonnancement. De plus, cette méthode ne remet pas en cause l'ordre initial de la description. Si une instruction i_1 a été écrite avant une instruction i_2, et que ces deux instructions font intervenir une même ressource, alors l'instruction i_1 sera toujours ordonnancée avant l'instruction i_2, même s'il n'y a pas de dépendance de données entre ces deux instructions. L'ordonnancement APVI paraît donc plus indiqué pour des systèmes de synthèse partant d'une description où l'assignation est fixée par le concepteur et où le contrôle sur les opérations est important.

Les approches fondées sur le recuit simulé ne sont pas totalement adaptées à la synthèse de haut niveau telle qu'elle a été définie jusqu'à présent. En effet, leurs critères sont actuellement des estimations de la surface et des performances du circuit. Ils ne prennent pas en compte la surface exacte du circuit après placement et routage, ni la surface et les performances de la partie de contrôle. Par conséquent, le circuit final ne peut être obtenu que par itérations successives des phases de synthèse, ce qui est incompatible avec le temps nécessaire au recuit simulé.

L'ordonnancement par liste assigne toutes les opérations possibles, en respectant la dépendance des données et les contraintes de ressources, dès les premiers pas de contrôle. Cela a l'inconvénient de donner des ordonnancements avec une forte occupation des premiers pas de contrôle et une faible occupation des derniers pas de contrôle. Si cela ne pose pas forcément de problème en ce qui concerne le nombre d'opérateurs, le nombre minimal de registres nécessaires est souvent prohibitif. En effet, les intervalles de vie des variables se voient trop largement étendus pour pouvoir transmettre des données aux opérations des derniers pas de contrôle.

L'ordonnancement par programmation linéaire en nombres entiers est une voie à explorer. Cette méthode se base sur un problème NP-Complet. Cependant, la formulation de [HWAN 91] semble être efficace pour des problèmes de taille raisonnable. En partitionnant des problèmes de taille trop importante, la programmation linéaire en nombres entiers serait donc une solution applicable. Il faut tout de même faire remarquer que le problème de partitionnement en minimisant la coupe (le nombre de variables passées d'une partie à l'autre) est encore un problème NP-Complet. De plus, la résolution de l'ordonnancement ne remettra pas en cause la partition initiale.

Nous avons choisi l'ordonnancement orienté par les forces qui permet de résoudre l'ordonnancement sous contrainte de temps. Nous sommes partis de l'idée qu'un tel type d'ordonnancement permet de donner une indication sur le nombre de ressources nécessaires durant les premières étapes de la synthèse. Une indication sur le temps est déjà donnée par le chemin critique du graphe de précédence (GP définition I.14). De plus, l'ordonnancement orienté par les forces permet d'obtenir une bonne occupation des opérateurs au cours du temps, ceci par une approche globale (contrairement à l'ordonnancement par liste qui utilise une fonction locale) qui prend en compte les effets de bord de l'assignation d'une opération à un pas de contrôle sur les opérateurs. Les opérations non ordonnancées influent sur le choix d'un pas de contrôle de manière probabiliste plutôt que déterministe. Enfin, les compromis entre les besoins en opérateurs, registres et connexions sont résolus en utilisant une pondération. Cette méthode a cependant le défaut de ne pas traiter les boucles imbriquées et successives et d'être une méthode d'ordonnancement séquentielle par opération.

I.6. Résultats

Les résultats de l'ordonnancement OPF de base peuvent être trouvés dans la littérature [PAUL 88], [PAUL 89b]. En général, ces résultats sont équivalents à ceux obtenus par d'autres systèmes pour les exemples traités, voire meilleurs dans certains cas. Dans [HEIJ 90], l'ordonnancement OPF, avec contrainte de raideur globale, anticipation globale et réduction progressive des intervalles de positionnement [VERH 91], a été comparé à différentes stratégies d'ordonnancement par liste. Il en ressort que les deux algorithmes ont des performances similaires. L'OPF amélioré donne des résultats optimaux dans légèrement plus de cas que l'ordonnancement par liste, mais, dans ces cas là, la différence par rapport à l'optimum est plus importante que pour l'ordonnancement par liste. Si l'ordonnancement OPF permet d'intégrer plus de cas particuliers de la synthèse, il semblerait que son temps de calcul soit trop important pour des exemples complexes: $0(m^2.n^3)$ contre $0(n^2)$.
D'après les essais que nous avons réalisés, il semblerait que l'OPF donne de très bons résultats pour une contrainte de temps globale proche du chemin critique.
Les deux sections qui vont suivre comparent les différents algorithmes d'ordonnancement OPF, afin d'évaluer les améliorations proposées dans ce chapitre.
L'ordonnancement de base proposé dans [PAUL 89b] et l'ordonnancement avec nos améliorations ont été implantés en langage C sur station SUN 4, en prenant comme description initiale un graphe représentant une boucle au format ASCIS [ASCI 91]
Les exemples traités dans cette section sont quatre exemples types de la communauté internationale sélectionnés au "5th workshop on High Level Synthesis": l'équation différentielle présentée précédemment [PAUL 86], le filtre du 5ème ordre [KUNG 85], le filtre FIR [PARK 88] et l'exemple de MAHA [PARK 86]. Deux exemples, proposés par IMEC, notés mean2_exp et vb7_fgm ont également été utilisés.
Le GP de l'exemple du filtre du 5ème ordre peut être trouvé en annexe III.
Le GP de vb7_fgm est donné figure I.19. Dans ce graphe, les opérations "S" sont des décalages et durent 10 ns; les opérations "+" et "-" sont exécutées sur le même opérateur AS_CORE et durent 20 ns; les opérations "nil" sont des opérations d'entrée/sortie; elles n'ont pas été prises en compte.
Le GP de mean2_exp est donné figure I.20. Dans ce graphe, les opérations ShiftD sont allouées à l'opérateur "Shifter" et durent 10 ns; les opérations "add", "sign" et "abs" durent 20 ns; les opérations "abs" et "sign" sont allouées au même opérateur "FullAdder"; l'opération "add" est allouée à l'opérateur "MC_CORE".

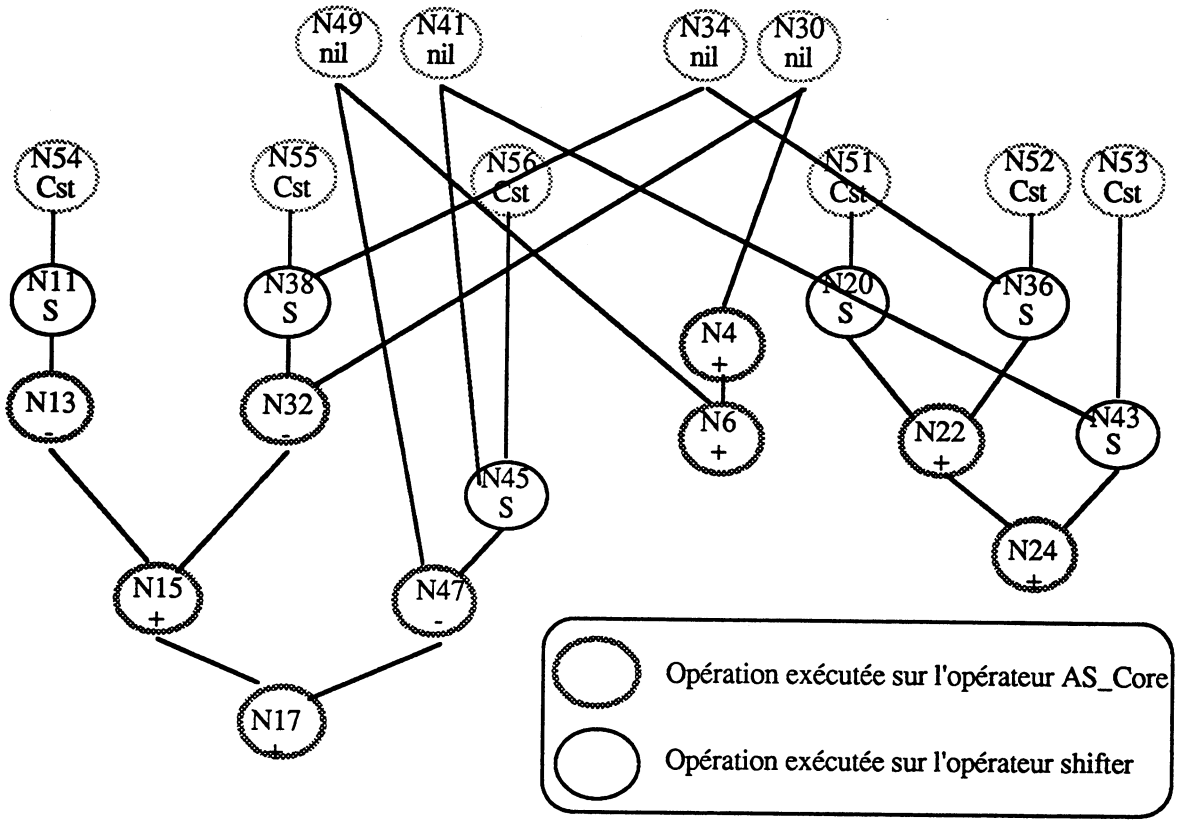


Figure I.19: GP de l'exemple vb7_fgm

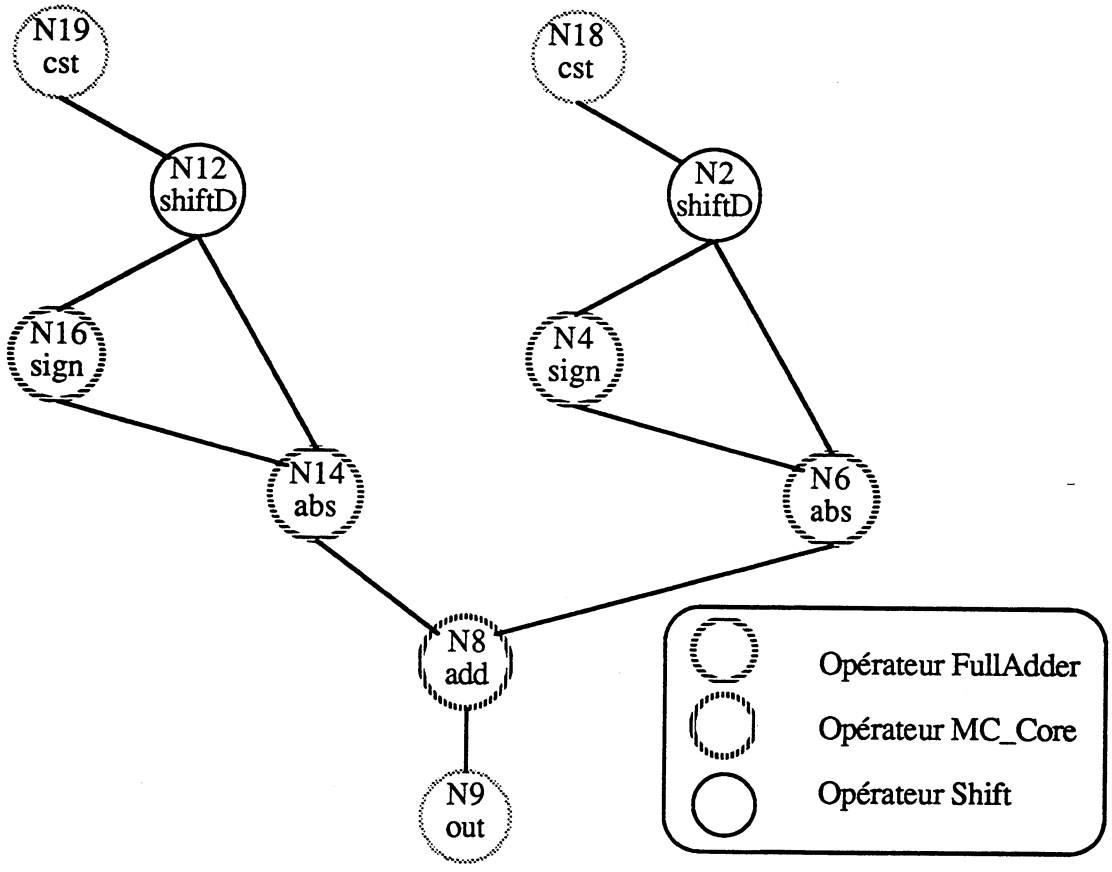


Figure I.20 : GP de l'exemple de mean2_exp

I.6.1. Résultats sur l'anticipation

Cette section va évaluer les résultats de l'OPF selon l'anticipation choisie. OPF dénote l'ordonnancement de base proposé dans [PAUL 89b] (définition I.28), OPF_Loc l'ordonnancement OPF avec anticipation locale de [PAUL 89b] (définition I.32). OPF_Màj dénote l'ordonnancement avec anticipation globale par mise à jour progressive des graphes de distribution (définition I.33), proposée dans cette thèse. OPF_Glob dénote l'ordonnancement avec anticipation globale (définition I.34), proposée dans [VERH 91], sans les autres améliorations proposées dans le même article. Pour OPF_Loc comme pour OPF_Glob, le facteur d'anticipation est de 1/3. Le tableau I.2 donne des résultats sur le filtre du 5ème ordre et les tableaux I.3 et I.4 sur l'exemple vb7_fgm. Les résultats en gras et encadrés sont les résultats différents d'une méthode à l'autre. Le tableau I.5 compare les résultats des méthodes entre elles.

Tableau I.2: Résultats sur le filtre du 5ème ordre

en nombre d'additionneurs (+), de multiplieurs (*) et de multiplieurs pipelinés (*p), et récapitulatif en nombre de résultats supérieurs (+), égaux (=) et inférieurs (-) d'une méthode à l'autre pour différentes contraintes de temps globales en nombre de pas de contrôle (pas de cont.)

pas de cont.	17	18	19	20	21	28	+	=	-
OPF	3+, 3*	3+, 2*	2+, 2*	2+, 2*	2+, 1*	2+, 1*	0	6	0
OPF_Loc	3+, 3*	**3+, 3***	2+, 2*	2+, 2*	**2+, 2***		0	3	2
OPF_Glob	3+, 3*	**2+, 2***	2+, 2*	2+, 2*	2+, 1*	2+, 1*	1	5	0
OPF_Màj	3+, 3*	3+, 2*	2+, 2*	2+, 2*	2+, 1*	2+, 1*	0	6	0
pas de cont.	17	18	19	20	21	28	+	=	-
OPF	3+, 2*p	3+, 1*p	2+, 1*p	2+, 1*p	**2+,2*p**		0	4	1
OPF_Loc	3+, 2*p	**3+,2*p**	2+, 1*p	2+, 1*p	2+, 1*p		0	4	1
OPF_Glob	3+, 2*p	3+, 1*p	2+, 1*p	2+, 1*p	2+, 1*p		0	5	0
OPF_Màj	3+, 2*p	3+, 1*p	2+, 1*p	2+, 1*p	2+, 1*p	2+, 1*p	0	6	0

Tableau I.3: Résultats sur vb7_fgm avec une horloge à 10 ns

en nombre d'AS_Core (A) et de Shifter (S), et récapitulatif en nombre de résultats supérieurs (+), égaux (=) et inférieurs (-) d'une méthode à l'autre pour différentes contraintes de temps globales en nombre de pas de contrôle (pas de cont.)

pas de cont	7	8	9	10	11	18	19 à 30	+	=	-
OPF	3A, 2S	3A, 2S	3A,2S	2A, 1S	**2A,2S**	**2A,2S**	**2A,2S**	0	4	14
OPF_Loc	3A, 2S	3A, 2S	3A,2S	2A, 1S	2A,1S	2A, 1S	1A, 1S	0	18	0
OPF_Glob	3A, 2S	3A, 2S	**3A, 1S**	2A, 1S	2A, 1S	2A, 1S	1A, 1S	1	17	0
OPF_Màj	3A, 2S	3A, 2S	**3A, 1S**	2A, 1S	2A, 1S	2A, 1S	1A, 1S	1	17	0

Tableau I.4: Résultats sur vb7_fgm avec une horloge à 20 ns

en nombre d'AS_Core (A) et de Shifter (S), et récapitulatif en nombre de résultats supérieurs (+), égaux (=) et inférieurs (-) d'une méthode à l'autre pour différentes contraintes de temps globales en nombre de pas de contrôle (pas de cont.)

pas de cont	4	5	6	7	8	9	10 à 14	+	=	-
OPF	**3A, 3S**	2A, 2S	2A, 2S	**2A, 2S**	**2A, 2S**	2A, 1S	**2A, 1S**	0	3	8
OPF_Loc	3A, 2S	**3A, 2S**	2A, 2S	2A, 1S	2A, 1S	**1A, 1S**	1A, 1S	1	9	1
OPF_Glob	3A, 2S	2A, 2S	2A, 2S	2A, 1S	2A, 1S	2A, 1S	1A, 1S	0	11	0

Tableau I.5: Récapitulatif des résultats

en nombre de résultats supérieurs (+), égaux (=) et inférieurs (-) d'une méthode à l'autre et gain d'une méthode par rapport aux autres (Gain)

	+	=	-	Gain
OPF	0	17	23	- 23
OPF_Loc	1	34	4	- 3
OPF_Glob	2	27	0	+ 2
OPF_Màj	1	40	0	+ 1

En général, l'anticipation, quelle qu'elle soit, améliore le résultat de OPF. L'anticipation locale donne de moins bons résultats que les deux types d'anticipation globale. Dans un cas, l'anticipation globale de Veraegh est meilleure que l'anticipation par mise à jour successive. Cependant, ce résultat a été donné pour une autre implantation d'OPF. Il est possible que cette implantation comporte une amélioration que nous n'avons pas implantée.

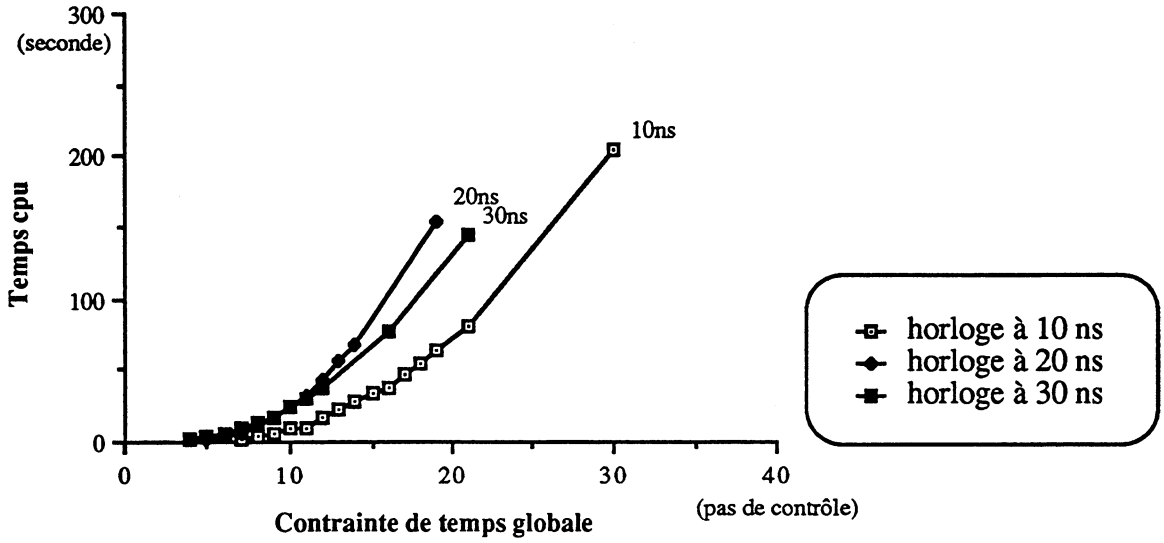
Ainsi, assez curieusement, pour l'exemple du filtre du 5ème ordre, l'ordonnancement avec anticipation locale donne de moins bons résultats que l'ordonnancement sans anticipation. L'OPF_Loc ne donne pas les résultats annoncés dans la littérature par Paulin [PAUL 89a et b]. Cette différence semble être due au fait que la version d'OPF implantée par Paulin intègre une correction sur le calcul des graphes de distribution que nous n'avons pas implantée. Cette correction soustrait aux GD de chaque opérateur la valeur de la capacité de cet opérateur (le nombre d'opérateurs alloués moins le nombre d'opérations du pas de contrôle) [PAUL 87].

L'amélioration de l'anticipation par mise à jour successive des graphes de distribution, proposée dans cette thèse, donne donc de meilleurs résultats que l'anticipation proposée initialement par Paulin, sans augmenter de manière considérable le temps de calcul. Dans un cas seulement, l'anticipation globale proposée par Verhaegh donne de meilleurs résultats que celle par mise à jour successive.

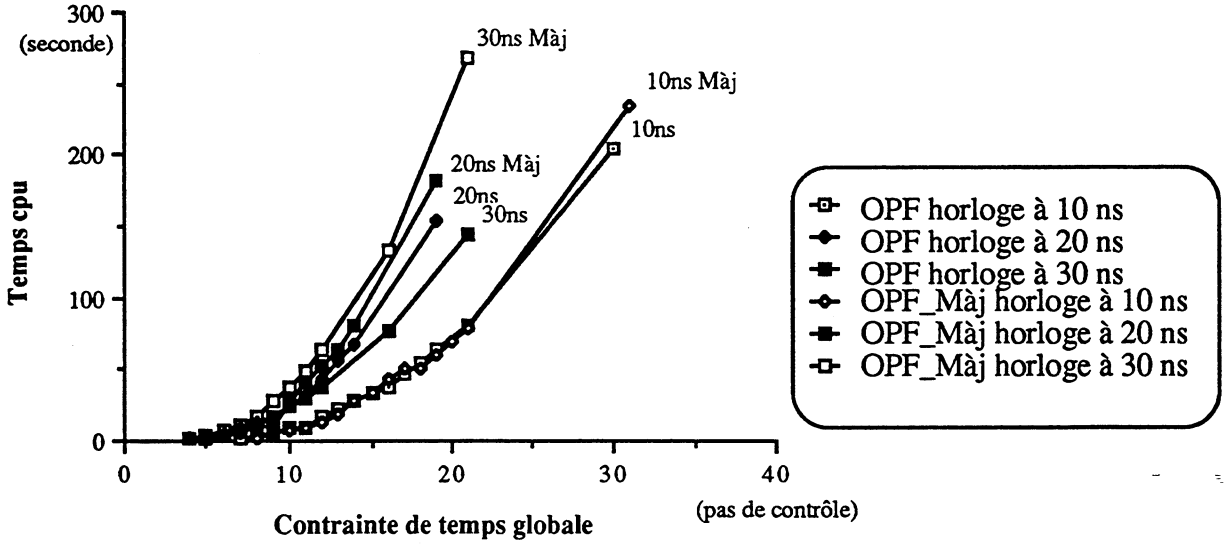
I.6.2. Résultats sur le temps cpu

Pour pouvoir évaluer les différences effectives de complexité de calcul entre l'OPF de base et l'OPF avec mise à jour progressive des graphes de distribution proposé ici, la figure I.21 donne les courbes du temps cpu en secondes sur l'exemple vb7_fgm, pour les trois durées de pas de contrôle essayées,

par rapport à la contrainte globale de temps en pas de contrôle. La figure I.21(a) donne le temps cpu pour OPF et la figure I.21(b) donne le temps cpu pour OPF_Màj et OPF. Les expérimentations sur différents exemples ont montré que le temps cpu augmentait d'environ 15%, du fait de l'amélioration que nous avons proposée par mise à jour temporaire des graphes de distribution.



(a) temps cpu pour OPF sur l'exemple vb7_fgm avec différentes durées de cycle d'horloge



(b) comparaison entre OPF et OPF_Màj sur l'exemple vb7_fgm avec différentes durées de cycle d'horloge

Figure I.21: Courbes des temps cpu pour OPF et OPF_Màj avec différentes contraintes de temps pour l'exemple vb7_fgm

Une critique possible de l'OPF est celle de sa complexité de $0(m^2.n^3)$. Pour les exemples traités jusqu'à présent, le temps de calcul est tout à fait acceptable puisqu'il ne dépasse jamais 10 minutes pour des descriptions contenant jusqu'à 34 opérations et 28 pas de contrôle. En général, le temps de calcul est presque instantanné pour des contraintes de temps globales proches du chemin critique de l'exemple traité. Pour tester plus à fond cette complexité, nous avons ordonnancé deux exemples

contenant une centaine d'opérations. Le premier est un exemple de 135 opérations avec un chemin critique de 65 pas de contrôle pour une horloge à 50 ns. C'est l'exemple d'un filtre que l'on nomme "mwdelf", présenté dans [VERH 92]. Le deuxième est un exemple factice formé de trois filtres du $5^{ème}$ ordre mis en parallèle; le nombre d'opérations de cet exemple est donc de 102, son chemin critique reste 17 pas de contrôle.

L'exemple factice est ordonnancé en 6 minutes pour une contrainte de temps égale à son chemin critique. Pour une contrainte de temps augmentant de 30% par rapport au chemin critique, soit 21 pas de contrôle, le temps de calcul augmente considérablement pour atteindre 1/2 heure.

L'exemple du filtre mwdelf a nécessité 4 minutes pour un ordonnancement avec une contrainte de temps globale égale à son chemin critique et plus d'une heure pour une contrainte de 80 pas de contrôle (soit une contrainte éloignée de 25 %).

Le temps de calcul de l'ordonnancement orienté par les forces avec anticipation globale par mise à jour successive des graphes de distribution est acceptable pour une description contenant jusqu'à cent opérations et une contrainte de temps globale égale au chemin critique. Le temps cpu va alors de quelques secondes à quelques minutes. Pour des contraintes de temps globales plus éloignées du chemin critique (jusqu'à 30 %), le temps de calcul peut atteindre 1/2 heure pour des descriptions initiales contenant moins de cent opérations et plusieurs heures lorsque ces descriptions contiennent une centaine d'opérations.

Au delà de cent opérations, le temps de calcul sera de plus d'une 1/2 heure.

La complexité de l'algorithme implanté est de $0(m^2.n^3)$. Cette complexité peut être réduite à $0(m.n^2)$ en prenant une anticipation locale et non une anticipation globale du calcul des fonctions propres. Le temps de calcul de l'algorithme en sera forcément réduit.

I.6.3. Résultats sur le chaînage

La qualité d'un ordonnancement, en ce qui concerne le chaînage des opérations, se mesure en fonction du nombre de chaînages différents résultants et de la fréquence d'utilisation d'un chaînage.

En effet, plus un chaînage existant est utilisé, plus le nombre de registres et de connexions entre les opérateurs ainsi chaînés diminue. Nous proposons une mesure de la qualité d'un ordonnancement par rapport au chaînage, valant le rapport entre le nombre d'opérations chaînées et le nombre de chaînages différents (Q). Cette mesure illustre le taux d'occupation par les opérations chaînées des unités exécutives créées pour le chaînage. Elle reflète donc bien une régularité du chaînage par rapport au nombre d'opérations chaînées.

Le chaînage est illustré sur les deux exemples vb7_fgm, mean2_exp et l'exemple du filtre FIR, et non sur les exemples habituellement utilisés, parce que les GP de ces deux exemples comportent des régularités qu'il serait bon de retrouver lors du chaînage d'opérations au cours de l'ordonnancement. Sur tous les exemples traités, le nombre d'opérateurs est le même pour un OPF avec ou sans prise en compte du coût de chaînage. Après plusieurs essais sur le coefficient ß représentant le ratio entre le coût d'un nouveau chaînage et le coût d'un chaînage inverse et le coefficient Conn représentant le coût d'une connexion si un opérateur a un coût de 1, ces coefficients ont été fixés respectivement à 0,5 et 0,2.

Les tableaux I.6 et I.7 illustrent les résultats de l'OPF pour l'exemple mean2_exp.
Cet exemple avec une horloge à 30 ns donne le même résultat, avec ou sans coût de chaînage: deux apparitions du chaînage entre Shifter et FullAdder, pour une contrainte globale allant de 3 à 5 pas de contrôle.

Tableau I.6: Résultats sur mean_exp avec une horloge à 40 ns

Pour l'OPF, l'OPF avec coût sur le chaînage (OPF_Chaîn), en nombre de chaînages entre Shifter et FullAdder (SF) et entre FullAdder et MC_Core (FM) et en qualité du chaînage (Q). Les trois dernières colonnes récapitulent les résultats en nombre de résultats supérieurs (+), de résultats égaux (=) et de résultats inférieurs (-)

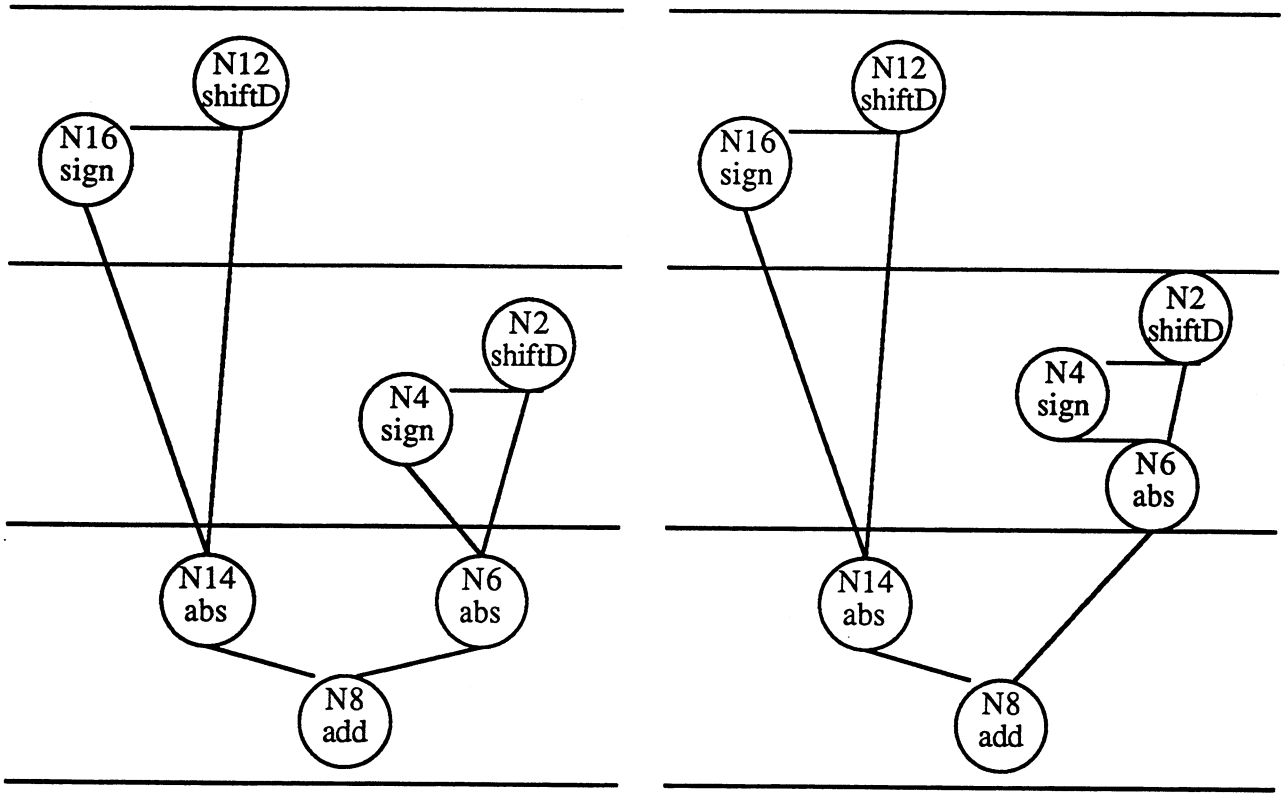
pas de contrôle	2			3			4					
Chaînage	SF	FM	Q	SF	FM	Q	SF	FM	Q	+	=	-
OPF	2	1	3/2	2	2	2	2	1	3/2	0	3	0
OPF_Chaîn	2	1	3/2	2	2	2	2	1	3/2	0	3	0

Tableau I.7: Résultats sur mean_exp avec une horloge à 50 ns

Pour l'OPF, l'OPF avec coût sur le chaînage (OPF_Chaîn), en nombre de chaînages entre Shifter et FullAdder (SF), entre FullAdder et AS_Core (FA) et entre FullAdder et MC_Core (FM) et en qualité du chaînage (Q). Les trois dernières colonnes récapitulent les résultats en nombre de résultats supérieurs (+), de résultats égaux (=) et de résultats inférieurs (-)

pas de contrôle	2				3				4						
Chaînage	SF	FA	FM	Q	SF	FA	FM	Q	SF	FA	FM	Q	+	=	-
OPF	2	0	2	2	3	1	1	5/3	2	0	1	3/2	0	2	1
OPF_Chaîn	2	0	2	2	**2**	**0**	**2**	**2**	2	0	1	3/2	1	2	0

Dans un cas, l'OPF avec coût sur le chaînage crée une situation de chaînage de moins que l'OPF sans coût sur le chaînage, tout en créant le même nombre de chaînages. Ce cas particulier est illustré figure I.22 (a) et (b).



(a) avec coût de chaînage (b) sans coût de chaînage

Figure I.22: Résultat de l'OPF pour mean2_exp pour une horloge à 50 ns et 3 pas de contrôle

Les résultats par rapport au chaînage sur l'exemple vb7_fgm sont illustrés tableaux I.8, I.9 et I.10.

Tableau I.8: Résultats sur vb7_fgm avec une horloge à 30 ns

Pour l'OPF, l'OPF avec coût sur le chaînage (OPF_Chaîn), en nombre de chaînages entre Shifter et AS_Core (SA) et en qualité du chaînage (Q). Les trois dernières colonnes récapitulent les résultats en nombre de résultats supérieurs (+), de résultats égaux (=) et de résultats inférieurs (-)

pas de contrôle	3		4		5		6		7		8		9		10				
Chaînage	SA	Q	SA	Q	SA	Q	SA	Q	SA	Q	SA	Q	SA	Q	SA	Q	+	=	-
OPF	6	6	5	5	5	5	4	4	3	3	4	4	4	4	4	4	0	3	5
OPF_Chaîn	6	6	5	5	**6**	**6**	4	4	**4**	**4**	**5**	**5**	**5**	**5**	**5**	**5**	5	3	0

Tableau I.9: Résultats sur vb7_fgm avec une horloge à 40 ns

Pour l'OPF, l'OPF avec coût sur le chaînage (OPF_Chaîn), selon le nombre de pas de contrôle (pas de cont), en nombre de chaînages entre Shifter et AS_Core (SA), en nombre de chaînages entre AS_Core et AS_Core (AA) et en qualité du chaînage (Q). Les trois dernières colonnes récapitulent les résultats en nombre de résultats supérieurs (+), de résultats égaux (=) et de résultats inférieurs (-)

pas de contrôle	2			3			4			5					
Chaînage	SA	AA	Q	SA	AA	Q	SA	AA	Q	SA	AA	Q	+	=	-
OPF	6	5	4	6	1	7/2	4	3	7/2	5	1	3	0	4	0
OPF_Chaîn	5	3	4	6	1	7/2	4	3	7/2	5	1	3	0	4	0
pas de contrôle	6			7			8			9					
Chaînage	SA	AA	Q	SA	AA	Q	SA	AA	Q	SA	AA	Q	+	=	-
OPF	4	2	3	4	0	4	3	0	3	3	0	3	0	1	3
OPF_Chaîn	4	2	3	**5**	**0**	**5**	**4**	**0**	**4**	**4**	**0**	**4**	3	1	0

Tableau I.10: Résultats sur vb7_fgm avec une horloge à 50 ns

Pour l'OPF, l'OPF avec coût sur le chaînage (OPF_Chaîn), selon le nombre de pas de contrôle (pas de cont), en nombre de chaînages entre Shifter et AS_Core (SA), en nombre de chaînages entre AS_Core et AS_Core (AA) et en qualité du chaînage (Q). Les trois dernières colonnes récapitulent les résultats en nombre de résultats supérieurs (+), de résultats égaux (=) et de résultats inférieurs (-)

pas de contrôle	2			3			4			5					
Chaînage	SA	AA	Q	SA	AA	Q	SA	AA	Q	SA	AA	Q	+	=	-
OPF	5	5	5	5	2	7/2	4	4	4	4	2	3	0	3	1
OPF_Chaîn	5	5	5	5	2	7/2	**5**	**4**	**9/2**	4	2	3	1	3	0

Les résultats de l'OPF avec coût de chaînage sont toujours meilleurs pour l'exemple vb7_fgm; une illustration en est donnée figure I.23. Sur cette figure, il est clair que le résultat de l'OPF sans coût de chaînage utilisera un registre supplémentaire.

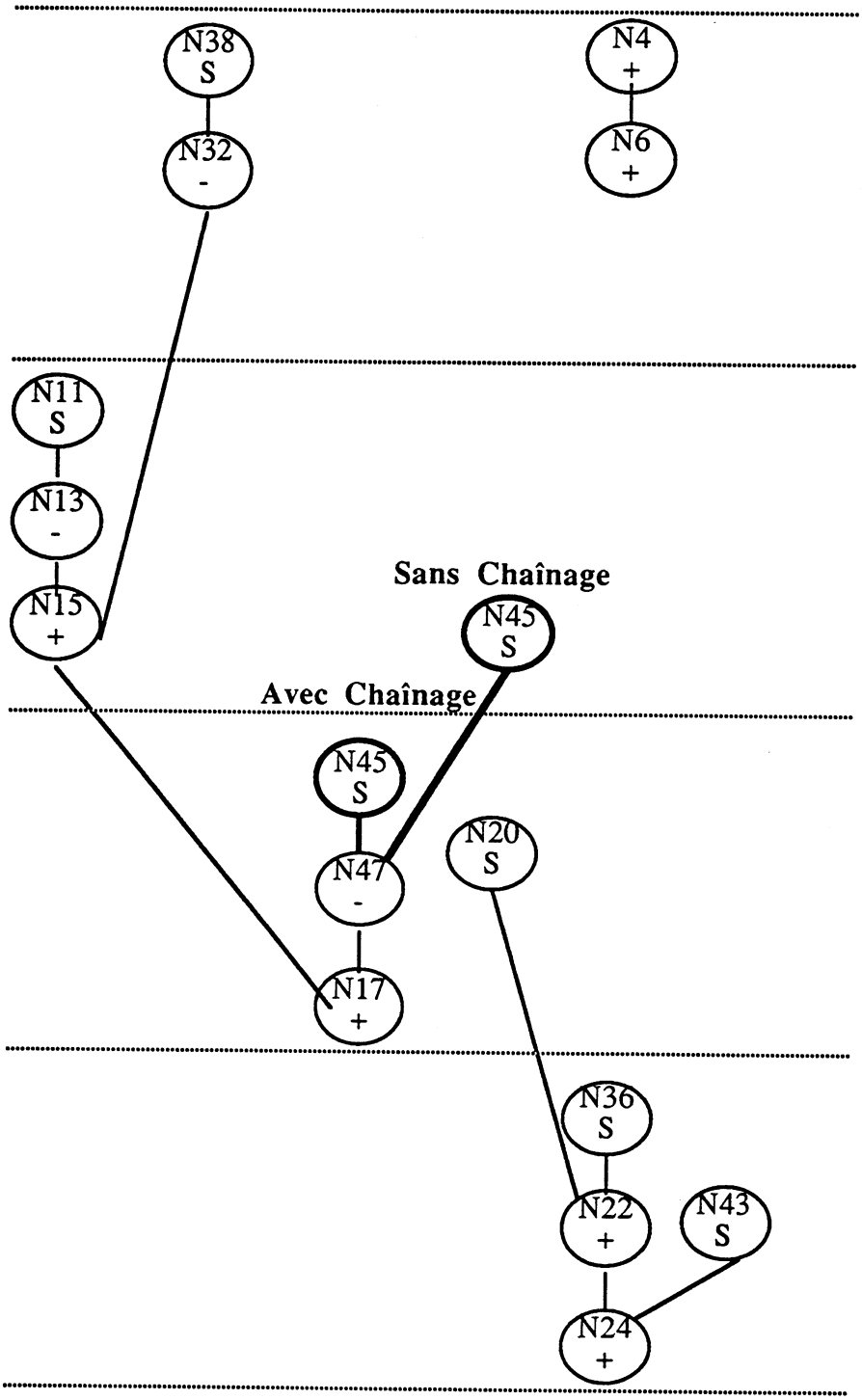


Figure I.23: Résultat d'OPF pour vb7_fgm, avec une horloge à 50 ns et 4 pas de contrôle.

Les résultats pour l'exemple du filtre FIR sont donnés dans le tableau I.11. Pour une horloge à 100 ns, les résultats sont toujours les mêmes: entre 4 et 10 pas de contrôle, on obtient 3 chaînages identiques.

Tableau I.11: Résultats sur FIR avec une horloge à 150 ns

Pour l'OPF, l'OPF avec coût sur le chaînage (OPF_Chaîn), en nombre de chaînages entre l'additionneur et le multiplieur (AM), entre l'additionneur et l'additionneur (AA) et entre le multiplieur et l'additionneur (MA) et en qualité du chaînage (Q). Les trois dernières colonnes récapitulent les résultats en nombre de résultats supérieurs (+), de résultats égaux (=) et de résultats inférieurs (-)

pas de contrôle	4				5				6						
Chaînage	AM	AA	MA	Q	AM	AA	MA	Q	AM	AA	MA	Q	+	=	-
OPF	7	4	1	4	4	3	3	10/3	**3**	**3**	**3**	**3**	0	2	1
OPF_Chaîn	5	4	3	4	3	3	4	10/3	4	3	3	10/3	1	2	0
pas de contrôle	7				8				9						
Chaînage	AM	AA	MA	Q	AM	AA	MA	Q	AM	AA	MA	Q	+	=	-
OPF	**3**	**3**	**3**	**3**	5	2	3	10/3	**2**	**3**	**2**	**7/3**	0	1	2
OPF_Chaîn	3	3	4	10/3	5	2	3	10/3	3	3	2	8/3	2	1	0
pas de contrôle	10														
Chaînage	AM	AA	MA	Q									+	=	-
OPF	2	3	2	7/3									0	0	1
OPF_Chaîn	3	3	2	8/3									1	0	0

Sur l'exemple du filtre FIR, les résultats sont meilleurs avec un coût sur le chaînage.

Le récapitulatif des résultats comparatifs de l'OPF sans coût de chaînage (OPF) et de l'OPF avec coût de chaînage (OPF_Chaîn) est donné dans le tableau I.12.

Tableau I.12: Récapitulatif des résultats

En nombre de résultats supérieurs (+), égaux (=) et inférieurs (-) d'une méthode à l'autre et gain d'une méthode par rapport à l'autre (Gain)

	+	=	-	Gain
OPF	0	14	10	- 10
OPF_Chaîn	10	14	0	+ 10

L'OPF intégrant un coût de chaînage présenté dans cette thèse a permis de ne pas générer une situation de chaînage supplémentaire dans un cas particulier et permet en général de réutiliser au maximum un chaînage déjà existant, ceci sans jamais augmenter le nombre d'opérateurs. Ce coût sur le chaînage permet donc de régulariser le chaînage des opérations. Cependant, ce coût n'intègre que des situations de chaînage formées de deux opérations. Des chaînage plus complexes, utilisés principalement pour des processeurs de traitement numérique du signal avec un haut débit, devront être pris en compte dans le futur.

I.7. Traitement de descriptions hiérarchisées

Les sections précédentes traitaient du problème d'ordonnancement des opérations d'une boucle élémentaire. Notons que plusieurs méthodes pour la réduction du temps d'exécution d'une boucle élémentaire ont été proposées. [PAUL 89b], [GIRC 87] proposent de pipeliner l'exécution du corps d'une boucle. [GOOS 89a] propose une approche similaire par pliage de boucles. [BERG 91a] et [POTA 90] proposent d'ordonnancer le traitement préliminaire et le traitement final d'une boucle avec le corps d'une boucle. Nous allons maintenant aborder les boucles successives et imbriquées.
Différents types de boucles peuvent être identifiés: les boucles finies et les boucles infinies .
Les boucles finies ont une condition d'arrêt ne dépendant pas du corps de la boucle, mais dont le nombre d'itérations est défini explicitement (ce sont des boucles de type "pour" avec borne constante).
Pour les boucles infinies, la condition d'arrêt dépend de variables calculées dans le corps de la boucle, par une fonction arithmétique plus complexe que l'incrémentation (ce sont des boucles de type "tant que"). Dans ce cas, le nombre d'itérations nécessaires à l'exécution de la boucle n'est pas connu à l'avance. Ce temps est considéré comme infini.
Les boucles finies peuvent être déroulées. Dans ce cas, les dépendances cycliques sont déroulées, donc supprimées. Les boucles finies non déroulées sont considérées comme des boucles infinies; la variable de condition d'arrêt est incrémentée en même temps que le calcul du corps de la boucle.
Dans le cas de boucles finies déroulées, les contraintes de précédence ne sont plus cycliques. Certains algorithmes permettent de réaliser l'ordonnancement de l'ensemble des boucles de manière globale. [LIPP 88] propose une adaptation de l'ordonnancement OPF pour l'ordonnancement d'une liste de boucles finies successives. Les ordonnancements APTA et APTO sont adaptés en prenant en compte la durée totale d'exécution des boucles. Le calcul des fonctions précédentes et suivantes prend en compte le fait que l'assignation d'une opération, à un pas de contrôle dans une boucle, peut avoir un effet sur les intervalles de positionnement des opérations des boucles précédentes et suivantes. [VERH 92] propose une représentation des boucles finies (avec condition d'arrêt linéaire, et nombre de pas de contrôle nécessaires à chaque boucle déterminé) par un calcul matriciel permettant de modéliser le problème d'ordonnancement par un programme linéaire en nombres entiers. Cette approche, bien que restreinte à un domaine particulier d'application, a l'originalité de remettre en cause la hiérarchie des boucles finies.
Dans le cas de boucles infinies, deux solutions permettent de contourner le cas de dépendance cyclique. Une première solution est basée sur la représentation par graphe de contrôle, l'autre sur une représentation hiérarchisée. La première consiste à décomposer un GFC en chemins de contrôle et utiliser l'ordonnancement APVI, présenté section I.5.7; la deuxième consiste à décomposer le GFC initial en chemins.

I.7.1. Hiérarchisation

La solution basée sur la *hiérarchisation* commence par décomposer le GFC en GFC représentant chacun une boucle élémentaire, comme nous l'avons vu section I.1.2. Les corps des boucles élémentaires sont d'abord ordonnancés puis les boucles sont séquencées les unes par rapport aux autres.
La hiérarchisation introduit des niveaux dans la description initiale. Les boucles élémentaires sont les boucles de plus bas niveau.
A un niveau donné, est alors construit un GFC connectant des macro-blocs représentant des boucles de niveau inférieur, en étendant la définition I.6.

Définition I.39: Un *macro-bloc de base* est constitué d'une boucle de niveau inférieur et des blocs de base (ne représentant pas de boucle) suivant ou précédant la boucle. Ces blocs sont construits de la manière suivante:
- Supprimer successivement chaque arc de retour des blocs représentant une boucle élémentaire.
- A chaque arc supprimé, reconsidérer la recherche des opérations de tête des blocs de base (définition I.4) (le branchement conditionnel de fin de boucle a auparavant été supprimé de la liste des opérations).

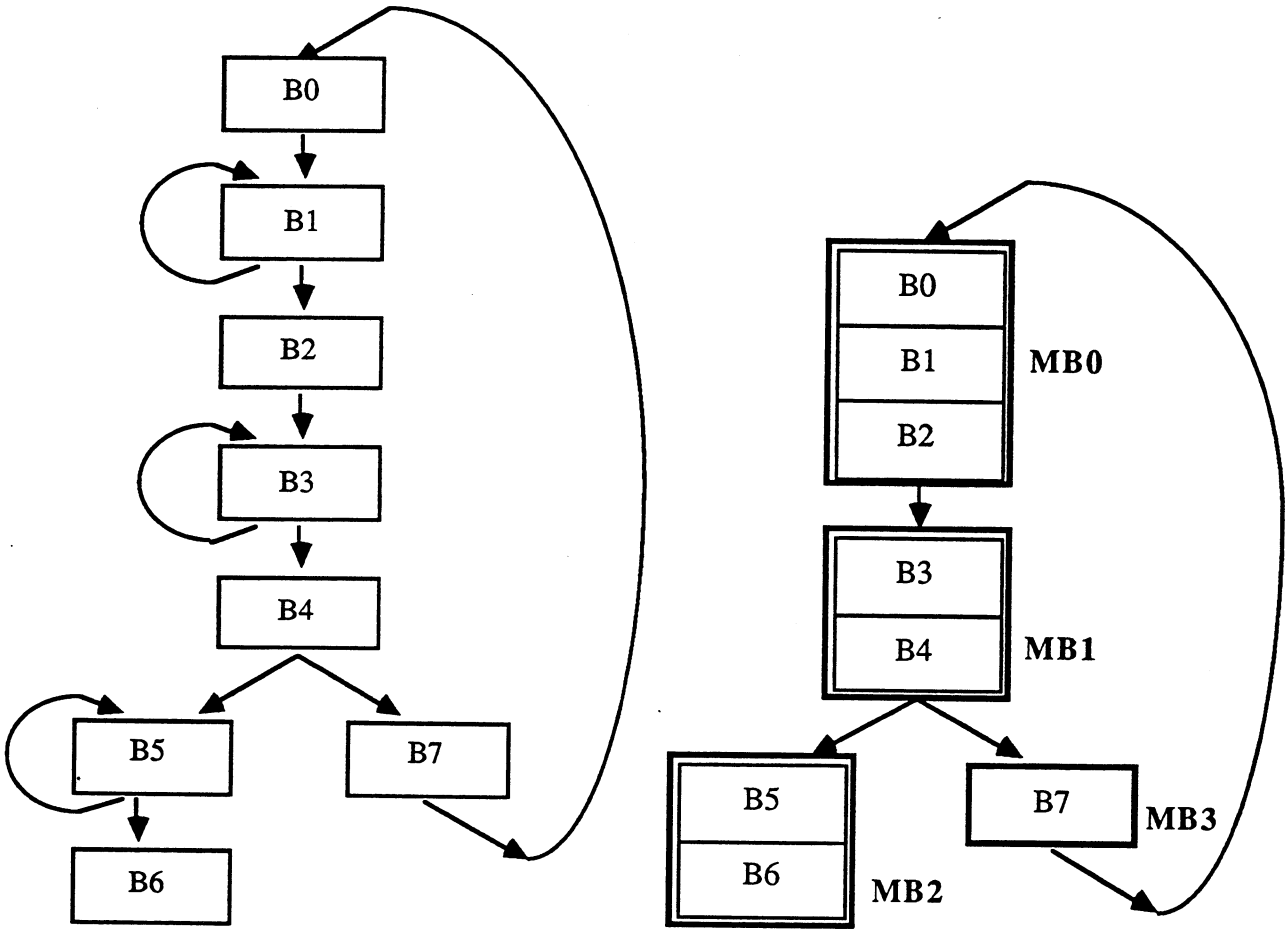
Par l'application de cette méthode de construction, chaque boucle est "collée" successivement à ses blocs de base directement précédents ou suivants. Il est donc intéressant de commencer par supprimer les arcs de retour de boucles très fortement connectées, du point de vue de la dépendance de données, à ses blocs suivants ou précédents.

Définition I.40: Le *GFC de niveau supérieur* du GFC G(B,A) (définition I.6) est le graphe G(MB,A-C), où MB est l'ensemble des macro-blocs de base représentant des boucles élémentaires et où A-C est l'ensemble des arcs de A en supprimant les arcs de retour des boucles élémentaires et les arcs entre les blocs de base regroupés (C).

Définition I.41: Les *boucles successives* sont des boucles de même niveau. Soit B_1 une *boucle imbriquée* dans une boucle B_2, B_1 est de niveau inférieur à B_2.

Définition I.42: Le GP d'une boucle élémentaire de *niveau supérieur* est le graphe G(MB,U) où MB est l'ensemble des macro-blocs de base représentant les boucles élémentaires imbriquées dans la boucle élémentaire et U l'ensemble des arcs représentant la dépendance de données. Un arc lie m_1 à m_2, si une valeur au moins est définie par m_1 et utilisée par m_2.

Le GFC de la figure I.24 (a) devient celui de la figure I.24(b) en passant à un niveau supérieur.



(a) Niveau le plus bas (b) Niveau supérieur

Figure I.24: GFC de niveaux différents

I.7.2. Ordonnancement de GP hiérarchisés

Nous nous proposons donc de définir une méthode d'ordonnancement par hiérarchisation. L'ordonnancement est appliqué en partant du plus bas niveau de hiérarchie. Lors de l'ordonnancement d'un GFC de niveau supérieur, le corps de chaque macro-bloc de base est donc déjà ordonnancé.

Pour les ordonnancements de la classe OCT, le problème consiste à répartir la contrainte de temps de l'ensemble du GFC sur les contraintes de temps de chaque macro-bloc de base. Dans le cas de boucles imbriquées, la contrainte de temps sur un macro-bloc est retransmise au niveau inférieur. Le problème consiste à répartir une contrainte de temps sur plusieurs boucles successives de même niveau.

Nous proposons une solution se ramenant à un problème de la classe OCT qui consiste à réaliser l'ordonnancement du corps de chaque macro-bloc de base, en respectant son chemin critique, puis à ordonnancer les macro-blocs de base les uns par rapport aux autres, en respectant la contrainte globale de précédence. C'est un ordonnancement sous contraintes de précédence avec un seul processeur. L'ordonnancement APTO donne un résultat optimal. Si la contrainte de temps globale est respectée, la contrainte de temps locale, correspondant au macro-bloc de base utilisant le plus de ressources, est relâchée.

[POTK 89] propose une solution similaire en se ramenant à un problème OCM et en réalisant l'ordonnancement de chaque sous-graphe séparément. Si la contrainte de temps globale est respectée, une ressource, coûteuse en surface et peu utilisée, est supprimée et l'ordonnancement est à nouveau réalisé.

Nous venons de présenter une méthode par hiérarchisation permettant l'ordonnancement des boucles imbriquées et successives. Cette méthode représente chaque boucle élémentaire imbriquée par un sommet de type macro-bloc de base (définition I.39). Au plus bas niveau de la hiérarchie, les corps de boucle sont ordonnancés grâce à la méthode OPF, présentée en section I.4, en répartissant la contrainte de temps globale par itération de l'ordonnancement. Un premier ordonnancement est réalisé pour des contraintes de temps globales égales aux chemins critiques des boucles élémentaires, les pas de contrôle supplémentaires sont distribués en réordonnançant les boucles utilisant le plus de ressources avec des contraintes de temps globales plus éloignées du chemin critique. Aux niveaux supérieurs, les boucles successives sont ordonnancées au plus tôt (APTO défini section I.4.1).

I.8. Conclusion sur l'ordonnancement

Dans ce premier chapitre, nous avons démontré que le problème d'ordonnancement dans le cadre de la synthèse de haut niveau est un problème NP-Complet. Néanmoins, nous avons extrait quelques algorithmes polynomiaux de résolution exacte de l'ordonnancement. Les algorithmes par préemption semblent applicables à l'ordonnancement de graphes partitionnés. C'est une voie à explorer dans le futur.
Pour l'ordonnancement d'un graphe représentant une boucle élémentaire, nous avons choisi l'heuristique d'ordonnancement orienté par les forces. Cet algorithme, tel qu'il a été présenté dans [PAUL 89b], donne de bons résultats sur des exemples simples sans chaînage. Nous avons proposé deux améliorations de l'algorithme de base. La première permet de mieux anticiper sur l'évolution des graphes de distribution et donne des résultats qui améliorent effectivement l'algorithme de base de l'ordonnancement OPF et l'algorithme avec anticipation locale, proposé initialement par [PAUL 89b]. La deuxième amélioration concerne le chaînage. Cette amélioration contourne deux défauts de l'OPF: le fait que cet algorithme soit séquentiel et le fait qu'il ne prenne pas en compte la régularité des chaînages créés. Cette amélioration donne de meilleurs résultats que la formulation initiale de l'ordonnancement orienté par les forces.
Enfin, nous avons proposé une méthode par hiérarchisation pour l'ordonnancement de boucles imbriquées qui répartit une contrainte de temps globale sur des boucles successives.

Selon le domaine d'application, l'ordonnancement doit s'adapter à des descriptions initiales ayant diverses propriétés.
Les applications destinées au traitement numérique du signal, aux radars, à la vidéo, aux télécommunications ou à l'avionique effectuent des opérations arithmétiques complexes et mettent en jeu peu de branchements conditionnels. On dit qu'elles ont des descriptions initiales dominées par les données. Pour des questions de simulation avant et après la synthèse, ces descriptions contiennent parfois des contraintes de temps, en nombre de cycles d'horloge, entre les entrées et les sorties du circuit. A l'inverse, les applications destinées à la robotique ou à l'automatisme mettent en jeu principalement des branchements conditionnels, elles ont des descriptions dominées par le contrôle.

Les descriptions dominées par le contrôle impliquent peu, voire pas, d'ordonnancements. En effet, les blocs de base contiennent alors parfois qu'une seule opération. En ce sens, un ordonnancement Orienté Par les Forces (OPF, défini section I.4), qui équilibre l'occupation des pas de contrôle en nombre d'opérateurs sur le corps d'une boucle, n'est pas d'une utilité flagrante. Un ordonnancement utilisant au maximum les chemins de contrôle, tel que l'ordonnancement au plus vite (APVI), serait mieux adapté.
Au niveau de transferts de registres, des techniques remettant en cause un ordonnancement initiale, telles que [MARW 92], SCHOLAR [BERG 91], [STOL 92] ou [MIGN 92c], sont tout à fait

adaptées à des descriptions initiales dominées par le contrôle; des techniques de retemporisation, telles que [NOTE 89], le sont aussi.

Pour les spécifications qui ne sont pas dominées par le contrôle, la plupart des blocs de base contiennent plus d'une opération. Dans ce cas, l'ordonnancement orienté par les forces donne souvent des résultats optimaux. Cette technique d'ordonnancement peut donc être utilisée pour des spécifications dominées par les données.
Le temps de calcul de l'ordonnancement orienté par les forces, pour des descriptions contenant moins de cent opérations, est acceptable puisqu'il ne dépasse pas trente minutes. Cependant, dans le domaine des processeurs dédiés au traitement numérique du signal, certaines descriptions intiales contiennent plus de cent opérations [DEPU 90a et b], dans le pire des cas, ces opérations se trouvent toutes dans une même boucle. Pour plus d'une centaine d'opérations ou pour des contraintes de temps très éloignées du chemin critique, le temps de calcul peut atteindre plusieurs heures. Soit ce temps est acceptable pour un concepteur, sachant que l'OPF donne généralement des résultats optimaux avec anticipation globale, soit ce temps est inacceptable. Dans ce cas, une version de complexité réduite de l'OPF avec anticipation locale en $0(m.n^2)$ (par rapport à $0(m^2.n^3)$) peut être utilisée (section I.4.3). Les résultats de l'ordonnancement seront peut être moins bons, mais le temps de calcul sera largement réduit. Il faut alors noter que d'autres techiniques d'ordonnacement adaptées aux descriptions initiales dominées par le contrôle, telles que l'ordonnancement par liste ou l'ordonnancement par programmation linéaire en nombres entiers, ne supportent pas non plus les descriptions contenant plus de cent opérations: la programmation linéaire en raison de son temps de calcul et l'ordonnancement par liste en raison du nombre de registres que cette techinique génére. Généralement pour des descriptions comportant plus de cent opérations, un partionnement de la description initiale est nécessaire [DEPU 90a et b].

On distingue généralement trois types de processeurs dédiés au traitement numérique du signal: ceux à haut débit, ceux à moyen débit et ceux à bas débit. Dans les deux premiers cas, il est nécessaire de chaîner des opérations durant l'ordonnancement afin d'obtenir de meilleures performances du circuit. Dans le cas de processeurs à moyen débit, les chaînages excèdent rarement deux opérations. L'amélioration de l'ordonnancement orienté par les forces (section I.4.4.) pour prendre en compte la régularité des chaînages est alors efficace.
Dans le cas de processeurs à haut débit, des chaînages d'opérations complexes sont nécessaires. Ces chaînages contiennent de l'ordre d'une dizaine d'opérations. L' amélioration du chaînage, pour prendre en compte la régularité, ne mesure que les chaînages entre deux opérations. Elle ne suffit pas à générer un ordonnancement régulier au delà de deux ou trois opérations chaînées. Dans ce cas particulier, il est nécessaire que les opérations soient chaînées avant l'ordonnancement.

L'ordonnancement orienté par les forces résout le problème d'Ordonnancement sous Contraintes de Temps (OCT, défini section I.3.1); il permet donc de respecter les contraintes de temps, entre les entrées et les sorties d'un circuit, données dans une description initiale.

On notera, cependant, que les techniques de pipeline fonctionnel ou de pliage, habituellement appliquées pour améliorer le comportement des boucles d'une spécification dominée par les données, n'ont pas été abordées dans cette thèse.

Chapitre II

ALLOCATION DE RESSOURCES

Après l'ordonnancement, la phase d'allocation de ressources va construire progressivement la partie opérative du circuit. Dans notre contexte, la sélection du type d'opérateur, qui identifie l'opérateur en bibliothèque capable de réaliser chaque opération élémentaire de la description initiale, a été définie avant la synthèse. De plus, l'heuristique d'ordonnancement, décrite au chapitre précédent, a fixé le nombre d'instances d'opérateurs. Pour terminer la construction du réseau d'interconnexions de la partie opérative, il ne reste donc qu'à déterminer le nombre de registres nécessaires, à assigner ces registres aux données d'une description initiale ordonnancée et à allouer des éléments physiques de connexions (multiplexeurs ou bus) aux transferts de données entre les opérations élémentaires de la spécification initiale. L'allocation de registres et d'opérateurs est réalisée avec comme objectif la réduction de la surface de la partie opérative et en particulier de la surface des connexions. Cette surface est directement dépendante du style d'implantation de la partie opérative ciblé.

Une première section de ce chapitre énumère les styles d'architecture de la partie opérative que nous avons identifiés. Une deuxième section donne le point de départ de l'allocation de ressources qui est un graphe de dépendance de données ordonnancé, dans lequel des éléments virtuels ont été insérés. Les deux sections suivantes sont dédiées à la description de notre méthode qui commence par déterminer le nombre minimal d'opérateurs et de registres nécessaires et qui réalise ensuite une assignation de registres et d'opérateurs par un algorithme flexible, permettant d'intégrer les critères et les contraintes de différents styles d'architecture. Les autres approches pour l'allocation de ressources sont parcourues et comparées en dernière section.

II.1. Styles d'architecture

Une partie opérative connecte des registres à des opérateurs ou à de la logique combinatoire. Les architectures de la partie opérative utilisent des multiplexeurs, des bus et des segments de bus pour implanter les connexions.
Nous avons donc classifié les styles d'architecture selon le type d'éléments d'interconnexion que ces architectures utilisent.
Trois classes principales d'architecture de la partie opérative sont identifiées: les architectures à multiplexeurs, aussi appelées architectures à topologie aléatoire [LUI 91], les architectures à bus, aussi appelées architectures à topologie linéaire [LUI 91] et les architectures mixtes.

II.1.1. Architecture à base de multiplexeurs

L'architecture à base de multiplexeurs n'utilise que des connexions privées. Les sélections en entrée d'opérateurs et en entrée de registres se résolvent par l'introduction d'un ou plusieurs multiplexeurs. Cette architecture est illustrée figure II.1.

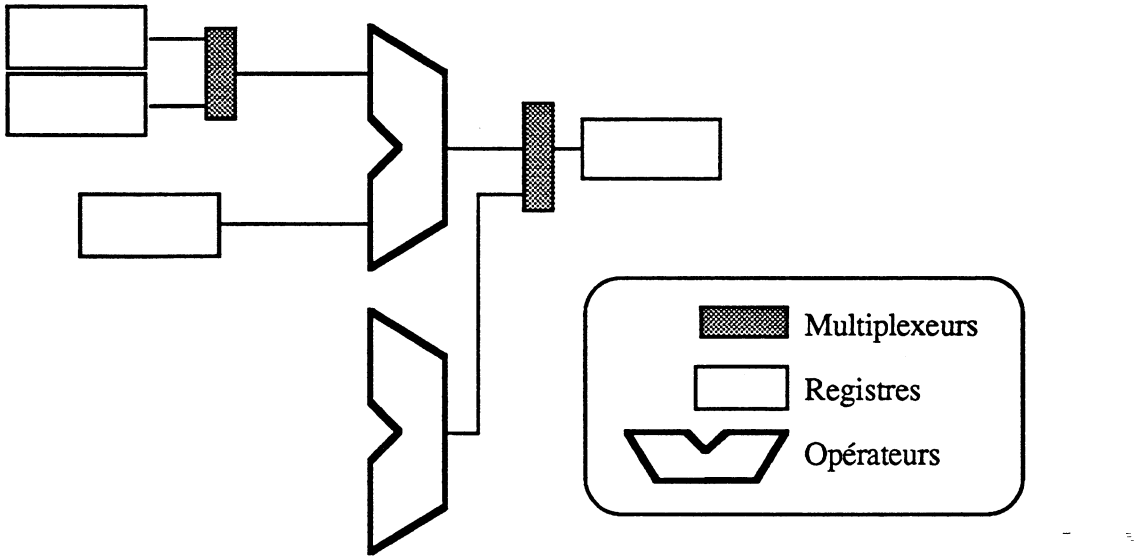


Figure II.1: Architecture à base de multiplexeurs

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 souvent plus rapide qu'une connexion commune par un bus. Ainsi, le processeur TMS320025 [SHAN 89] de Texas Instrument utilise deux bus, pour les transferts de données entre les deux parties principales du processeur, et des multiplexeurs, pour les transferts internes à chaque partie. C'est aussi le cas pour le processeur d'Intel i860 [KOHN 89].

II.1.2. Architectures à base de bus

L'intérêt de l'utilisation des bus par rapport aux multiplexeurs est qu'un même bus sert à l'implantation de plusieurs connexions. Les architectures à base de bus se distinguent selon qu'elles utilisent des bus ou des segments de bus.

Les architectures à segments de bus allouent deux bus pour toute la PO. Ces deux bus sont découpés en segments de bus. Deux segments de bus sont assignés à chaque unité fonctionnelle, un par entrée de l'unité. Les segments de bus, reliés à l'une des deux entrées des unités fonctionnelles, sont reliés deux à deux par des éléments commutateurs, tels que l'illustre la figure II.2. Ces éléments commutateurs peuvent être des portes trois états ou des interrupteurs. Ils ont deux états: ouvert ou fermé. Selon l'état du commutateur, deux segments de bus sont considérés comme deux bus séparés ou comme un seul bus.

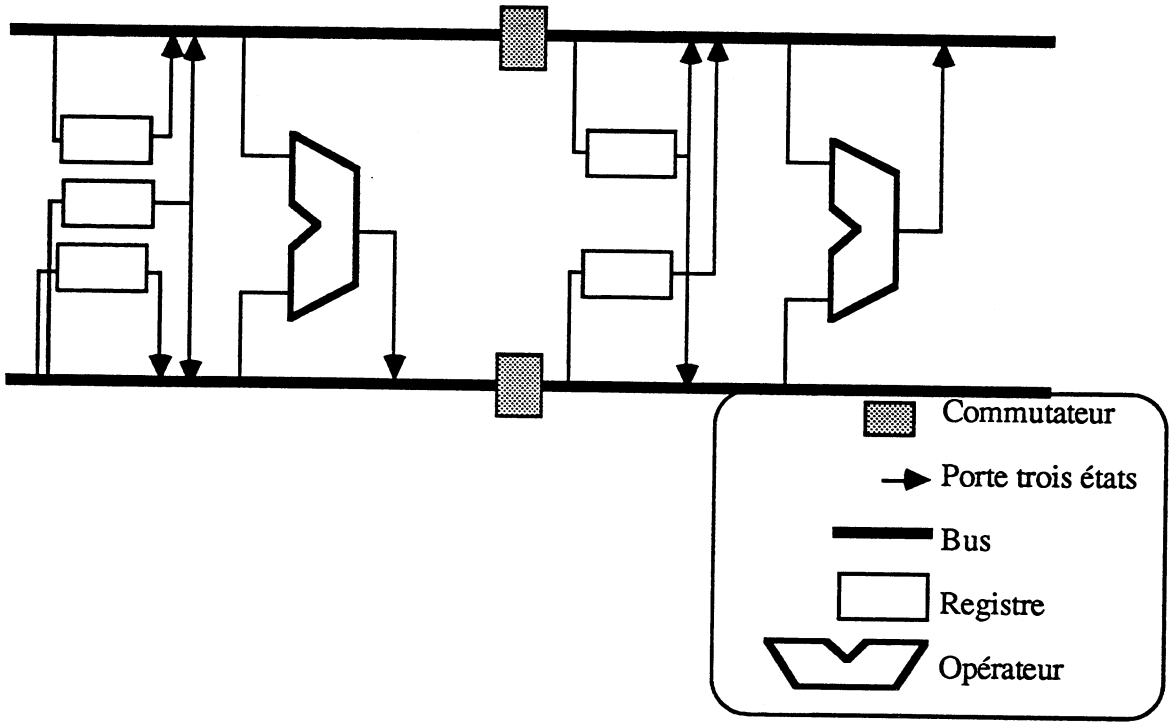


Figure II.2: Arhitecture à base de segments de bus

La notion de segments de bus est principalement utilisée pour les processeurs de Motorola. Le MC68000 et le 68020 [GROV 84] utilisent deux bus parallèles segmentés pour les transferts de données entre les unités traitant les bit de haut niveau d'adresse, celles traitant les bit de bas niveau d'adresse et celles traitant les données. Le WE32100 [SHAN 89] de AT&T utilise également des éléments commutateurs sur une architecture à base de bus.

Le style d'architecture à base de segments de bus a l'avantage de proposer une flexibilité de l'utilisation des bus. En revanche, il a l'inconvénient de contraindre les transferts de données à uniquement deux bus parallèles segmentés. Une donnée peut avoir besoin de plusieurs pas de contrôle pour transiter d'une unité fonctionnelle à une autre. L'ordonnancement doit donc être réalisé en conséquence. Ce style d'architecture implique une remise en cause de l'ordonnancement.

Les architectures à base de bus utilisent des paires de bus parallèles. Plusieurs styles d'architecture à bus sont distingués selon la localisation des registres. Les registres peuvent être soit communs à l'ensemble des opérateurs, soit dédiés en entrée d'opérateurs, soit dédiés en sortie d'opérateurs.

Définition II.1: Une architecture à bus et à *registres dédiés en entrée (en sortie)*, ne peut pas avoir un même registre connecté à deux entrées (sorties) différentes d'opérateurs. De plus, ces architectures utilisent des connexions privées pour les transferts en entrée (en sortie) d'opérateurs et des bus pour les transferts en sortie (en entrée) d'opérateurs.

Les styles d'architecture à registres dédiés impliquent la nécessité de dupliquer certains registres, afin que chaque registre soit effectivement dédié à un opérateur.
Dans le cas d'une architecture à registres dédiés en entrée, un registre apparaissant à plus d'une entrée d'opérateur, dans la description comportementale, doit être dupliqué. Une copie du registre est alors présente à chaque entrée d'opérateur devant laquelle il apparaît. Ce style d'architecture peut dédier plusieurs registres à une même entrée. Les registres dédiés à une même entrée d'opérateur sont parfois regroupés dans un banc de registres.
Pour une architecture à bus avec registres dédiés en sortie, les registres sont dupliqués de la même manière. Un exemple d'architecture à base de bus avec registres dédiés en sortie est donné figure II.3. Sur cet exemple, le registre R1 apparaissait en sortie de l'opérateur Op1 et en sortie de l'opérateur Op2. Il a donc été nécessaire de le dupliquer pour obtenir les registres R1_1 et R1_2.

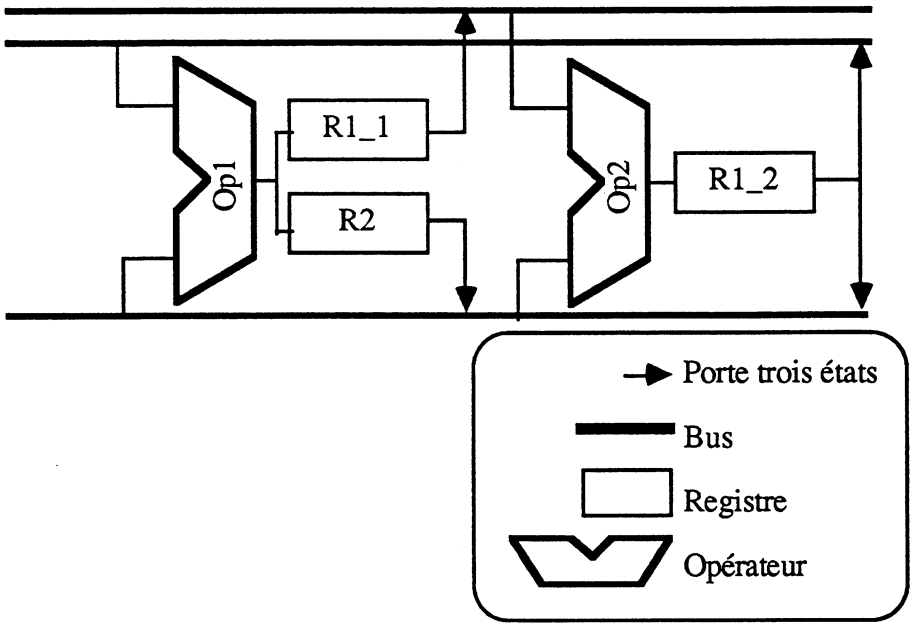


Figure II.3: Architecture à base de bus avec registres dédiés en sortie

La plupart des processeurs utilisent deux bus parallèles pour les transferts de données vers les entrées des opérateurs. Cependant, certaines architectures ont plus de deux bus parallèles; c'est le cas des deux processeurs RISC (Reduced Instruction Set Computer) TI et MDAC [CUSH 88].
Le cas typique de registres dédiés est le cas d'un registre accumulateur dédié en sortie d'UAL (Unité Arithmétique et Logique). Le TMS320C25 et le TMS32020 [SHAN 89] de Texas Instrument et le processeur RISC MDAC [CUSH 88] en sont des exemples.

Pour le système CATHEDRAL [DEMA 86], a été définie une architecture cible à base de bus et de registres dédiés en entrée d'unités fonctionnelles. Cette architecture a été définie après de nombreuses expérimentations, avec des concepteurs, sur la réalisation de processeurs pour le traitement d'images et le traitement audio [CATT 90].

Les architectures à registres dédiés ont le désavantage de nécessiter la duplication de registres. Nous allons donc définir une architecture à registres communs qui a cependant l'inconvénient d'utiliser plus de bus, puisque des bus sont alloués à la fois à la sortie et à l'entrée des opérateurs.

Définition II.2: Les architectures à bus et à *registres communs* utilisent des bus parallèles. Ces bus sont utilisés pour les transferts en entrée et en sortie d'opérateurs.

Dans le cas de registres communs, pour diminuer le nombre de bus, des registres tampons sont parfois insérés en entrée et/ou en sortie d'opérateurs. Les transferts de données en entrée d'opérateurs ne sont alors plus réalisés dans le même cycle d'horloge que les transferts de données en sortie d'opérateurs. L'insertion de registres tampons permet aussi de diminuer la durée d'un cycle d'horloge. Dans le cas d'une insertion de registres tampons en entrée ou en sortie d'opérateurs, un pas de contrôle est divisé en deux cycles d'horloge. Dans le cas d'insertion de registres tampons en entrée et en sortie d'opérateurs, un pas de contrôle est divisé en trois cycles d'horloge.

Les registres tampons sont distingués des registres dédiés par le fait qu'au cours de la synthèse, ils seront instanciés systématiquement en plus des registres nécessaires au stockage des variables d'une description initiale.

Un exemple d'une architecture avec des registres communs et des registres tampons en entrée est représenté figure II.4.

Le MC68332 [JELE 89] de Motorola utilise une architecture avec deux bus parallèles, des registres communs et des registres tampons en entrée d'opérateurs. Les architectures RISC utilisent souvent des registres tampons en entrée et en sortie d'opérateurs.

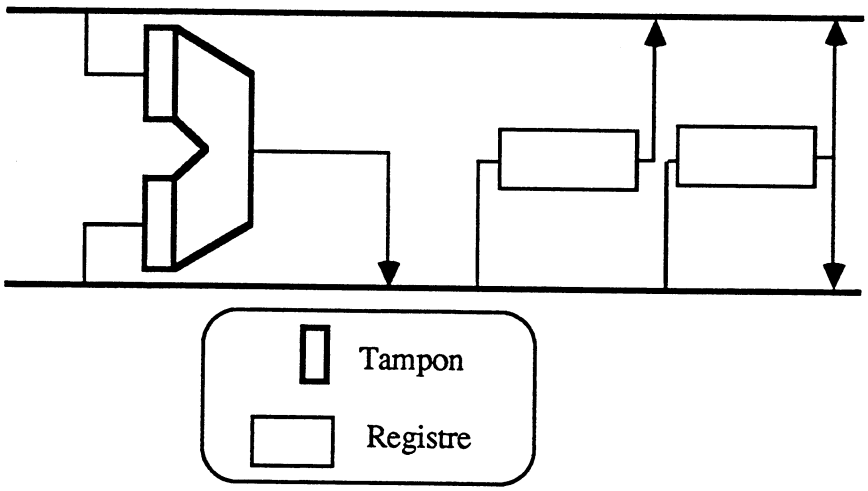


Figure II.4: Architecture à base de bus avec registres communs et registres tampons en entrée.

II.1.3. Architectures mixtes

Les architectures mixtes utilisent des bus et des multiplexeurs indifféremment en entrée de registres et d'opérateurs. Dans ce cas, le nombre de bus est généralement préalablement fixé.

II.1.4. Quelques cas particuliers

Les opérateurs de la partie opérative ne sont pas en permanence occupés par une opération. Les opérateurs libres peuvent donc servir au transfert d'une valeur. Il suffit pour cela d'exécuter une opération neutre sur cet opérateur.

Dans certains cas particuliers, la partie opérative peut être partiellement allouée avant le début de la phase d'allocation de ressources; les ressources déjà allouées peuvent resservir quand elles sont inoccupées; on parle alors de pré-assignation.

Le système de synthèse dédié au traitement d'images de bas niveau [BOUB 92a] est un exemple de cette particularité.

Les registres sont généralement des registres maître/esclave, pouvant donc être lus et écrits dans un même cycle d'horloge. Certaines architectures regroupent les registres dans des bancs de registres.

Définition II.3: Un *banc de registres* intègre plusieurs registres R_1, ..., R_n. Il possède des ports de lecture (*l* ports), des ports d'écriture (*e* ports) ou des ports de lecture/écriture (*el* ports). La lecture (ou l'écriture) d'un registre est réalisée par l'imposition de ce registre sur l'un des ports de lecture (ou d'écriture). Ainsi, plusieurs registres peuvent être lus ou écrits au même pas de contrôle sur un banc de registres: *l* registres lus, *e* écrits et *el* registres soit écrits, soit lus.

II.1.5. Styles d'architecture en synthèse

Habituellement, chaque système de synthèse architecturale définit et cible un style unique d'architecture.

Le système CATHEDRAL II [DEMA 86] définit des unités exécutives capables de réaliser plusieurs opérations chaînées. Ces unités ont chacune deux bancs de registres dédiés en entrée et sont reliées par des bus. Le système CADDY [KRAM 90] utilise des opérateurs ayant soit des tampons en entrée et en sortie et des registres communs, soit des registres dédiés en entrée d'opérateurs; il utilise des bus. Les systèmes MABAL [KUCU 89] et ELF [LY 90] utilisent des architectures mixtes et exploitent la possibilité de réaliser une opération neutre pour le transfert d'une donnée. Les systèmes APPOLON [JAMI 85] et AMICAL [PARK 92] utilisent des architectures à bus découpés en segments par des portes trois états. [EWER 90] utilise une architecture à bus parallèles segmentés par des interrupteurs. La plupart des autres systèmes utilisent des architectures à base de multiplexeurs. Pour le système FLORA [LUI 91], plusieurs styles d'architecture ont été définis: une architecture à base de multiplexeurs et des architectures à base de bus utilisant des bancs de registres communs. Ces architectures diffèrent par le nombre de bancs de registres et leur nombre de ports, le nombre de connexions entre les bancs de registres et les bus et le nombre de registres tampons insérés.

Nous nous proposons d'utiliser l'ensemble des architectures définies précédemment (à l'exception

des architectures à segments de bus n'utilisant que deux bus). Nous verrons dans le chapitre IV des estimations de surfaces de parties opératives permettant d'aider un concepteur dans le choix d'un style d'architecture.

Par la suite, nous présenterons des méthodes originales pour lesquelles nous nous attacherons à prendre comme objectif prioritaire la flexibilité par rapport aux différents styles d'architecture que nous venons de définir. Ces méthodes résolvent les problèmes d'assignation de ressources, pour la section II.4 de ce chapitre, et d'allocation des connexions, pour le chapitre suivant.

II.2. Description de départ de l'allocation de ressources

L'allocation de ressources assigne des éléments physiques d'une description initiale aux éléments virtuels de la PO. Les éléments physiques sont des opérateurs, des registres et des bus ou des multiplexeurs; les éléments virtuels représentent les opérations élémentaires, les données et les transferts de données. L'allocation de ressources nécessite une description de départ permettant de représenter ces éléments. Le point de départ de l'allocation de ressources est donc un GDD ordonnancé représentant une boucle élémentaire.

Définition II.4: un *GDD ordonnancé* est un GDD G(T, U) (définition I.12), tel que, pour chaque tâche t, Ord(t) est fixé.

Dans le GDD, les transferts de données sont représentés par des arcs, les éléments virtuels qui seront assignés à un registre doivent donc être insérés. Ils représentent des données qui doivent être conservées pendant une durée déterminée par l'ordonnancement. Dans ce but, les données définies à l'extérieur de la boucle, les données définies par la boucle et les données issues de dépendances cycliques doivent être représentées. Dans le GDD ordonnancé initial vont donc être insérés des sommets de type entrée ou sortie, des sommets de type début ou fin et des sommets représentant plus généralement les données.

Définition II.5: Un sommet de type *entrée* d'une boucle élémentaire représente une valeur définie dans un bloc de base du GFC ne faisant pas partie de la boucle élémentaire et utilisée dans la boucle élémentaire. Un sommet de type *sortie* d'une boucle élémentaire représente une valeur définie par la boucle et utilisée par une opération dans un bloc de base du GFC ne faisant pas partie de la boucle élémentaire.

Définition II.6: Un couple de sommets de type *début* et *fin* permet de représenter une valeur émise en fin de boucle et réutilisée en début de boucle. Les sommets de type *début* sont des sommets *entrée* particuliers. Les sommets de type *fin* peuvent être des sommets de type *sortie* particuliers.

Définition II.7: Les sommets de type *constante* représentent les constantes d'un GFC.

Définition II.8: La définition d'un *GDD ordonnancé* d'une boucle élémentaire est étendue en insérant des sommets de type début ou fin et entrée ou sortie.
Une paire de sommets début t_d et fin t_f est insérée sur chaque arc de retour a_r allant d'un sommet t_1 à un sommet t_2. L'arc de retour a_r est décomposé en deux arcs a_d et a_f, le premier allant de t_2 à t_d, le second allant de t_1 à t_f.
Un sommet entrée (respectivement sortie) est inséré pour chaque donnée définie (respectivement utilisée) à l'extérieur de la boucle et utilisée (respectivement définie) à l'intérieur de la boucle qui n'est pas encore représentée par un sommet de début (respectivement fin).

La date d'ordonnancement des sommets entrée et début est la valeur minimale de la date d'ordonnancement de l'ensemble des opérations de la boucle; celle des sommets sortie et fin est la valeur maximale de la date de l'ensemble des opérations de la boucle, incrémentée de 1.

Deux types de représentation des éléments virtuels symbolisant les données peuvent être insérés dans un GDD ordonnancé: les variables et les éléments de stockage virtuels.
La durée pendant laquelle une donnée doit être conservée est analysée afin d'établir le temps d'occupation du registre qui lui sera alloué et d'en déduire le nombre de registres nécessaires. Une donnée utilisée uniquement entre deux opérations élémentaires chaînées, ayant donc la même date d'ordonnancement, ne nécessitera pas d'être stockée dans un registre. Dans le cas de donnée émise par une opération élémentaire t multicycle (ou pipelinée), la donnée ne sera disponible qu'à la date Ord(t) + l(t) - 1. Par convention, la date d'ordonnancement d'un élément symbolisant une donnée émise par une opération t est Ord(t)+l(t).

Définition II.9: Une *variable* représente une donnée émise par une opération, à un pas de contrôle appelé la *date de début* de la variable, et lue par une autre opération, à un autre pas de contrôle appelé la *date de fin* de la variable. L'*intervalle de vie* d'une variable est l'intervalle situé entre sa date de début et sa date de fin.

Définition II.10: Un *élément de stockage* représente une donnée pendant un pas de contrôle. Une variable correspond donc à plusieurs éléments de stockage. Les sommets *début, fin, entrée* et *sortie* sont des éléments de stockage particuliers.

Nous avons choisi la représentation des données virtuelles par éléments de stockage (défintion II.10) pour deux raisons: d'abord parce que celle-ci permet d'exprimer des assignations prédéfinies de ressources; ensuite parce qu'elle autorise les transferts entre deux registres. Dans certains cas particuliers, cela permet d'économiser des connexions.
Dans un GDD ordonnancé G(T, U, Ord), les éléments de stockage sont insérés et créent le GDD $G(T \cup S, U \cup U_S, Ord)$ de la manière suivante:
Pour chaque sommet t_i soit l'arc u_i de U, d'origine t_i et de destination e_i, tel que $Ord(e_i)$ soit maximum, soient d_i et f_i respectivement définis par $Ord(t_i)+l(t_i)-1$ et $Ord(e_i)$, soit A_i l'ensemble des arcs sortant de t_i (excepté u_i):
- Créer successivement s éléments de stockage, numérotés de 1 à s et notés s_j, où s vaut f_i-d_i.
- Scinder l'arête u_i en s arêtes u_1, ..., u_s telles que u_1 va de t_i à s_1; u_2, ..., u_{s-1} notées u_j vont de s_j à s_{j+1}; u_s va de s_s à e_i. La date d'ordonnancement de s_j est $Ord(t_i)+l(t_i)-1+j$.
- Chaque arc a_i de A_i, soit ea_i sa destination, a_i a maintenant pour origine s_j, où $j=Ord(ea_i)-Ord(t_i)+1$.

La figure II.5 représente le GDD de l'équation différentielle correspondant au GP de la figure I.8, après ordonnancement orienté par les forces et en nommant les variables; celui de la figure II.6 représente le même GDD, sur lequel ont été insérés des éléments de stockage avec la méthode décrite ci-dessus. Dans ce graphe, les sommets rectangulaires représentent des opérations, les sommets ronds des sommets de début ou de fin et les sommets ronds et noirs des éléments de stockage; les sommets de constante ne sont pas encerclés.

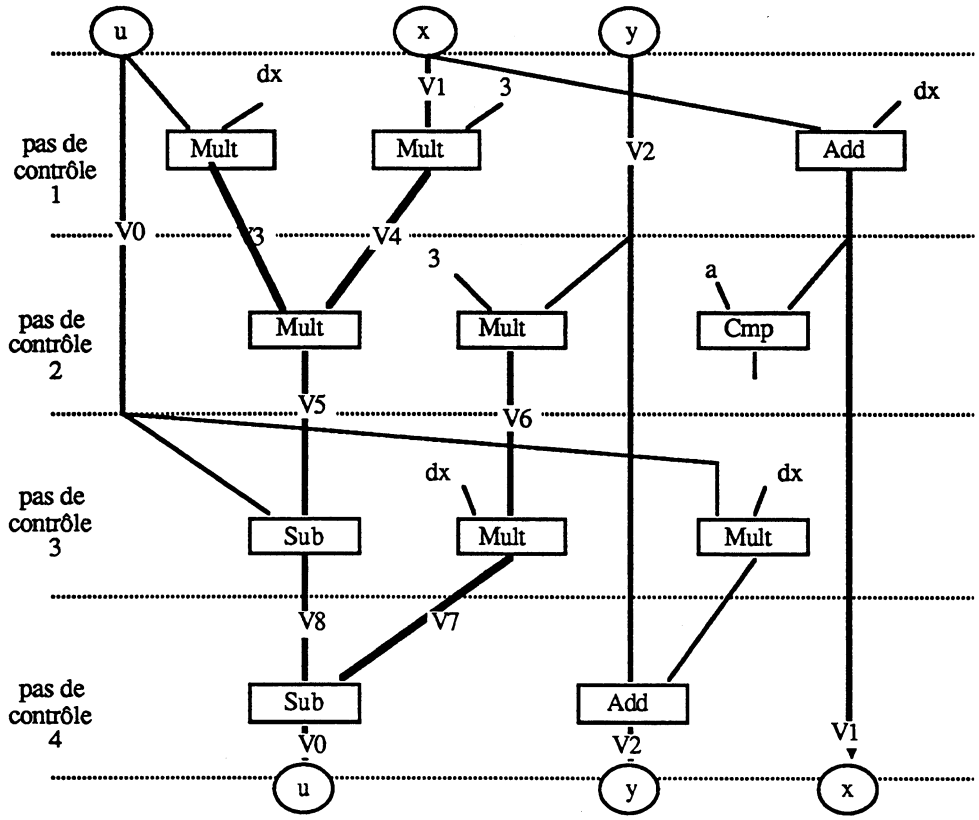


Figure II.5: GDD ordonnancé de l'équation différentielle de la figure I.8

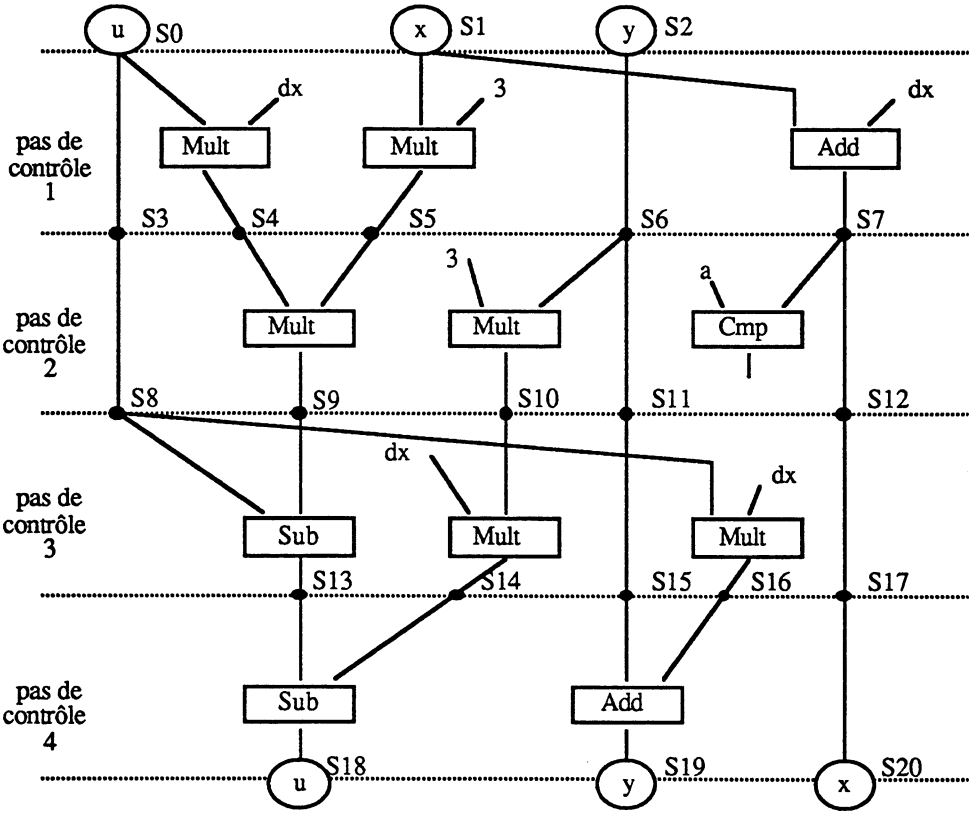


Figure II.6: GDD ordonnancé avec éléments de stockage, extrait de la figure I.8

A partir du choix d'une architecture cible, l'allocation de ressources doit donc définir le nombre de registres et d'opérateurs de chaque type et assigner les opérations aux opérateurs et les registres aux

éléments de stockage, en minimisant la surface de la partie opérative.

II.3. Détermination du nombre minimal de ressources

Le nombre de registres et d'opérateurs est fixé par l'utilisateur avant l'allocation. Par défaut, le nombre minimal de ressources est instancié. Cependant, l'utilisation de registres ou d'opérateurs supplémentaires permet parfois de réduire le coût en connexions. La première étape de l'allocation de registres et d'opérateurs consiste en la recherche du nombre minimal de ressources (Nb(op) définition I.21).

Dans notre contexte, le nombre de registres minimal est déterminé par le nombre maximal d'éléments de stockage ordonnancés au même pas de contrôle; le nombre minimal d'instances d'opérateurs est déterminé par la valeur finale des graphes de distribution (définition I.27).

Dans le cas général, le nombre minimal de registres peut être calculé en appliquant l'algorithme de l'arête gauche, adapté à la synthèse par [KURD 87]. En effet, cet algorithme détermine ce nombre pour un graphe de dépendance de données acyclique. La description initiale du GDD représentant le corps d'une boucle, le graphe est acyclique.

L'algorithme de l'arête gauche a initialement été proposé pour le routage dans les canaux. Pour son application à la détermination du nombre de ressources minimal, chaque variable est représentée par une connexion. Cette connexion a une origine et une destination correspondant à l'intervalle de vie de la variable. Les intervalles de vie des variables de la figure II.5 sont représentés figure II.7. Les pistes représentent les registres. L'algorithme classe les variables selon leur date de début, de gauche à droite, et place dans la première piste la variable commençant le plus tôt, donc celle se trouvant le plus à gauche (par exemple v_0). Puis, il place dans la même piste la prochaine variable commençant après les arêtes déjà placées dans la piste, donc l'arête la plus à gauche après l'arête déjà placée (par exemple v_7); et ainsi de suite jusqu'à ce que la piste soit remplie. Si toutes les variables n'ont pu être placées, l'algorithme crée une nouvelle piste et recommence. Cet algorithme donne un résultat optimal en nombre de registres dans le cas de graphes acycliques sans branchement conditionnel [KURD 87].

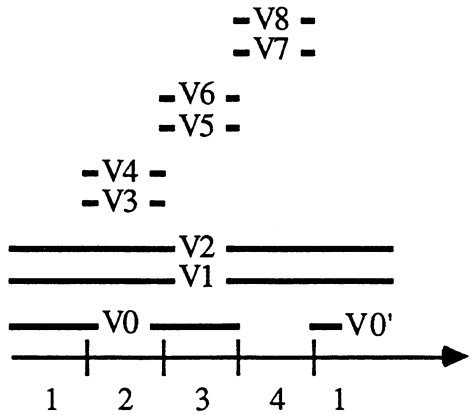


Figure II.7: Intervalles de vie des variables de la figure II.5

Le nombre minimal d'opérateurs se calcule de la même façon. Pour les opérations multicycles, l'intervalle d'une opération t est compris entre sa date d'ordonnancement Ord(t) et sa date de fin

d'exécution Ord(t)+l(t)-1, où l(t) est le délai en pas de contrôle de l'opération. Pour les opérations pipelinées, le délai considéré est celui de la latence de l'opération.

Par exemple, figure II.6, le résultat de l'application de l'algorithme de l'arête gauche sur le corps de la boucle fixe le nombre minimal de registres à 5. Le nombre maximal de multiplieurs par pas de contrôle est de 2.

II.4. Méthode d'assignation de ressources

L'assignation de ressources part donc d'un nombre de registres et d'instances d'opérateurs fixé. Cette étape assigne une ressource physique à chaque élément virtuel (opération ou élément de stockage définition II.10) du graphe de dépendance de données initial (GDD définition II.8).
Elle détermine les fonctions $ASSIGN_{ins}$ du chapitre précédent.

Pour l'assignation de ressources, nous proposons une approche originale qui est un compromis entre les approches d'assignations concurrentes de registres, d'opérateurs et de connexions, par traitement séquentiel avec évaluation locale, et les approches d'assignations indépendantes, par traitement global avec évaluation globale. Cette approche réalise simultanément l'assignation d'opérateurs et de registres par une optimisation globale par pas de contrôle. L'assignation de ressources à chaque pas de contrôle est un problème d'affectation d'éléments virtuels à des éléments physiques. Ce problème se modélise aisément par un problème de couplage maximal dans un graphe biparti. La recherche d'un couplage maximal n'est pas un problème NP-Complet. Les arêtes du graphe biparti sont créées et pondérées par l'application de règles. Un ensemble de règles, liant les éléments virtuels aux éléments physiques, permet de réaliser l'optimisation en vue de chaque style d'architecture. Cette approche a deux avantages: elle permet de réaliser l'assignation de ressources de manière spécifique à chaque style d'architecture et elle est fondée sur un problème à résolution polynomiale.

A chaque pas de contrôle, des registres sont assignés aux éléments de stockage virtuels d'entrée du pas de contrôle et des opérateurs aux opérations du pas de contrôle. La phase d'assignation construit au fur et à mesure un réseau de blocs simplifié (ne contenant pas de multiplexeurs) de la partie opérative. Ainsi, à chaque pas de contrôle, le résultat des assignations des pas de contrôle précédents est utilisé. Le réseau partiel de la PO permet d'estimer le coût de l'assignation d'une ressource donnée à un élément virtuel donné.
Un ensemble de règles d'assignation spécifique est associé à chaque style d'architecture. Les règles lient un ou plusieurs registres à chaque élément de stockage du pas de contrôle courant. Ces registres sont dits *registres candidats* pour l'élément de stockage considéré.

II.4.1. Règles d'assignation

Les règles d'assignation sont divisées en deux catégories: les règles de décision et les règles de pondération. Les règles de décision expriment une contrainte de l'architecture, tandis que les règles de pondération expriment des critères de cette architecture. Par conséquent, toutes les règles de décision doivent être respectées, tandis que les règles de pondération sont facultatives: le respect d'une règle de pondération permet de minimiser des critères. Chaque règle est définie par une *situation*, qui détermine les éléments de stockage auxquels cette règle est applicable, une *action*, qui détermine l'ensemble des registres candidats de chaque élément de stockage, et un poids dans le cas d'une règle de pondération. La pondération d'une règle dépend du nombre d'éléments de connexions et d'interconnexions qu'elle permet d'économiser.

Dans un premier temps, les règles et leur utilisation sont définies pour le cas d'une architecture à base de multiplexeurs. Les extensions aux autres styles d'architecture seront présentées section II.4.3.

Pour une architecture à base de multiplexeurs, une règle de décision permet d'éviter les connexions entre deux registres. Elle réduit à priori le nombre de multiplexeurs et d'interconnexions. Elle est définie de la manière suivante:

Règle 1: Connexions entre deux registres
- Règle de décision
- Situation: élément de stockage stockant une valeur n'ayant pas été modifiée pendant le pas de contrôle précédent.
- Action: le registre candidat est celui qui a été assigné au pas de contrôle précédent.

Par exemple, sur la figure II.8, les pas de contrôle 1 et 2 ont déjà été traités, l'algorithme traite le pas de contrôle 3. Les opérations de stockage à assigner sont S_8, S_9, S_{10}, S_{11} et S_{12}. L'élément de stockage S_8 répond à la situation décrite par la règle 1, le registre R_0 qui a été assigné au pas de contrôle précédent est donc candidat.

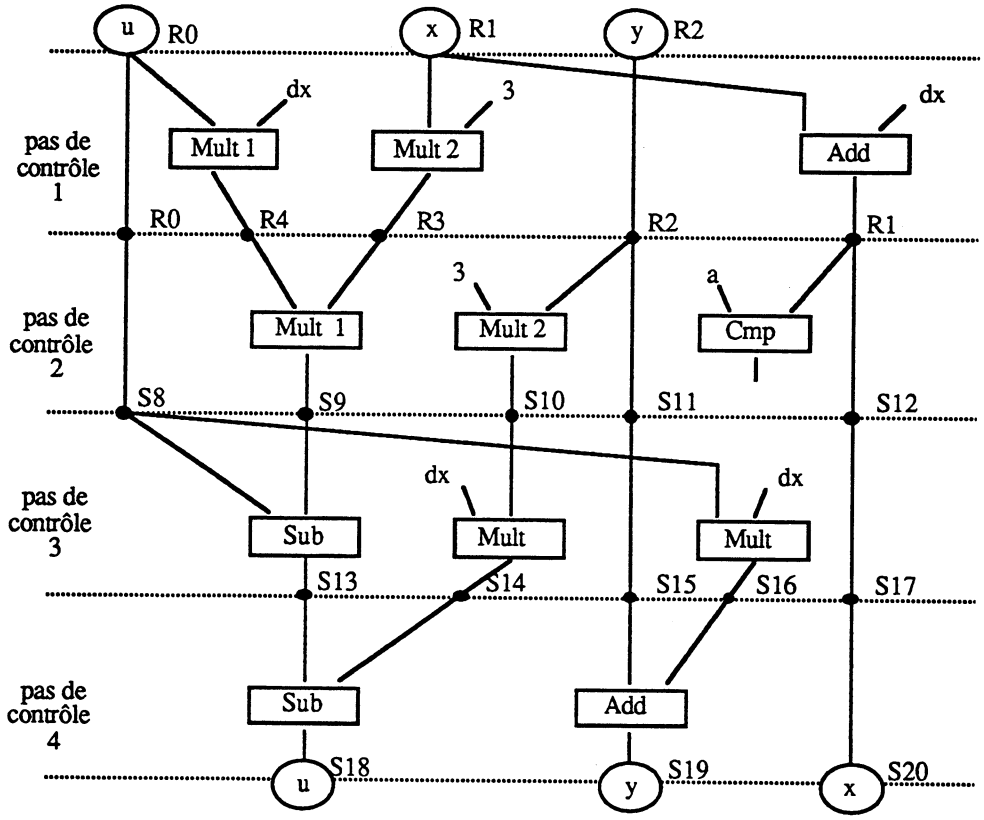


Figure II.8: Résultat partiel de l'assignation de la figure II.6

La règle 2 permet d'éviter les multiplexeurs et les interconnexions en entrée d'opérateurs et en entrée de registres. Elle se décompose en deux règles:

Règle 2a: Minimisation des multiplexeurs en entrée de registres
- Règle de pondération
- Poids: 2
- Situation: élément de stockage en sortie d'un opérateur
- Action: les registres candidats sont ceux déjà assignés en sortie de cet opérateur.

Par exemple, figure II.8, l'élément S_9 répond à la situation de la règle 2a, le registre R_4, déjà assigné à la sortie de $Mult_1$, est donc candidat.

Règle 2b: Minimisation des multiplexeurs en entrée d'opérateurs
- Règle de pondération
- Poids: 2
- Situation: élément de stockage à l'entrée d'un opérateur.
- Action: les registres candidats sont ceux déjà assignés à l'entrée de cet opérateur.

Remarque: Pour les opérateurs exécutant une opération commutative dans le pas de contrôle considéré, aucune distinction n'est faite entre les entrées de l'opérateur. Tous les registres déjà présents en entrée de cet opérateur sont candidats.

Par exemple, figure II.8, S_{10} répond à la situation de cette règle. En supposant que l'assignation d'opérateurs courante a affecté le multiplieur $Mult_1$ à la première multiplication, les registres R_4, R_3 et R_0 sont candidats; dans le cas inverse, $Mult_2$ a été affecté à la première multiplication, les registres R_1 et R_2 sont alors candidats.

La règle suivante permet d'éviter des interconnexions et de futurs multiplexeurs. Elle intervient pour l'initialisation de l'assignation lorsque tous les registres n'ont pas encore été assignés.

Règle 3: Initialisation des registres
- Règle de pondération
- Poids: 1
- Situation: élément de stockage en sortie (respectivement en entrée) d'un opérateur.
- Action: les registres candidats sont ceux non encore assignés en sortie (respectivement en entrée) d'aucun opérateur.

Cette règle a un poids inférieur à la règle 2. En effet, si un registre a déjà été assigné en sortie de l'opérateur considéré, ce registre est prioritaire par rapport à ceux candidats de la règle 3.

Le fait d'ajouter une nouvelle entrée à un multiplexeur existant est souvent moins coûteux en surface que le fait d'assigner un nouveau multiplexeur. La règle 4 permet d'éviter un nouveau multiplexeur: elle privilégie une entrée supplémentaire sur un multiplexeur existant, plutôt que l'ajout d'un multiplexeur. Elle est aussi divisée en deux règles.

Règle 4: Nouveaux multiplexeurs

Règle 4a:
- Règle de pondération
- Poids: 0,5
- Situation: élément de stockage en entrée d'un opérateur ayant déjà plus de trois registres en entrée.
- Action: tous les registres sont candidats.

Règle 4b:
- Règle de pondération
- Poids: 0,5
- Situation: élément de stockage en sortie d'un opérateur.
- Action: Les registres ayant déjà été assignés à la sortie de plus d'un opérateur sont candidats.

II.4.2. Algorithme d'assignation

Pour simplifier la description de l'algorithme, nous faisons l'hypothèse que le GDD représente une boucle élémentaire sans branchement conditionnel. L'extension aux cas de branchements conditionnels sera présentée ultérieurement, en section II.4.3.3.

Dans les exemples traités, le nombre d'opérateurs identiques est généralement faible (de l'ordre de 3 au maximum). Pour un ensemble O d'opérateurs op et Nb(op) opérateurs identiques de type op, N

assignations d'opérateurs différentes sont possibles, où N est défini de la manière suivante: $N = \prod_{op \in O} Nb(op)!$

Dans l'exemple figure II.6, le nombre d'assignations différentes est 2. Pour un exemple avec 1 soustracteur, 3 additionneurs et 3 multiplieurs (ce qui est déjà très complexe), le nombre d'assignations différentes est 36.

L'assignation d'opérateurs est donc réalisée exhaustivement. Pour chaque assignation d'opérateurs, l'assignation de registres est effectuée par une heuristique basée sur la définition des règles. Si cette assignation exhaustive est trop coûteuse en temps de calcul, l'heuristique décrite ultérieurement (section II.4.2) est appliquée successivement à l'assignation d'opérateurs et à l'assignation de registres.

Les règles lient des éléments de stockage à des registres candidats. Le résultat de l'application de chacune de ces règles est donc un graphe biparti d'assignation G.

Définition II.11: Un *graphe biparti d'assignation* est défini par $G(S_R(p) \times R, A_R, W)$, où $S_R(p)$ est l'ensemble des nœuds représentant des éléments de stockage du pas de contrôle courant, R est l'ensemble des nœuds représentant les registres et A_R est l'ensemble des arêtes du graphe. Une arête a_{ij} de A_R lie le nœud S_i de $S_R(p)$ au nœud R_j de R si R_j est un registre candidat pour S_i. Les arêtes de ce graphe sont pondérées par le poids auquel elles correspondent. Une assignation de registres pour le pas de contrôle courant est un couplage de ce graphe.

Les règles de décision expriment des contraintes de l'architecture. Par conséquent, l'algorithme applique d'abord les règles de décision une à une, puis l'ensemble des règles de pondération. Si, dans le graphe issu d'une règle de décision, il existe deux arêtes adjacentes, plusieurs assignations sont possibles. Aucune d'elles ne respecte toutes les contraintes, puisqu'une arête a été supprimée du graphe pour obtenir un couplage. Dans ce cas, l'algorithme essaye de remettre en cause une décision arbitraire dans les pas de contrôle précédents.

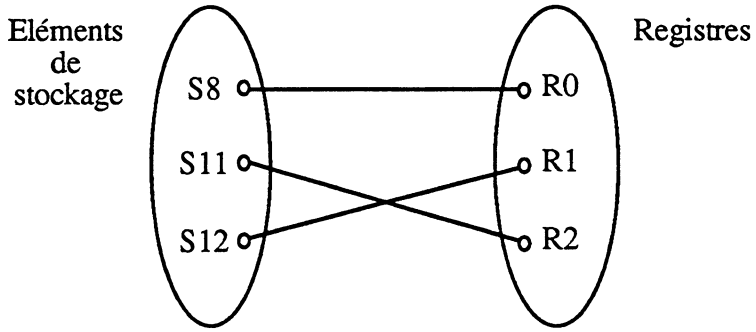
L'ensemble des règles de pondération est appliqué en même temps pour permettre une optimisation globale au niveau du pas de contrôle. Les graphes de chaque règle sont regroupés pour ne former qu'un seul graphe biparti. Une arête de ce graphe, reliant les sommets S_i et R_j, est pondérée par la somme des arêtes entre S_i et R_j des graphes bipartis résultant de chaque règle.

L'algorithme choisit ensuite l'assignation d'opérateurs qui minimise le nombre de ressources et d'interconnexions à rajouter. Cet algorithme est résumé figure II.9.

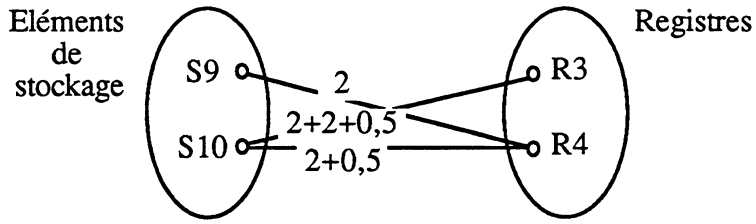
Pour chaque pas de contrôle p **faire**
  Pour chaque assignation d'opérateurs **faire**
    Pour chaque règle de décision **faire**
      Créer le graphe biparti $G(S_R(p) \times R, \emptyset)$
      Appliquer la règle de décision en créant les arêtes A de G
      Réaliser une affectation sur $G(S_R(p) \times R, A_R)$
      Assigner les éléments de stockage résultant de l'affectation
    FinPour
    Pour chaque règle de pondération **faire**
      Créer le graphe biparti $G(S_R(p) \times R, \emptyset, W)$
      Appliquer la règle de pondération en créant les arêtes A de G
    FinPour
    Réaliser une affectation sur $G(S_R(p) \times R, A_R, W)$
    Assigner les éléments de stockage résultant de l'affectation
  FinPour
  Choisir l'assignation d'opérateurs minimale
  Mettre à jour la netlist partielle de la PO
FinPour

Figure II.9: Algorithme d'assignation de ressources

Le graphe résultant de l'application de la règle de décision 1 au pas de contrôle 3 est représenté figure II.10(a). Il implique l'assignation des registres R_0, R_1 et R_2 aux éléments de stockage respectifs S_8, S_{12} et S_{11}. Ces registres ne sont plus candidats pour l'application des règles 2, 3 et 4. Le graphe résultant de l'application de ces trois dernières règles au pas de contrôle 3, (en supposant que l'assignation d'opérateurs courante affecte la première multiplication à $Mult_1$, la deuxième à $Mult_2$) est représenté figure II.10(b). Les registres R_3 et R_4 sont donc assignés respectivement aux éléments de stockage S_{10} et S_9.



(a) pour les règles de décision



(b)pour les règles de pondération

Figure II.10: Couplages de la figure II.8.

La figure II.11 montre le résultat de l'algorithme d'assignation de ressources pour les règles 1, 2, 3 et 4 d'une architecture à base de multiplexeurs.

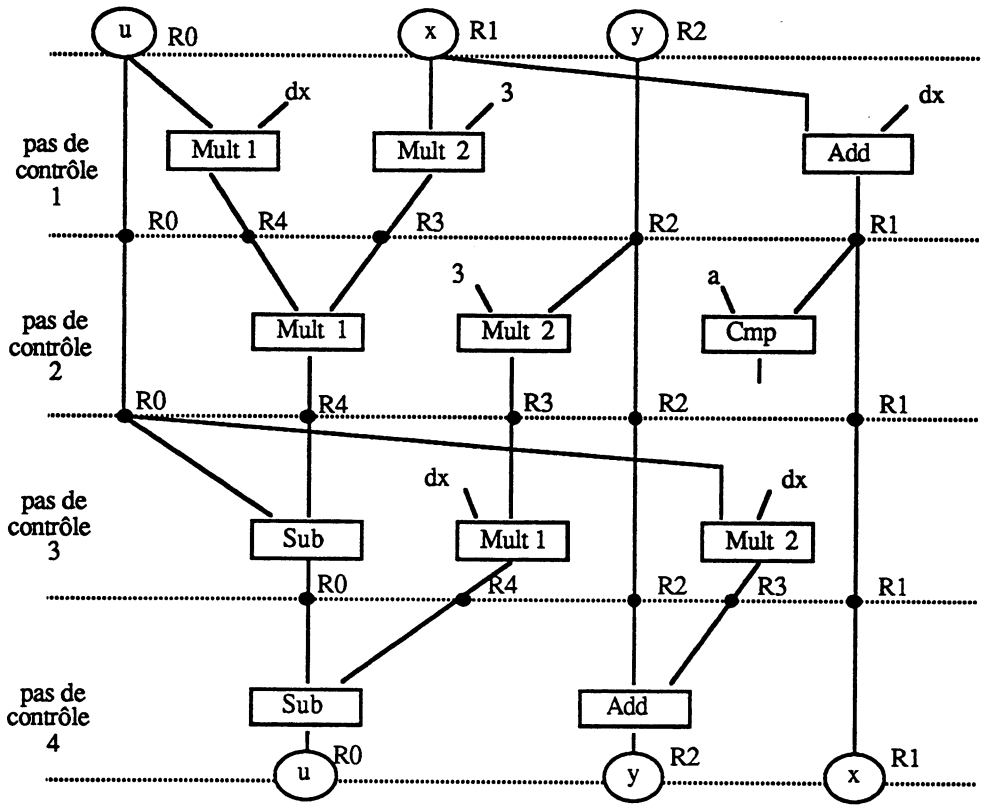


Figure II.11: GDD ordonnancé et assigné de la figure II.6 pour une architecture à base de multiplexeurs

II.4.3. Extensions

Nous venons de définir notre heuristique d'assignation de ressources pour le cas particulier d'une description sans branchement conditionnel. Nous allons étendre cet algorithme, afin qu'il puisse prendre en compte les cas de branchements conditionnels.

Cette heuristique est fondée sur la définition de règles spécifiques à chaque style d'architecture.

Nous avons énuméré les règles d'une architecture à base de multiplexeurs. Nous allons maintenant lister les règles des architectures à base de bus et des architectures mixtes et définir des règles pour prendre en compte la taille des registres.

II.4.3.1. Extensions aux autres architectures

Architectures à base de bus

Pour les architectures à base de bus considérées ici, des bus virtuels sont alloués à chaque opérateur de manière spécifique à chaque style d'architecture. Deux bus virtuels sont alloués par entrée d'opérateur pour les architectures à base de registres dédiés et les architectures à base de registres communs. De plus, un bus virtuel est alloué à chaque entrée de registre pour les architectures à registres communs.

Les architectures à base de bus avec registres communs ont exactement les mêmes règles d'assignation que les architectures à base de multiplexeurs. En effet, la règle 1 interdit d'allouer un bus pour relier deux registres; la règle 2a (respectivement 2b) permet de minimiser le nombre de

portes trois états en sortie d'opérateurs (respectivement de registres) et le nombre de multiplexeurs en entrée de registres (respectivement d'opérateurs); la règle 3 sert en phase d'initialisation; la règle 4 privilégie l'ajout d'une porte trois états à l'insertion d'un nouveau bus.

Les architectures à base de bus avec registres dédiés en entrée (respectivement en sortie) doivent privilégier les règles concernant des registres en entrée (respectivement en sortie) d'opérateurs afin de minimiser le nombre de registres à dupliquer par la suite. Ainsi la règle 2b (respectivement 2a) aura un poids plus fort que la règle 2a (respectivement 2b). Ce poids est de 5 pour la première règle contre 2 pour la seconde.

Pour l'exemple de la figure II.6, lors de l'assignation du troisième pas de contrôle, si l'assignation des pas précédents est celle de la figure II.8, après application des règles de pondération, la méthode d'assignation crée le graphe biparti de la figure II.12, pour une architecture avec registres dédiés en entrée, et celui de la figure II.13, pour une architecture avec registres dédiés en sortie. L'application des règles de décision n'est pas changée par rapport à l'architecture à base de multiplexeurs.

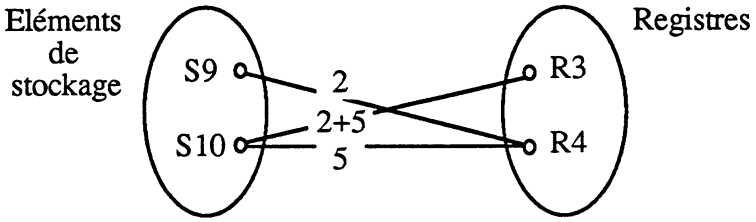


Figure II.12: Couplage de la figure II.8 pour une architecture à registres dédiés en entrée

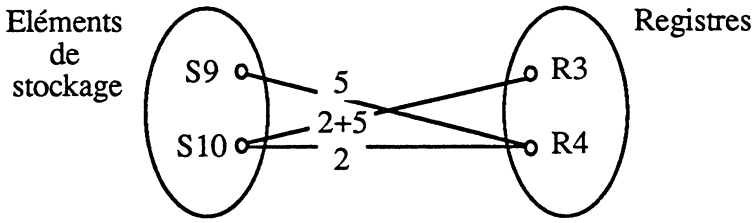


Figure II.13: Couplage de la figure II.8 pour une architecture à registres dédiés en sortie

Architectures mixtes irrégulières

Pour ce qui concerne les architectures mixtes irrégulières, les assignations de registres et d'opérateurs sont exécutées concomitamment à l'assignation de bus. En effet, le nombre de bus est alors fixé au début de l'assignation. Les premières phases d'assignation influent beaucoup sur l'assignation de bus. Un bus pourra facilement être assigné à l'entrée d'une ressource qui n'en a pas besoin, lors de décisions arbitraires en traitant le premier pas de contrôle. Nous proposons donc de réaliser l'assignation d'opérateurs et de registres sur l'ensemble du GDD ordonnancé, avant de commencer l'assignation des bus.

Un transfert de données est considéré comme allant d'une ressource origine vers plusieurs ressources destinations.

L'assignation de bus est réalisée avec la même méthode que l'assignation de registres. Il suffit donc

de définir des règles pour l'assignation des bus. La règle 5 suivante permet d'optimiser l'utilisation d'un nombre fixé de bus, en faisant en sorte que ceux-ci soient dédiés aux ressources (opérateurs ou registres). Cela permet de minimiser la longueur d'un bus, le nombre de multiplexeurs et le nombre de portes trois états.

Règle 5: Affectation des bus aux ressources
- Règle de pondération
- Poids: 2
- Situation: transfert de données en sortie (respectivement en entrée) d'une ressource.
- Action: les bus candidats sont ceux déjà assignés en sortie (respectivement en entrée) de cette ressource.

Une autre règle permet d'optimiser l'utilisation d'un nombre fixé de bus en assignant en priorité les bus devant une ressource ayant beaucoup d'entrées différentes (en distinguant les deux entrées d'un opérateur commutatif).

Règle 6: Assignation prioritaire de bus
- Règle de pondération
- Poids: 2/(M-x), où x est le nombre de ressources différentes à l'entrée des ressources de destination et M est le nombre maximal de transferts de données différents sur le pas de contrôle courant.
- Situation: transfert de données
- Action: tous les bus sont candidats

Le point de départ de l'assignation de bus est le résultat de l'assignation de registres et d'opérateurs pour une architecture à base de multiplexeurs, donné figure II.11. De ce résultat sont extraites les informations de la figure II.14. Lors du traitement du premier pas de contrôle, pour l'assignation de deux bus, le couplage de la figure II.15 est résolu. Il en résulte l'assignation d'un bus commun à l'entrée de Add et Mult1 et d'un bus commun à l'entrée de Add et Mult2.

	Nombre de transferts ≠ en entrée de
R0	1
R1	1
R2	1
R3	1
R4	1
Mult1	4
Mult2	5
Add	4
Sub	2
Cmp	2
Total	22

Figure II.14: Informations sur les transferts de données

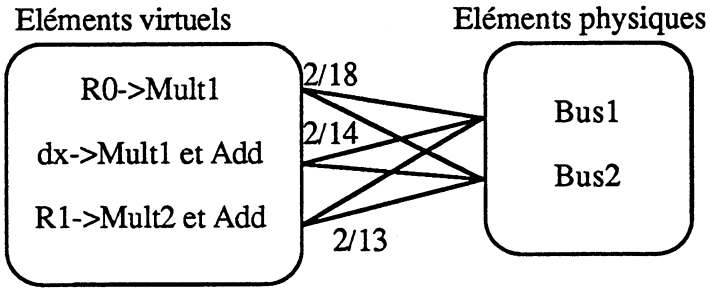


Figure II.15: Couplage de l'assignation de bus au premier pas de contrôle

Récapitulatif des règles par rapport aux style d'architecture

Le tableau II.0 résume l'ensemble des règles à appliquer pour chaque style d'architecture.

Tableau II.0: Récapitulatif des règles par rapports aux architectures

Chaque règle est référencée par son numéro. Les architectures sont à base de multiplexeurs (Multiplexeurs) ou mixtes (Mixte) ou à base de bus (Bus) avec registres dédiés en entrée (Entrée) ou dédiés en sortie (Sortie) ou communs (Communs). Ce tableau donne le poids de chaque règle de pondération pour une architecture donnée.

	Multiplexeurs	Mixte	Bus		
Règles			Entrée	Sortie	Communs
1: transferts	Décision	Décision	Décision	Décision	Décision
2a: # d'entrées	2	2	5	2	2
2b: # de sorties	2	2	2	5	2
3: Initialisation	1	1	1	1	1
4a: nouvel élément	0,5	0,5	0	0,5	0,5
4b:nouvel élément	0,5	0,5	0,5	0	0,5
5: affectation bus	0	2	0	0	0
6: bus prioritaire	0	2/(M-x)	0	0	0

II.4.3.2. Les extensions aux différents types de ressources

Pour l'algorithme d'assignation de ressources défini, tous les types d'opérateurs peuvent être pris en compte. Les opérateurs multicycles, les opérateurs pipelinés et les opérations chaînées sont pris en compte pendant l'insertion des éléments de stockage dans le GDD ordonnancé.

Les registres ont, jusqu'à présent, été considérés comme des registres maître/esclave. Un registre maître/esclave est un registre qui peut être lu et écrit dans le même cycle d'horloge, la valeur lue étant celle présente au début du cycle d'horloge avant écriture de la nouvelle valeur. S'il s'agit maintenant de registres non maître/esclave, ceux-ci sont pris en compte en réduisant la liste des registres candidats à ceux qui n'ont pas été lus dans le pas de contrôle considéré.

Lors de l'assignation de registres, les éléments de stockage et les registres sont parfois de différentes tailles. Dans ce cas, deux solutions sont envisageables: soit les registres nécessaires pour chaque taille d'élément de stockage sont instanciés, soit un ensemble minimal de registres est défini en supposant qu'une donnée de taille n peut être stockée dans un registre de taille m, si n est inférieur à m. Dans le

premier cas, une allocation de sous-ensembles de registres à des sous-ensembles d'éléments de stockage est définie implicitement. Cette allocation fait correspondre les registres et les éléments de stockage de même taille. Ensuite, l'assignation de registres est effectuée séparément pour chaque sous-ensemble. Dans le second cas, une règle permet de définir une assignation intéressante. Cette règle privilégie l'utilisation de registres de taille appropriée: un élément de stockage a pour candidat des registres de tailles similaires à la taille de cet élément de stockage, avec un poids plus fort que les registres de tailles très différentes. C'est la règle 7 suivante.

Règle 7: Tailles des registres
- Règle de pondération
- Poids: 1/(1+m-n)
- Situation: élément de stockage de taille n bit.
- Action: les registres candidats ont une taille de m bit.

II.4.3.3. Prise en compte des branchements conditionnels

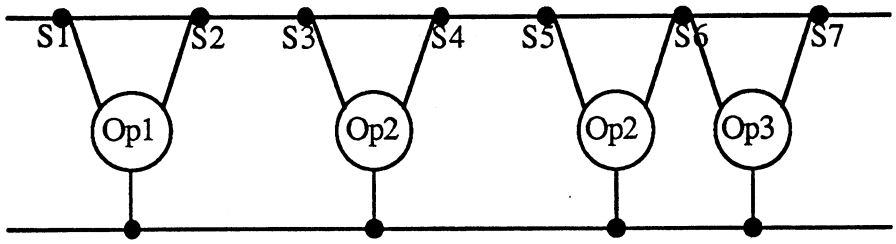
Dans le cas de branchements conditionnels, le système [HUAN 90] (qui utilise une méthode d'assignation similaire à la nôtre) propose un traitement de ces branchements fondé sur la définition d'un arbre AND/XOR.

Définition II.12: Un *arbre AND/XOR* au pas de contrôle *p* est un arbre A(AX, V), où a∈ AX, un nœud de l'arbre, est soit une feuille, soit un nœud XOR, soit un nœud AND. Chaque feuille représente une opération du pas de contrôle p. Un nœud particulier est identifié comme nœud racine de l'arbre. Les fils d'un nœud XOR sont des nœuds mutuellement exclusifs et les fils d'un nœud AND ne sont pas mutuellement exclusifs [HWAN 91] et [HUAN 90].

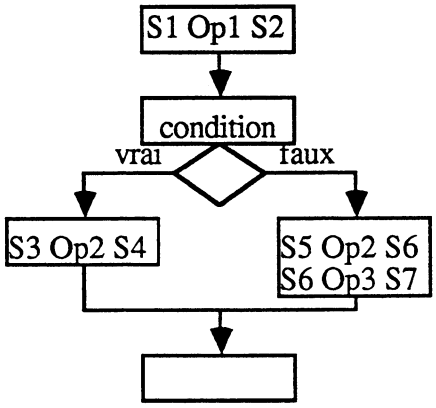
Cette approche consiste à parcourir l'arbre AND/XOR de haut en bas, jusqu'à rencontrer une feuille, puis traite successivement les nœuds de l'arbre, de bas en haut. Si un nœud AND est rencontré, les ressources sont assignées aux fils du nœud par un couplage entre les fils du nœud AND et l'ensemble des registres. Si, en revanche, un nœud XOR est rencontré, l'assignation de ressources est appliquée pour chaque fils du nœud XOR sur l'ensemble des ressources assignées au nœud XOR.
Soit, par exemple, à un pas de contrôle donné, le GDD ordonnancé de la figure II.17(a) et le GFC de la figure II.17(b) correspondant à la description initiale de la figure II.16, l'arbre AND/XOR de ce GDD est illustré figure II.18.

S_1 Op1 S_2
Si condition **alors**
    S_3 Op2 S_4
Sinon
    S_5 Op2 S_6
    S_6 Op3 S_7
FinSi

Figure II.16: Description initiale



(a) GDD ordonnancé de la figure II.16



(b) GFC de la figure II.16

Figure II.17: GDD ordonnancé et GFC contenant un branchement

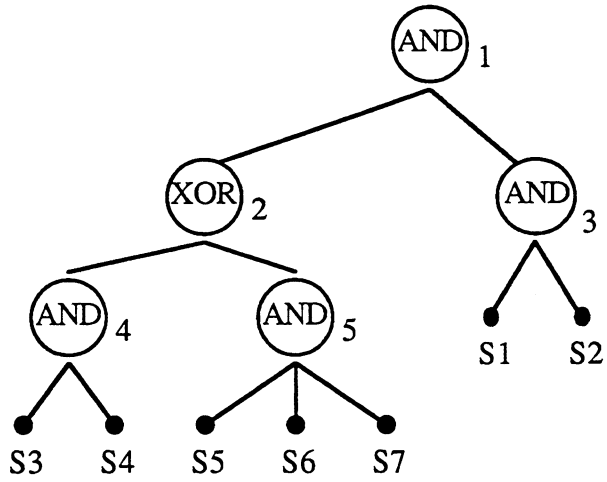


Figure II.18: Arbre AND/XOR correspondant à la figure II.17

Dans cet exemple, l'algorithme proposé par [HUAN 90] parcourt l'arbre de haut en bas et rencontre, par exemple, AND_1, puis XOR_2, puis AND_5, puis des feuilles. L'algorithme assigne donc en premier les fils du nœud AND_5. Supposons que le résultat de cette assignation affecte R_1 à S_5, R_4 à S_6 et R_6 à S_7. Le nœud XOR_2 est donc assigné à $\{R_1, R_4, R_6\}$. Lors du traitement du nœud AND_3, les fils de ce nœud ne pourront être assignés qu'aux registres restants R_2, R_3 ou R_5.

L'inconvénient de cette approche est qu'elle traite successivement les fils d'un nœud AND (dans l'exemple AND_1), quand ceux-ci contiennent à la fois un nœud XOR (dans l'exemple XOR_2) et une feuille ou un nœud AND (dans l'exemple AND_3). Autrement dit, c'est une méthode d'assignation

séquentielle et locale à chaque fils.

Dans le cas particulier de fils d'un nœud AND faisant intervenir à la fois un nœud XOR *xor* et une feuille ou un nœud AND *f_and*, nous proposons une méthode pour diviser l'ensemble des registres en deux sous-ensembles: l'un pour le nœud *f_and* et l'autre pour le nœud *xor*.

Définition II.13: Soit S_{Max} l'ensemble d'éléments de stockage du fils du nœud *xor* contenant le nombre maximal d'éléments de stockage, le nombre d'*éléments exclusifs* est égal au nombre d'éléments de stockage de S_{Max}. Il y a bijection entre l'ensemble des éléments exclusifs S_E et l'ensemble S_{Max}.

Chaque élément exclusif s_e de S_E représente un élément de stockage s_m de S_{Max} et l'ensemble des éléments de stockage qui sont mutuellement exclusifs à s_m.

Un élément exclusif représente plusieurs éléments de stockage mutuellement exclusifs. Dans l'exemple précédent, les éléments de stockage S_1 et S_2 ne sont pas mutuellement exclusifs, ils sont fils du nœud AND *f_and*. Les éléments S_5, S_6 et S_7 appartiennent au fils du nœud XOR *xor* ayant le plus d'éléments de stockage; ils créent donc les éléments exclusifs SE_5, SE_6 et SE_7, représentant respectivement S_5, S_6 et S_7 et l'ensemble des autres éléments de stockage mutuellement exclusifs à S_5, S_6 et S_7, c'est-à-dire S_3 et S_4. La définition du grahe biparti d'assignation (II.11) est étendue pour prendre en compte les branchements conditionnels.

Définition II.14: Le *graphe biparti local* d'un fils de *xor* est le graphe biparti créé par l'application de l'ensemble des règles sur les éléments de stockage du fils de *xor*. $G(b \times R, A_R, W)$, où b est l'ensemble des éléments de stockage du fils du nœud *xor* et W est la pondération usuelle des arêtes.

Définition II.15: Le *graphe biparti global* G est défini par $G((S_C \cup S_E) \times R, A_R, PoidsGlobal)$, où S_C est l'ensemble des nœuds représentant les éléments de stockage du nœud *f_and*, S_E est l'ensemble des éléments exclusifs du nœud *xor* , R est l'ensemble des nœuds représentant les registres et A_R est l'ensemble des arêtes du graphe.

Pour l'ensemble S_C, la création des arêtes et leur pondération restent les mêmes que dans le cas général (PoidsGlobal = W). Pour la pondération des éléments de S_E, les graphes bipartis locaux sont créés pour chaque fils du nœud *xor*. Une arête est créée entre s_e et chaque registre candidat au moins une fois R_i, du fait de l'application des règles; cette arête est pondérée par *PoidsGlobal(s_e, R_i)*, défini de la manière suivante:

$$PoidsGlobal(s_e, R_i) = 1/\|B\| \cdot (W(s_m, R_i) + \sum_{b \in B\text{-}S_{Max}} PoidsLocal(b, R_i)),$$

où B est l'ensemble des fils du nœud *xor*, et $\|B\|$ est le nombre de fils du nœud *xor*, PoidsLocal(b,r) est le poids du graphe local d'un des fils du nœud *xor*, et est défini par:

$$PoidsLocal(b, R_i) = 1/\|b\| \sum_{S_j \in b} W(S_j, R_i),$$

où $\|b\|$ est le nombre d'éléments de stockage de b et $W(S_j,R_i)$ est le poids de l'arête a_{ji}, allant de l'élément de stockage S_j au registre R_i, dû à l'application des règles par la méthode générale.

Dans l'exemple de la figure II.18, le poids d'une arête allant de SE_5 à un registre R_i est donc défini

par *PoidsGlobal(SE5, Ri)* de la manière suivante:

PoidsGlobal(SE_5,R_i) = 1/2 (W(S_5,R_i) + 1/2(PoidsLocal(S_3,R_i)+ PoidsLocal(S_4,R_i)))

La résolution d'un couplage de poids maximal sur le graphe biparti G(($S_C \cup S_E$) X R, A_R, PoidsGlobal), ainsi pondéré, implique une assignation de registres pour chacun des éléments de S_C et un sous-ensemble de registres pour l'ensemble des fils du nœud *xor*.

Après la résolution du couplage sur le graphe biparti global, il ne reste qu'à appliquer la méthode d'assignation générale sur chacun des fils du noeud *xor* séparément.

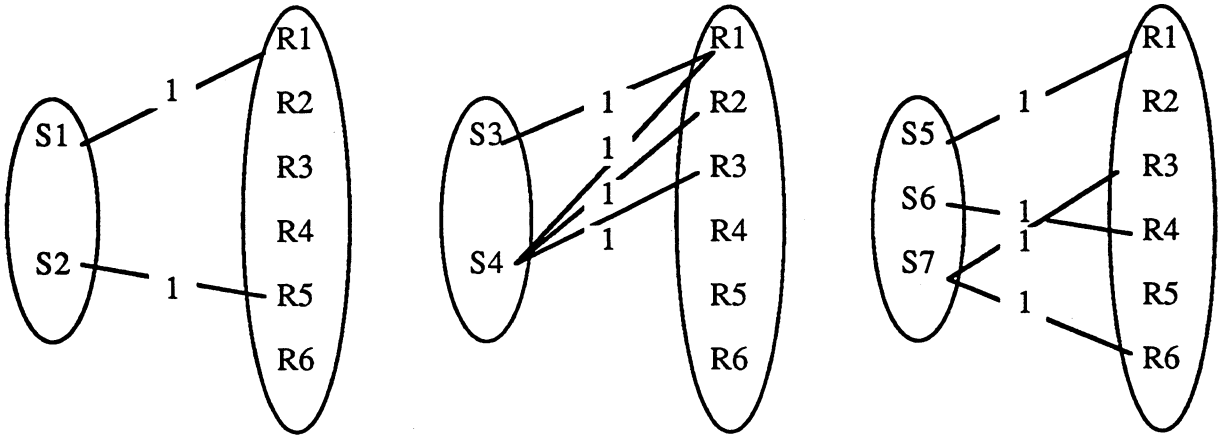


Figure II.19: Graphes bipartis locaux

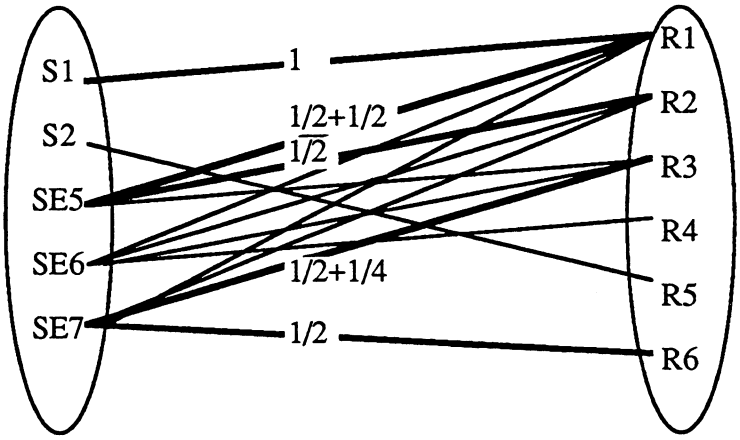


Figure II.20: Graphe biparti global

Dans l'exemple précédent, l'application des règles sur les éléments de stockage des fils du nœud XOR *xor* et du nœud AND *f_and* donne les graphes bipartis locaux de la figure II.19; ceux-ci ne servent qu'à construire le graphe biparti global G(($S_C \cup S_E$) X R, A_R, PoidsGlobal), illustré figure II.20. Le couplage maximal obtenu permet de diviser l'ensemble des registres en deux sous-ensembles {R_1, R_5} et {R_2, R_3, R_4, R_6}. Les registres R_1 et R_5 sont assignés respectivement à S_1 et S_2, la méthode générale d'assignation est ensuite appliquée pour les autres registres sur chaque fils du nœud XOR, en assignant à chaque fils l'ensemble des registres {R_2, R_3, R_4, R_6}.

La méthode décrite dans cette section, pour la prise en compte des branchements conditionnels, permet donc de diviser un ensemble de registres en sous-ensembles de registres qui seront assignés

séparément. Cette méthode a l'avantage de réaliser cette division de manière globale.

II.4.4. La complexité de l'algorithme

L'algorithme présenté dans ce chapitre est fondé sur la résolution d'un problème de couplage dans un graphe biparti. C'est un problème d'affectation. Ce problème est résolu par la méthode hongroise [PAPA 82]. Cette implantation a une complexité de $0(n^3)$, où n est le nombre de nœuds d'une des deux parties du graphe.

Soient m le nombre de pas de contrôle du GDD ordonnancé, r le nombre de registres, s le nombre d'éléments de stockage, f le nombre maximal d'opérations exécutées sur le même opérateur, o le nombre maximal d'opérateurs identiques et n le nombre d'opérateurs différents, si l'algorithme est appliqué à l'assignation d'opérateurs et à l'assignation de registres, la complexité de l'assignation d'opérateurs, à un pas de contrôle donné, est majorée par $0(n(\text{Max}(f,o)^3))$. La complexité de l'assignation de registres, à un pas de contrôle donné, est majorée par $0(\text{Max}(s,r)^3)$. La complexité de l'algorithme est donc majorée par $0(m(\text{Max}(n(\text{Max}(f,o)^3, \text{Max}(s,r)^3))))$. Le nombre de ressources physiques est toujours supérieur au nombre d'éléments virtuels ($f \leq o$ et $s \leq r$). Le plus souvent, le nombre d'assignations d'opérateurs est inférieur au nombre d'assignations de registres ($n.f^3 \leq r^3$).

Par conséquent, la complexité de l'algorithme est habituellement majorée par $0(m.r^3)$.

II.5. Autres méthodes pour l'allocation de ressources et étude comparative

Les allocations de connexions, de registres et d'opérateurs ont pour but de déterminer le nombre de ressources et d'assigner celles-ci aux éléments virtuels du GDD.
Les différentes étapes d'allocation sont interdépendantes. On distingue généralement deux classes de méthodes pour la réalisation de ces assignations: les méthodes d'assignations indépendantes et les méthodes d'assignations concurrentes de registres et d'opérateurs. L'assignation de registres et d'opérateurs affecte énormément l'allocation des connexions [FARL 87]. Par conséquent, ces phases prennent parfois en compte les connexions en les estimant pendant la phase d'assignation. Certaines méthodes vont jusqu'à allouer les connexions pendant l'assignation d'opérateurs et de registres.

II.5.1. Méthodes d'allocation indépendantes

Les allocations d'opérateurs, de registres et d'interconnexions réalisées de manière indépendante sont des problèmes NP-Complets. En effet, la section II.5.1.1 modélise ces problèmes sous forme d'un problème NP-Complet: le partitionnement en cliques.
Trois classes de méthodes de résolution des allocations indépendantes sont présentées: les méthodes basées sur le partitionnement en cliques ou un problème équivalent, les méthodes basées sur le couplage d'un graphe biparti et les méthodes basées sur la séparation/évaluation.

II.5.1.1. Partitionnement en cliques et modèles similaires

Les systèmes ayant une approche séparant chacune des étapes d'allocation utilisent le plus souvent des méthodes globales de résolution. Ces méthodes sont basées sur le partitionnement en cliques d'un graphe, pour les systèmes HAL [PAUL 89a, 89b], FACET [TSEN 86] et EASY [STOK 88] ou sur son problème dual: la k-coloration d'un graphe, pour le système CADDY [KRAM 89].
Ces méthodes sont utilisées pour l'allocation de registres, l'allocation d'opérateurs, l'allocation de multiplexeurs ou l'allocation de bus. Elles sont fondées sur la notion de compatibilité entre ressources.
Deux variables sont *compatibles* si elles peuvent partager le même registre, c'est-à-dire si leurs intervalles de vie sont disjoints. Dans le cas contraire, les variables sont *incompatibles*.
Dans le cas de multiplexeurs et de bus, la notion de compatibilité fait intervenir les registres qu'il faut multiplexer sur l'ensemble des pas de contrôle. Une définition de la compatibilité des multiplexeurs et des bus est donnée dans le chapitre III sur l'allocation de connexions.
Deux opérations monocycles sont compatibles si leur fonctionnalité permet de les exécuter sur le même opérateur et si elles ne sont pas ordonnancées au même pas de contrôle. La compatibilité peut être étendue aux opérations multicycles en considérant leur délai et en étendant la définition de compatibilité des variables.

Les méthodes fondées sur le partitionnement en cliques partent d'un graphe de compatibilité. Un *graphe de compatibilité* est un graphe G(V,E), où V est l'ensemble des nœuds du graphe représentant les éléments virtuels et où une arête de E existe entre deux nœuds si et seulement si les éléments qu'ils représentent sont compatibles.

Pour prendre en compte les connexions pendant l'allocation de registres ou d'opérateurs, les méthodes indépendantes d'allocation pondèrent le graphe de compatibilité, en estimant le coût en interconnexions de l'allocation de deux variables au même registre ou de deux opérations au même opérateur.

FACET [TSEN 86] est le premier système à avoir utilisé le partitionnement en cliques pour l'allocation. Il réalise l'allocation de registres, puis l'allocation d'opérateurs et enfin l'allocation des interconnexions. Pour l'allocation de registres, le graphe de compatibilité n'est pas pondéré. Pour l'allocation d'opérateurs, il est pondéré de 1 à 4, selon le nombre de registres communs en entrée et en sortie des deux opérations à regrouper. Pour l'allocation des interconnections, deux nœuds, représentant deux transferts de données, sont reliés par une arête si les transferts ne sont pas réalisés au même instant. Cette arête est pondérée de 0 à 2 selon le nombre de sources et destinations communes aux deux transferts.

Le système EASY [STOK 88] commence par l'allocation d'opérateurs en pondérant les arêtes par une fonction coût "avantage". Cette fonction calcule la différence de coût entre l'implantation séparée des deux opérations et l'implantation sur un même opérateur. Pour l'allocation de registres, EASY applique l'algorithme de l'arête gauche, défini section II.3, en pondérant les intervalles au moment d'en choisir un parmi plusieurs commençant au même instant. EASY réalise ensuite l'allocation de bancs de registres en regroupant plusieurs registres dans le même banc. Les bancs de registres sont supposés n'avoir qu'un port de lecture et un port d'écriture. Une arête existe entre deux registres si ceux-ci sont compatibles, c'est-à-dire s'ils ne sont jamais lus ou écrits en même temps. Ce graphe est pondéré par le nombre de fois où les valeurs lues dans les registres ont la même source ou la même destination.

Le système HAL [PAUL 89a], quant à lui, réalise l'allocation d'opérateurs par un système expert. Pour l'allocation de registres, HAL définit quatre configurations de regroupement de deux nœuds du graphe de compatibilité, pondérées de 1 à 4. Pour l'allocation des multiplexeurs, la pondération dépend du nombre d'entrées communes entre les multiplexeurs.

Les méthodes basées sur la k-coloration d'un graphe partent d'un graphe d'incompatibilité. Un *graphe d'incompatibilité* est un graphe G(V, $\bar{\bar{E}}$) où les nœuds sont des éléments virtuels et une arête existe entre deux nœuds si les éléments virtuels qu'ils représentent sont incompatibles. L'ensemble d'arêtes $\bar{\bar{E}}$ est donc l'ensemble complémentaire de E.

L'allocation de ressources minimale en nombre de ressources sera obtenue par la coloration de ce graphe par un nombre de couleurs minimal, tel que deux arêtes adjacentes n'aient jamais la même couleur. Le système CADDY [KRAM 89] réalise les allocations dans le même ordre que FACET, mais sans pondération, ce qui permet de définir le nombre de couleurs. Puis, la coloration est recommencée en utilisant la pondération. Cette pondération est prise en compte en construisant un

graphe entre les variables, les opérations ou les connexions, dont les arêtes sont pondérées en fonction du nombre de sources et de destinations communes.

Le problème de partitionnement en cliques est un problème NP-Complet. Les auteurs définissent donc une heuristique adaptée à la structure particulière du graphe construit pour résoudre ce problème [SPRI 90]. Le système FACET utilise une heuristique qui commence par regrouper les nœuds ayant un nombre maximal de voisins communs. L'heuristique de résolution du problème dual, la k-coloration d'un graphe, commence par les nœuds de plus fort degré. Dans HAL, le partitionnement est résolu en commençant par regrouper les nœuds reliés par l'arête de plus forte pondération. REAL [KURD 87] utilise l'algorithme de l'arête gauche; c'est une heuristique de résolution du problème de k-coloration d'un graphe d'intervalle [SPRI 90] ou d'un graphe d'incompatibilité.
Pour l'ensemble des méthodes décrites précédemment, le coût des connexions est estimé, puisqu'on ne peut pas le connaître avant l'assignation d'opérateurs et de registres et l'allocation des connexions. De plus, la résolution du problème est réalisée par une heuristique, puisque les modèles de résolution sont NP-Complets. Enfin, ces méthodes regroupent des variables, des opérations ou des connexions pouvant partager la même ressource physique; elles ne réalisent pas, à proprement parler, d'assignation de ressources. Par conséquent, si les critères spécifiques de l'utilisation de registres dédiés peuvent être pris en compte, ce n'est pas le cas des pré-assignations de ressources ni de l'utilisation prioritaire de certaines ressources. Les méthodes de résolution globale ne sont pas assez flexibles pour prendre en compte tous les critères et contraintes des différents styles d'architecture.

II.5.1.2. Séparation/évaluation

Le système FLORA [LUI 91] réalise successivement l'assignation d'opérateurs, l'allocation de bancs de registres et l'allocation des connexions, par une méthode de séparation/évaluation. Les fonctions coûts expriment la minimisation du nombre de connexions, du nombre d'intervalles de vie se chevauchant et du nombre de conflits d'accès en lecture ou écriture sur les variables en entrée d'opérateurs. Ces fonctions coûts exprimant la réduction de plusieurs critères, des coefficients de pondération doivent être fixés. De plus, des paramètres permettent de définir la profondeur d'exploration de la méthode de séparation/évaluation.
Ce type de méthode est difficile à mettre au point du fait du nombre de paramètres à définir (15 pour les fonctions coûts, 3 pour la profondeur d'exploration). D'après les auteurs eux-mêmes, cette méthode est une bonne méthode pour l'obtention d'une solution initiale avant l'application d'une méthode itérative.

II.5.1.3. Couplage

Le système LYRA/ARYL [HUAN 90] réalise successivement l'assignation de registres et d'opérateurs (LYRA) ou d'opérateurs et de registres (ARYL) en formulant le problème comme un couplage de poids minimal à chaque pas de contrôle.
Le point de départ de l'assignation est un GDD en termes de variables. La pondération des arêtes, pour l'assignation de registres, dépend:
• du fait qu'aucune des variables déjà assignées au registre considéré R_i n'est à l'entrée du même type

d'opération o que la variable considérée v_j,

• du nombre de variables différentes assignées à l'entrée des opérateurs, si R_i est assigné à v_j,

• du nombre de registres en entrée d'opération dont la sortie est un registre déterminé, si le registre R_i est assigné à la variable v_j.

Pour l'assignation d'opérateurs, la pondération des arêtes dépend du nombre de multiplexeurs générés par l'assignation considérée. La formulation décrite dans [HUAN 90] ne prend pas en compte les opérateurs multicycles.

II.5.2. Méthodes concurrentes

Les méthodes traitant simultanément différents problèmes d'allocation sont souvent constructives et séquentielles. On distingue l'assignation séquentielle par pas de contrôle de celle séquentielle par élément virtuel à assigner, c'est-à-dire par opération, variable et transfert.

II.5.2.1. Assignation séquentielle par pas de contrôle

MIMOLA [BALA 88] réalise l'assignation de registres et d'opérateurs par pas de contrôle en même temps que l'ordonnancement. A chaque pas de contrôle, les opérateurs puis les registres sont alloués successivement, en estimant de manière globale, sur un pas de contrôle, le coût des assignations. Le problème d'assignation est modélisé par un programme linéaire en 0-1, où une variable x_{ij} est égale à 1 si l'élément virtuel i est assigné à l'élément physique j. Les contraintes de ce programme linéaire expriment le fait qu'un seul élément physique peut être assigné à un élément virtuel. La fonction coût est la somme des coûts de chaque assignation. Le coût d'une assignation est le nombre d'interconnexions supplémentaires qu'elle implique. MIMOLA se ramène à un problème NP-Complet, alors que ce système a choisi de décomposer l'assignation de ressources en pas de contrôle, ce qui devrait permettre de résoudre ce problème de manière polynomiale.

II.5.2.2. Assignation séquentielle par élément virtuel

Les systèmes EMUCS [HITC 83], ELF [LY 90] et MABAL [KUCU 89] réalisent l'allocation en traitant chaque élément virtuel successivement.

Concernant MABAL, pour chaque élément virtuel, différentes stratégies d'assignation sont essayées et la moins coûteuse en surface est sélectionnée. Ces stratégies tentent d'ajouter des ressources ou d'utiliser celles déjà allouées, d'utiliser des bus ou des multiplexeurs, d'utiliser un opérateur libre pour le transfert d'une variable, ...

Le système ELF utilise une approche similaire pour l'allocation des connexions.

Dans l'approche d'EMUCS, les éléments virtuels sont assignés de manière séquentielle. A chaque itération, un tableau de coûts des assignations est construit. Ce tableau représente les éléments virtuels par des colonnes et les éléments physiques par des lignes. Si un élément virtuel peut être assigné à un élément physique, la case correspondante du tableau prend pour valeur le coût associé à cette assignation. Ce coût exprime à la fois le coût de l'assignation courante et le coût de futures assignations. L'assignation de plus petit coût est sélectionnée.

Les méthodes séquentielles ont l'inconvénient de dépendre de l'ordre de traitement des éléments

virtuels à assigner.

II.5.3. Récapitulatif

Les méthodes d'allocation de ressources sont classifiées selon qu'elles effectuent:

- la réalisation simultanée ou indépendante de l'allocation d'opérateurs, de registres et de connexions;
- l'allocation par une méthode globale ou constructive (par pas de contrôle ou par élément virtuel);
- l'allocation de ressources (détermination de Nb(op) et ASSIGNins) ou seulement l'assignation de ressources;

et selon leur capacité à prendre en compte:

- différents critères (coût des connexions, registres dédiés, utilisation prioritaire de certaines ressources) ou contraintes (ressources pré-assignées);
- différents types de ressources (opérateurs multicycles, registres non maître/esclave, ...)

Le tableau II.1 suivant récapitule les méthodes d'ordonnancement et les classes auxquelles elles appartiennent.Dans ce tableau, *PL 0-1* dénote les méthodes par programmation linéaire en nombres entiers, *k-colorat* les méthodes par k-coloration.

Tableau II.1: Récapitulatif des méthodes d'allocation de ressources

	Clique k-colorat.	Séparat. / évaluat.	Couplage LYRA	Méthodes Locales	PL 0-1	Couplage ASYL
Allocations indépendantes	Oui	Oui	Oui		Oui	
Allocations simultanées	Non			Oui		Oui
Méthode Globale	Oui	Oui				
" par pas de contrôle			Oui		Oui	Oui
" par élément virtuel				O		
Nb(op)	Oui	Fixé	Fixé	Oui	Fixé	Fixé
Registres dédiés						Oui
Pré-assignation	Non					Oui
Utilisation prioritaire	Non				Non	Oui
Type de ressources			Non			Oui
Systèmes utilisant ces méthodes	HAL FACET EASY CADDY REAL	FLORA	LYRA ARYL	EMUCS MABAL	MIMOLA	ASYL

Les méthodes globales, telles que le partitionnement en cliques ou la k-coloration d'un graphe, déterminent et allouent un nombre minimal de ressources. Les méthodes locales et séquentielles par élément virtuel déterminent et allouent un nombre de ressources, mais qui peut être différent du minimum. A l'opposé, les méthodes séquentielles par pas de contrôle, telles que celles utilisées par

MIMOLA, ASYL et LYRA, partent d'un nombre déterminé de ressources. Cela permet parfois de diminuer le nombre de connexions futures. Ces méthodes permettent d'assigner n'importe quel ensemble de registres.

Les méthodes par partitionnement en cliques pourraient prendre en compte les registres dédiés en changeant la pondération des arêtes, mais en aucun cas elles ne peuvent prendre en compte l'utilisation prioritaire de certaines ressources ou la pré-assignation de ressources. La méthode fondée sur un couplage, implantée dans LYRA/ARYL, ne traite pas le cas d'opérateurs multicycles ou pipelinés. La formulation en programme linéaire de MIMOLA ne permet pas d'exprimer l'utilisation prioritaire de ressources spécifiques. Les méthodes par séparation et évaluation ou les méthodes séquentielles par élément virtuel à assigner pourraient intégrer l'ensemble des styles d'architecture, mais cette intégration reste à définir. La méthode d'assignation par couplage, que nous avons présentée, est identifiée par le nom ASYL; c'est la seule à intégrer explicitement tous les styles d'architecture de partie opérative.

Nous avons écarté les méthodes globales basées sur le partitionnement en cliques ou sur la k-coloration d'un graphe, puisqu'elles ne sont pas assez flexibles pour prendre en compte tous les styles d'architecture. En effet, celles-ci réalisent une allocation de registres et regroupent plusieurs éléments de stockage pour partager une même ressource.Ces méthodes ne peuvent donc pas prendre en compte une préassignation de ressources. De plus, ces méthodes cherchent systématiquement une allocation sur le nombre minimum de registres. Or, l'utilisation de registres supplémentaires peut diminuer le nombre de connexions et dans certains cas une connexion est plus coûteuse qu'un registre. A l'opposé, les méthodes purement séquentielles, telles que MABAL ou EMUCS, ont l'inconvénient de dépendre de l'ordre de traitement des éléments à assigner.

L'originalité de la méthode d'assignation de ressources que nous avons proposée est liée à la possibilité de cibler différents styles d'architecture. Les optimisations réalisées pendant l'assignation visent une implantation efficace du style d'architecture de la partie opérative choisi. Pour chaque style, des critères ou des contraintes spécifiques sont définis. Etant entendu que le problème d'allocation de ressources est un problème NP-Complet, une heuristique efficace en $o(c.n^3)$ a été définie. Notre méthode a une approche flexible, qui ne nuit pas à l'optimisation des ressources.

II.6. Résultats

Notre algorithme a été implanté sur SUN 4 en langage C. Trois exemples classiques de la communauté internationale, ordonnancés par la méthode orientée par les forces, décrite au chapitre précédent, ont été traités: l'exemple du filtre du 5ème ordre, celui du filtre FIR et celui de l'équation différentielle.

Pour comparer de manière impartiale les résultats de cette assignation, il faudrait partir du même résultat d'ordonnancement et comparer le nombre de connexions, de multiplexeurs et de portes trois états. Cependant, mis à part les systèmes HAL et REAL, chaque système a son propre résultat d'ordonnancement. De plus, après l'assignation de ressources, les connexions ne sont pas encore allouées. Il n'est donc pas possible d'évaluer une méthode d'assignation de registres par le nombre d'interconnexions, de multiplexeurs et de portes trois états qu'elle implique. Dans la littérature, le nombre de connexions est souvent estimé par le nombre d'entrées différentes des opérateurs et des registres.

En effet, en ne tenant pas compte du partage des connexions, le nombre de registres (respectivement opérteurs) différents en entrée d'un opérateur (respectivement registre) induira la taille du multiplexeur, ou le nombre de portes trois états sur le bus, en entrée de cet opérateur (respectivement registre). Ce nombre estime aussi le nombre d'interconnexions allant jusqu'au multiplexeur ou au bus en entrée de cette ressource. Les résultats des différentes méthodes seront donc comparés en fonction du nombre d'entrées différentes qu'elles génèrent.

Le tableau II.2 suivant comparent nos résultats à ceux de HAL [PAUL 89b], REAL [KURD 87], ELF [LY 90], MIMOLA [BALA 88], ARYL/LYRA [HUAN 90], FACET [TSEN 83], MABAL [KUCU 89] et EMUCS [HITC 83] sur l'exemple du filtre du 5ème ordre. Les résultats de REAL ont été calculés manuellement. Les résultats de FACET sont extraits de [TUTO 90]. Pour l'exemple de l'équation différentielle ordonné sur 4 pas de contrôle, notre algorithme donne 11 entrées différentes en entrée des ressources.

Les résultats de l'algorithme d'assignation de ressources, présentés dans les sections précédentes, sont référencés par l'appellation ASYL, qui est le système de synthèse développé au laboratoire CSI.

Tableau II.2: Résultats d'assignation pour le filtre de 5ème ordre,

en nombre de pas de contrôle (pas de contrôle), d'opérateurs (Opérateurs) additionneur (+), multiplieur (*) et multiplieurs pipelinés (*p), de registres et de ressources différentes en entrée de ressources (Entrées) pour chaque système et gain du système ASYL (ASYL) en nombre d'entrées par rapport à chaque système

Système	pas de contrôle	Opérateurs	Registres	Entrées	ASYL
REAL	17	2 *p, 3 +	10	50	- 12
ASYL	"	"	"	38	
ELF	"	"	11	28	- 3
ASYL	"	"	"	25	
HAL	"	"	12	31	- 7
ASYL	"	"	"	24	
REAL	"	3 *, 3 +	10	52	- 9
ASYL	"	"	"	43	
ASYL	"	"	11	27	
MIMOLA	"	"	12	32	- 6
ASYL	"	"	"	26	
REAL	19	1 *p, 2 +	10	35	- 5
ASYL	"	"	"	30	
FACET	"	"	11	27	+ 1
ELF	"	"	"	30	- 2
ASYL	"	"	"	28	
HAL	"	"	12	26	0
ASYL	"	"	"	26	
REAL	"	2 *, 2 +	10	39	- 6
ASYL	"	"	"	33	
ELF	"	"	11	30	+ 1
MIMOLA	"	"	"	30	+ 1
ASYL	"	"	"	31	
HAL	"	"	12	29	+ 1
EMUCS	"	"	"	34	- 6
ASYL	"	"	"	28	
FACET	"	"	14	28	- 1
ARYL-LYRA	21	1*, 2 +	10	30	- 9
ASYL	"	"	"	21	
MABAL	"	"	11	43	- 21
ELF	"	"	"	24	- 2
ASYL	"	"	"	22	

De ce tableau est déduit le tableau récapitulatif II.4.

Tableau II.4: Récapitulatif des résultats

Gain en nombre d'entrées du système ASYL par rapport aux autres systèmes (% d'entrées gagnées) et le nombre de comparaisons effectuées pour obtenir ce gain (Nb Comp).

Système	Nb Comp	% d'entrées gagnées
HAL	2	16.0 %
MIMOLA	2	4.5 %
ELF	4	3.5 %
ARYL-LYRA	1	42.0 %
REAL	4	21.3 %
FACET	2	- 7.0 %
MABAL	1	95.0 %

L'algorithme d'assignation d'ASYL peut réaliser l'assignation de registres avec n'importe quel

nombre de registres et en particulier avec le nombre minimal. Le nombre de connexions obtenues par ASYL, estimé par le nombre d'entrées différentes devant chaque ressource, est nettement meilleur que celui obtenu par les méthodes locales assignant des registres (MABAL, ELF). De plus, il est meilleur que celui résultant de méthodes ne cherchant pas à minimiser les connexions pendant l'assignation (REAL), ce qui n'est pas surprenant. ASYL a des résultats similaires par rapport aux méthodes de résolutions globales (HAL, FACET) et par rapport à une méthode séquentielle par pas de contrôle (MIMOLA). Comparée à une approche similaire, celle de ARYL/LYRA, les résultats n'ont pu être évalués que sur un exemple, puisque ce dernier système ne traite pas les cas d'opérateurs multicycles. Le résultat d'ASYL est meilleur sur cet exemple.

La méthode proposée a donc de bons résultats par rapport aux autres méthodes étudiées, pour une architecture à base de multiplexeurs. Pour les autres architectures, les différents systèmes présentés n'ont pas de fonction d'optimisation particulière. Il est donc vraisemblable qu'alors ASYL aura de meilleurs résultats.

En particulier, pour ce qui concerne les architectures avec registres dédiés, le nombre de registres qu'il faudra dupliquer dépend de l'assignation de ressources. L'algorithme décrit ici cherchera à dédier les registres et obtiendra forcément de meilleurs résultats que tous les systèmes réalisant une allocation uniforme de tous les registres.

II.7. Conclusion sur l'assignation de ressources

En principe, chaque système de synthèse architecturale définit un domaine d'application et le style d'architecture cible que celui-ci implique. A titre d'exemple, le système Cathedral II [DEMA 86], dédié aux processeurs de traitement numérique du signal, utilise une architecture cible formée d'un contrôleur à base de ROM [ZEGE 90] et d'une partie opérative à base de bus et de bancs de registres dédiés en entrée d'opérateurs. Le système Cathedral III [NOTE 91], dédié aux mêmes processeurs mais à haut débit de fonctionnement, utilise une architecture formée d'un contrôleur central et de plusieurs séquenceurs, chacun dédié à une partie de la partie opérative. Le système AMICAL [PARK 92] orienté vers le contrôle utilise une architecture cible à base de deux bus segmentés.
Nous avons choisi de ne pas être dédié à un domaine d'application spécifique du point de vue du style d'architecture de la partie opérative. Dans ce but, en section II.1, certaines architectures de partie opérative ont été définies. Ce sont les architectures à base de multiplexeurs, les architectures mixtes et les architectures à base de bus parallèles qui peuvent avoir des registres dédiés en entrée ou en sortie d'opérateurs ou des registres communs avec ou sans registres tampons en entrée et/ou en sortie d'opérateurs.
L'algorithme présenté dans ce chapitre réalise simultanément l'assignation de registres et d'opérateurs, en procédant par pas de contrôle. Sur un pas de contrôle, l'algorithme optimise l'assignation de manière globale en la modélisant par un problème polynomial de couplage de poids maximal. Cet algorithme réalise une optimisation dédiée à chaque style d'architecture, en s'appuyant sur la définition de règles liant les éléments virtuels aux éléments physiques. Avec une complexité de $0(c.r^3)$, les résultats obtenus sont bons, comparés à ceux des différents systèmes réalisant l'assignation de ressources. L'intérêt principal de la méthode définie reste sa grande flexibilité. Tous les styles d'architecture définis dans cette thèse ont pu être pris en compte, aussi bien les opérateurs complexes, les connexions régulières ou irrégulières.que les préassignations de registres.
Cependant , la liste des styles définis n'est pas exhaustive. Un domaine d'application spécifique peut impliquer un style d'architecture spécifique. Cette architecture est définie par un ensemble de critères et de contraintes. Ces critères et ses contraintes doivent être prises en compte durant l'assignation de ressources comme les contraintes des architectures déjà définies.
Ce type d'approche a été utilisé pour le développement d'un système de synthèse dédié au traitement d'images de bas niveau [BOUB 90, 92a et b]. Son architecture cible est basée sur celle du GAPP [CLOU 88]. Ce système génère un réseau de processeurs élémentaires reliés en un réseau à 2 dimensions. Chaque processeur élémentaire a deux registres de communication (un pour la direction Nord/Sud, l'autre pour la direction Est/Ouest) et une seule UAL, déjà connectés. Lors de l'assignation des ressources du processeur élémentaire, ces deux registres de communication doivent être utilisés en priorité, pour éviter la création de registres et de connexions supplémentaires. Pour cela, la méthode d'assignation de ressources décrite dans cette thèse a été appliquée. Des règles spécifiques ont été définies, qui permettent de cibler exactement l'architecture d'une processeur

élémentaire.

Le logiciel d'assignation de ressources est implanté en langage C. Il part d'un GDD ordonnancé défini par pas de contrôle et construit progressivement une PO formée de blocs interconnectés. Le module d'application des règles est séparé. Il construit un graphe biparti défini par une matrice d'adjacence grâce aux informations du GDD et de la PO. Les informations du GDD sont les éléments de stockage et leur présence en entrée ou en sortie d'opérateurs au pas de contrôle considéré. Les informations de la PO sont les registres déjà connectés en entrée ou en sortie d'opérateurs.

Bien que la programmation de l'application des règles soit dans un module séparé, il paraît peu vraisemblable qu'un concepteur programme de nouvelles règles. Il est impossible de vérifier la cohérence du graphe biparti construit pour l'application de nouvelles règles.

Chapitre III

ALLOCATION DES CONNEXIONS

Nous avons vu, dans le chapitre précédent, une heuristique pour l'assignation d'instances d'opérateurs aux opérations élémentaires et de registres aux éléments de stockage virtuels d'un GDD ordonnancé représentant le comportement du circuit. Cette heuristique a permis de préparer efficacement l'allocation des connexions par rapport au choix d'une architecture cible.

L'allocation de registres affecte des registres en entrée d'opérateurs sans distinction entre les différentes entrées d'un opérateur commutatif. Or, l'exploitation de la commutativité des opérateurs réduit le nombre de connexions de la partie opérative. Une première phase de l'allocation des connexions consiste donc en l'alignement des opérandes et est décrite en première section de ce chapitre.

Les différentes classes d'architecture répertoriées font intervenir des bus ou des multiplexeurs pour l'implantation des transferts de données. Les architectures à bus et registres dédiés impliquent la duplication de certains registres. L'allocation des connexions va donc allouer des bus ou des multiplexeurs et définir les registres à dupliquer, ces dernières étapes étant décrites en deuxième section de ce chapitre.

III.1. Alignement des opérandes

Durant l'assignation de ressources, par la méthode proposée au chapitre précédent comme par une méthode de partitionnement en cliques, la commutativité des opérateurs n'a pas été prise en compte. Les registres sont assignés à l'entrée d'un opérateur commutatif sans distinction entre ses ports. Il est donc nécessaire de donner un ordre aux éléments des paires de registres en entrée d'opérateurs commutatifs. De plus, l'exploitation de la commutativité des opérateurs peut réduire le coût en interconnexions de la partie opérative. Par exemple, une implantation de la partie opérative à base de multiplexeurs, respectant l'ordre des registres des paires en entrée d'opérateurs de la figure III.1, résultante de l'assignation de registres et d'opérateurs de la figure II.11, peut être celle de la figure III.2(a). Le registre de la valeur 3 est présent aux deux entrées du multiplieur $Mult_2$. Inverser l'ordre du couple (3, R_1) devant ce multiplieur induit un gain d'un multiplexeur et la PO de la figure III.2(b). Inverser maintenant l'ordre du couple (dx, R_1) devant l'additionneur permet de partager un multiplexeur entre l'additionneur Add et le multiplieur $Mult_2$. Un gain d'un multiplexeur et la PO de la figure III.2(c) sont obtenus.

	Mult1	Mult2	Add	Sub
Entrées	G D	G D	G D	G D
pas de contrôle 1	R0 dx	3 R1	dx R1	
pas de contrôle 2	R3 R4	R2 3		
pas de contrôle 3	R3 dx	R0 dx		R0 R4
pas de contrôle 4			R2 R3	R0 R4

Figure III.1: Une solution au problème d'alignement des opérandes à partir du résultat de la figure II.11

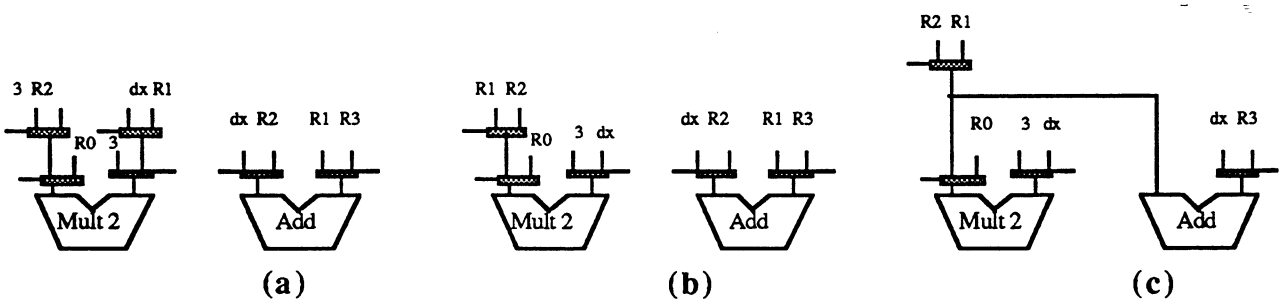


Figure III.2: Parties opératives à base de multiplexeurs

Définition III.1: L'assignation des registres contenant des opérandes à des entrées spécifiques d'opérateurs s'appelle l'*alignement des opérandes*. Par abus de langage, *opérateur commutatif* représentera un opérateur qui, à l'instant considéré, réalise une opération commutative. Un opérateur universel, réalisant à la fois des opérations commutatives et non commutatives, sera donc représenté à la fois par un opérateur commutatif et par un opérateur non commutatif, selon l'opération qu'il exécute à l'instant considéré. Une *opérande* est une valeur lue dans un registre à un pas de contrôle donné par un opérateur donné. Une *paire d'opérandes* est formée des deux opérandes apparaissant à l'entrée d'un opérateur à un instant donné.

Définition III.2: Une *opérande* notée $R_i^{p,op}$ représente la lecture du registre R_i, à l'entrée de l'opérateur op, à l'instant p.

Par exemple, $R_0^{1,Mult1}$ représente le registre R_0, de la figure III.1, lu au pas de contrôle 1 par l'opérateur $Mult_1$.

III.1.1. Description de la méthode

Dans cette section, une méthode originale d'alignement des opérandes (définition III.1), exploitant la commutativité des opérateurs pour diminuer le coût en connexion, est proposée. Cette méthode est fondée sur un problème de bipartitionnement de coupe de poids maximal de l'ensemble des opérandes. La pondération est définie par l'application de règles. La résolution du bipartitionnement se fait par bipartitionnements successifs par la méthode dite du *Min-cut*, définie par Kerninghan et Lin [KERN 70]. La méthode de base est adaptée à l'alignement des opérandes en procèdant par échanges successifs de paires d'opérandes qui augmentent le poids de la coupe. L'alignement des opérandes est fondé sur un modèle commun à tous les styles d'architecture définis.

Pour simplifier la description de la méthode, les opérateurs seront considérés, dans un premier temps, comme ayant tous deux entrées. Une extension aux opérateurs à plus de deux entrées sera décrite ultérieurement. La définition de ce modèle est fondée sur la définition de ressources d'interconnexion virtuelles et de Boite Noire d'Interconnexion (BNI).

Définition III.3: Les *ressources d'interconnexion virtuelles* sont des multiplexeurs et des bus virtuels.

En considérant une solution donnée au problème d'alignement des opérandes, il est possible d'identifier l'existence d'une ressource d'interconnexion virtuelle, dès l'instant où plusieurs ressources sont assignées à la même entrée d'une ressource donnée. Cependant, une ressource d'interconnexion virtuelle ne sera pas toujours affectée à une ressource d'interconnexion physique. Par exemple, figure III.1, $R_0^{3,Mult2}$ et $R_1^{2,Mult2}$ sont utilisés à des instants différents à la même entrée de l'opérateur $Mult_2$, mais un multiplexeur n'existe pas forcément entre les deux registres R_0 et R_1, tel que le montre la figure III.4(a).

Définition III.4: La *Boite Noire d'Interconnexion (BNI)* gauche (respectivement droite) représente l'ensemble des ressources d'interconnexion virtuelles connectées à toutes les entrées gauches (respectivement droites) des opérateurs.

L'ensemble des entrées des opérateurs est donc divisé en deux sous-ensembles: le sous-ensemble des entrées droites et le sous-ensemble des entrées gauches. Pour une solution donnée au problème d'alignement des opérandes, il existe donc une BNI droite et une BNI gauche. Les BNI de la figure III.3 sont obtenues à partir de la solution initiale de la figure III.1, en considérant que les paires d'opérandes de cette figure sont des couples.

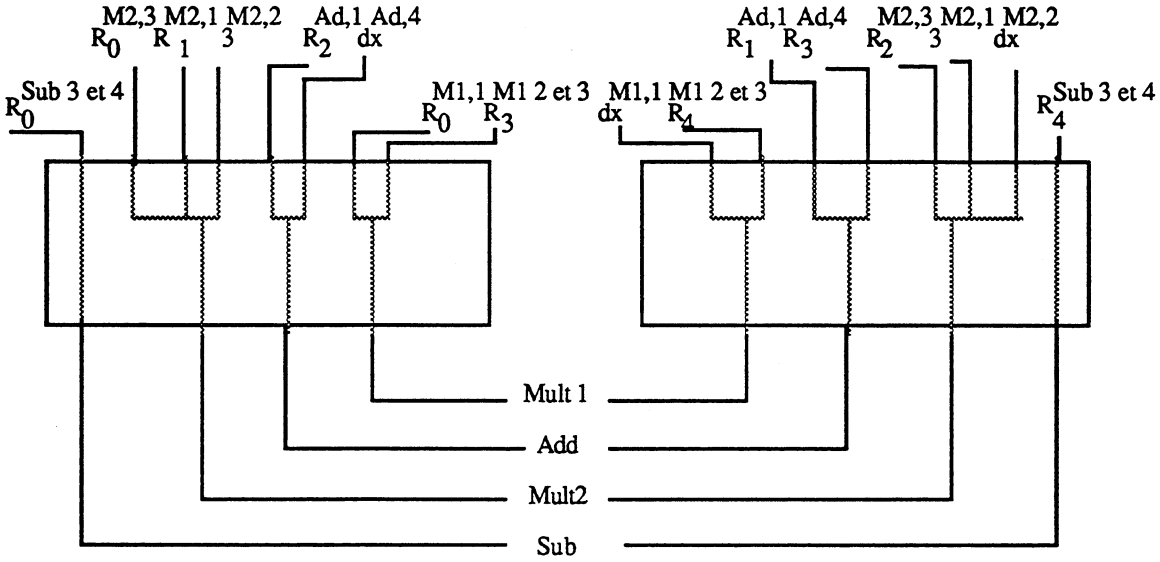
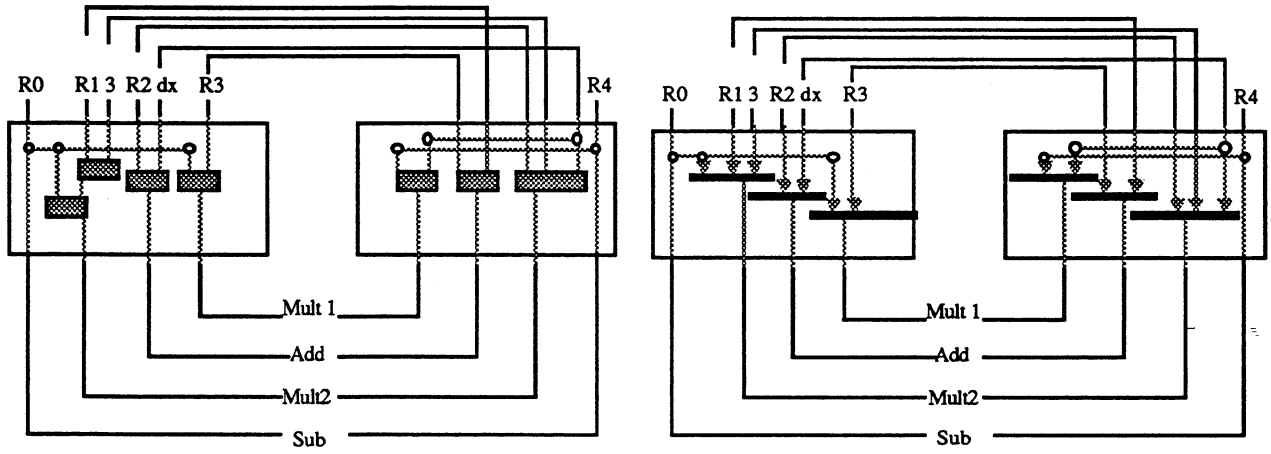


Figure III.3: Représentation des BNI pour l'exemple de la figure III.1.

Les BNI représentent des ressources virtuelles; elles peuvent être représentées selon différentes implantations: celle de la figure III.4(a) pour un style d'architecture à base de multiplexeurs ou celle de la figure III.4(b) pour une architecture à base de bus.



(a) à base de multiplexeurs **(b) à base de bus**

Figure III.4: Deux implantations des BNI

L'alignement des opérandes consiste alors à inverser l'ordre des couples d'opérandes à l'entrée d'un opérateur commutatif, de telle sorte que la surface de chaque BNI et le nombre d'interconnexions entre les BNI soient minimisés. Dans ce but, il faut, d'une part, minimiser le nombre de ressources d'interconnexion virtuelles à l'entrée de chaque opérateur et, d'autre part, optimiser le partage de ressources d'interconnexion virtuelles entre différents opérateurs.

Il faut remarquer que la minimisation des ressources d'interconnexion virtuelles réduit aussi le nombre d'interconnexions. Le partage d'une même ressource physique par deux ressources virtuelles induit, d'une part, deux fois moins d'interconnexions entre les registres et les ressources d'interconnexion virtuelles, et, d'autre part, une interconnexion à trois extrémités à la place de deux interconnexions bipoints entre les ressources d'interconnexion et l'entrée des opérateurs.

Pour une architecture à base de multiplexeurs, la surface d'une BNI peut être estimée par le nombre de multiplexeurs 2-1 nécessaires à son implantation. Ce nombre est $\sum_{o_i \in I} \lceil \log_2(no_i) \rceil$, où no_i représente le nombre de registres différents apparaissant à l'entrée de l'opérateur o_i. De ce nombre, il faut retrancher le nombre de multiplexeurs communs à plusieurs entrées. Les multiplexeurs communs ne seront identifiés qu'en fin d'allocation des connexions.

Par conséquent, plutôt que d'estimer la surface de ces BNI, nous définissons des règles d'attraction ou de répulsion entre les opérandes.

Une règle d'attraction entre deux opérandes incitera ces opérandes à être affectées à la même boîte noire d'interconnexion (BNI définition III.4) et donc à partager des ressources d'interconnexion virtuelles (définition III.3). En revanche, une règle de répulsion poussera deux opérandes à être affectées à des BNI différentes.

Deux règles d'attraction, la *similarité* et la *compatibilité*, et une règle de répulsion, l'*incompatibilité*, sont définies. La similarité exprime le fait que plusieurs opérandes d'un même registre doivent se trouver dans la même BNI, afin de minimiser les ressources d'interconnexion virtuelles à l'entrée de chaque opérateur, ainsi que les interconnexions entre les BNI. La compatibilité exprime la faculté de deux opérandes, à l'entrée d'opérateurs différents, de partager la même ressource d'interconnexion. Cette règle minimise le nombre de ressources d'interconnexion virtuelles à l'entrée de l'ensemble des opérateurs. L'incompatibilité est la règle complémentaire de la compatibilité.

La règle de similarité permet un gain de multiplexeurs et de bus, tel que l'illustre le passage de la figure III.2(a) à la figure III.2(b). La règle de compatibilité permet un gain tel que l'illustre le passage de la figure III.2(b) à la figure III.2(c).

Les définitions de ces règles sont les suivantes:

Soient deux opérandes $R_{i1}^{p1,o1}$ et $R_{i2}^{p2,o2}$.

- La similarité fait intervenir des opérandes à l'entrée d'un même opérateur.

Un *lien de similarité* est créé entre deux opérandes si elles apparaissent à l'entrée du même opérateur et sont extraites du même registre, donc si et seulement si $i_1=i_2$ et $o_1=o_2$. Dans l'exemple de la figure figure III.1, $R_3^{2,Mult1}$ et $R_3^{3,Mult1}$ sont liées par un lien de similarité.

- L'incompatibilité et la compatibilité font intervenir des opérandes apparaissant à l'entrée d'opérateurs différents. Il s'agit alors de définir dans quelles conditions ces opérandes pourront partager la même ressource d'interconnexion physique. Il est donc nécessaire de connaître le type de ressource d'interconnexion de l'architecture. Supposons, dans un premier temps, que ce sont des multiplexeurs; nous verrons par la suite l'utilisation de ces règles pour une architecture à base de bus.

Des opérandes d'un sous-ensemble SSO, de l'ensemble des opérandes SO, peuvent partager le même multiplexeur si elles sont affectées à la même BNI et sont compatibles. Autrement dit, si, à chaque

pas de contrôle, toutes les opérandes de SSO, lues par les opérateurs, représentent le même registre (condition I) et si toutes les apparitions de ce registre, devant les opérateurs considérés, sont dans SSO (condition II); c'est-à-dire si SSO vérifie les conditions (I) et (II) suivantes:

Soit BNI_1 l'ensemble des opérandes affectées à l'une des deux BNI, celle de droite par exemple, et soit SSO un sous-ensemble de BNI_1:

(I) $\forall$ p, si $\exists$ ($R_{i1}^{p,o1} \in$ SSO), alors $\nexists$ ($R_{i2}^{p,o2} \in$ SSO) / $i_1 \neq i_2$

(II) si $R_i^{p1,o1} \in$ SSO, alors $\nexists$ ($R_i^{p2,o2} \in BNI_1$-SSO).

• La compatibilité relie donc deux opérandes vérifiant la condition (I).

• L'incompatibilité est définie par la condition duale (nonI) de la condition de compatibilité (I) de la manière suivante:

(nonI) $\exists$ p / ($R_{i1}^{p,o1}$ et $R_{i2}^{p,o2} \in$ SSO) et $i_1 \neq i_2$

Par exemple, $R_2^{4,Add}$, $R_2^{2,Mult2}$, $dx^{1,Add}$, $dx^{3,Mult2}$ sont compatibles. $R_1^{1,Mult2}$ et $R_1^{1,Add}$ sont compatibles. En revanche, $dx^{1,Add}$ est incompatible à $R_1^{1,Mult2}$.

Le problème d'alignement des opérandes est maintenant simple à représenter par un graphe biparti. Chaque partie représente les opérandes affectées à une BNI. Les arêtes de ce graphe représentent les relations qui viennent d'être définies. Le graphe biparti de la figure III.5 est obtenu à partir de la figure III.1. Ce graphe ne représente que les relations de compatibilité et d'incompatibilité sur les opérandes de Mult2, car le graphe biparti contient 17 arêtes de compatibilité et 41 arêtes d'incompatibilité.

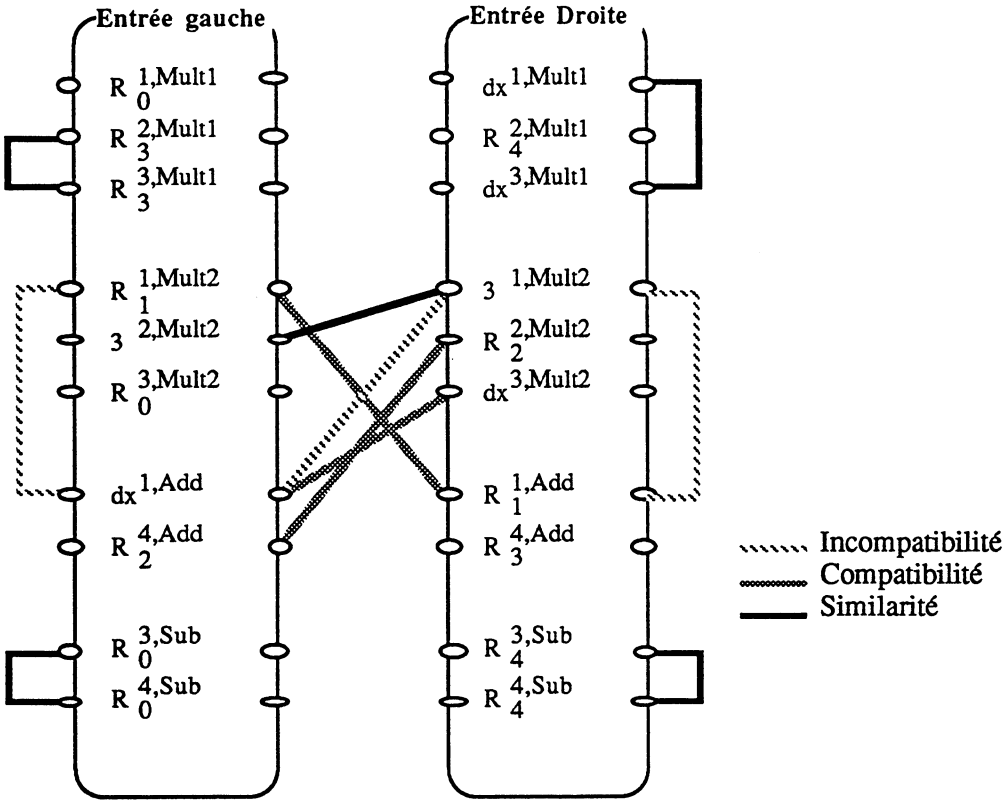


Figure III.5: Graphe biparti de la figure III.1

Afin d'optimiser l'alignement des opérandes, les arêtes de similarité et de compatibilité doivent se

trouver hors de la coupe du graphe biparti ainsi créé. En revanche, les arêtes d'incompatibilité doivent être dans la coupe. Le poids des arêtes est donc défini par la fonction p suivante:

$p(a_{Incomp}) = -p(a_{Comp})$ et $p(a_{Sim}) >> p(a_{Comp})$.

Un alignement des opérandes optimal sera obtenu pour une coupe de poids minimal. Les arêtes de similarité ont un poids infini par rapport aux arêtes de compatibilité car elles mènent systématiquement à un gain en ressources d'interconnexion. En effet, des opérandes affectées à la même BNI, formant un sous-ensemble, ne seront compatibles que si toutes les apparitions d'un registre à l'entrée des opérateurs considérés sont dans ce sous-ensemble (condition II). Cette condition n'est pas forcément respectée par un partitionnement de poids minimal.

Lors de la recherche de la coupe de poids minimal, une contrainte est induite du fait que deux opérandes d'une même paire ne peuvent se trouver dans la même BNI et donc dans la même partie de la bipartition. Cette contrainte peut se modéliser par l'introduction d'arêtes dites *d'impossibilité*, de poids moins l'infini. Nous verrons que l'algorithme défini pour la résolution du bipartitionnement respecte cette contrainte.

Le problème de bipartitionnement d'un graphe de coupe minimale est un problème NP-Complet. La résolution de ce problème peut être réalisée par une adaptation de l'heuristique du Min-cut [KERN 70]. A partir d'une solution initiale, des transformations du graphe biparti sont essayées en vue d'améliorer cette solution. Dans notre contexte, ces transformations consistent à échanger une paire d'opérandes devant un opérateur commutatif ou à échanger toutes les paires d'opérandes devant un opérateur non commutatif. Tous les échanges sont essayés, celui qui minimise le poids de la coupe est effectivement appliqué. Afin que l'algorithme s'arrête, les paires qui ont été échangées sont marquées et fixées. Afin de sortir d'un minimum local, l'algorithme du Min-cut accepte des augmentations du poids de la coupe à concurrence d'un seuil donné. Cet algorithme est résumé figure III.7.

La complexité de cet algorithme est majorée par $0(o^2)$, où o est le nombre de paires d'opérandes. Elle est donc majorée par $0(n^2)$, où n est le nombre d'opérations élémentaires.

L'application de cet algorithme sur l'exemple de l'équation différentielle de la figure II.11 est illustrée par la liste de couples de la figure III.6; ce résultat correspond à la PO de la figure III.2 (c).

	Mult1	Mult2	Add	Sub
Entrées	G D	G D	G D	G D
pas de contrôle 1	R0 dx	R1 3	R1 dx	
pas de contrôle 2	R3 R4	R2 3		
pas de contrôle 3	R3 dx	R0 dx		R0 R4
pas de contrôle 4			R2 R3	R0 R4

Figure III.6: Résultat de l'algorithme d'alignement des opérandes sur l'exemple de la figure II.11

Soit B une limite pour les gains négatifs et M le nombre maximal de minima locaux à observer.

```
Construire le graphe biparti initial
Lister tous les échanges possibles
Tant que (Il reste des échanges possibles) et ((le gain courant est supérieur à B)
ou (Le nombre de minima locaux n'est pas supérieur à M))
   Pour chaque échange possible faire
      Essayer l'échange courant
      Si le gain est positif alors
         Conserver le coût comme coût courant
      Sinon
         le gain est négatif
         Si ce gain est supérieur à B alors
         Conserver le coût comme borne locale
         FinSi
      FinSi
   FinPour
   Si On a trouvé un gain positif Alors
      Appliquer la permutation correspondant au coût courant
      Fixer la (ou les) paire(s) correspondante(s)
   Sinon
      Si On a trouvé un gain inférieur à la borne B Alors
      Appliquer la permutation correspondant à la borne locale
      Fixer la (ou les) paire(s) correspondante(s)
      Incrémenter le nombre de minima locaux
      FinSi
   FinSi
FinTantQue
```

Figure III.7: Algorithme d'alignement des opérandes

III.1.2. Extensions

Dans le cas d'architectures à base de bus, deux bus par opérateur sont à priori alloués. Deux bus pourront donc être regroupés s'ils sont compatibles. Deux bus sont compatibles si et seulement si à chaque instant une seule valeur transite par ces deux bus. C'est-à-dire que, pour une BNI donnée, quelles que soient les opérandes devant les deux opérateurs considérés, la condition (I) est vérifiée. Les relations d'incompatibilité et de compatibilité peuvent donc être appliquées telles qu'elles ont été définies. Le regroupement d'opérandes pour partager un même bus ne sera possible que si toutes les opérandes (affectées à une BNI donnée) à l'entrée des opérateurs sont compatibles. La relation de similarité entraînera un gain en portes trois états et en interconnexions. Par conséquent, l'algorithme d'alignement des opérandes peut être appliqué à une architecture à base de bus sans aucune modification.

Concernant les opérateurs à plus de deux entrées, le même modèle peut être conservé en affectant les opérandes à plus de deux BNI. L'heuristique du Min-cut devra donc être étendue au cas de multipartitionnements.

III.1.3. Autres méthodes d'alignement des opérandes et étude comparative

Peu de systèmes gèrent la commutativité des opérateurs. Le plus souvent, lors de l'utilisation de méthodes globale ou globale par pas de contrôle pour l'allocation de registres et d'opérateurs, les

registres sont alloués en entrée d'opérateurs sans distinction entre les deux entrées de l'opérateur.

La commutativité des opérateurs est prise en compte par les systèmes d'allocation de ressources utilisant une méthode locale. Les stratégies d'assignation essayées font intervenir l'entrée droite puis l'entrée gauche d'un opérateur, lors de l'assignation d'un transfert de données à un chemin dans la partie opérative. C'est le cas du système MABAL [KUCU 89] et du système d'allocation des interconnexions ELF [HITC 83]. Ce dernier réalise l'affectation de chaque connexion de manière séquentielle, en essayant pour chacune d'elles différents chemins dans la partie opérative. Ces chemins peuvent en particulier essayer d'utiliser soit un opérateur libre pour le transfert d'une donnée, soit un bus ou un multiplexeur existant dans la partie opérative, ou essayer d'allouer une nouvelle ressource. Le chemin minimisant une estimation de la surface sera choisi en définitive. Ce procédé d'assignation a pour cible des architectures irrégulières.

Une méthode locale et séquentielle d'alignement des opérandes est décrite dans [PAPA 90]. Cette méthode part d'une solution initiale du problème d'allocation des interconnexions et traite chaque registre un par un, en réalisant des transformations exploitant la commutativité des opérateurs et menant au regroupement de multiplexeurs en entrée d'opérateurs. L'inconvénient de cette méthode est qu'elle considère qu'un seul multiplexeur est alloué en entrée de chaque opérateur. Des regroupements partiels, tels que ceux présentés figure III.2, ne sont pas envisageables. Or, dans le cas d'exemples pratiques, le regroupement de tous les multiplexeurs est rare, du fait de leur incompatibilité.

Dans CALLAS [MARZ 89] et COMPASS [MAHM 92], les auteurs mentionnent un système à base de règles pour exploiter la commutativité des opérateurs, sans plus de détails.

Le système EASY [STOK 88] effectue en dernière étape une amélioration de la partie opérative générée. Pour cette amélioration, une méthode de recuit simulé a été choisie. Les transformations essayées par EASY sont l'échange des registres en entrée d'un opérateur commutatif et l'échange de l'assignation d'un registre ou d'un opérateur. La fonction coût est calculée par la projection d'un cube fait de plans de connexion. Un plan de connexion représente par un 1 l'existence d'une connexion entre un registre et un opérateur, à un pas de contrôle donné. Pour des exemples complexes, le recuit simulé est très coûteux en temps de calcul et en mémoire. Cette méthode est donc utilisable en phase d'amélioration, mais pas lors de la réalisation de la PO.

Une résolution exhaustive de ce problème a une complexité de $0(2^n)$, où n est le nombre de paires à ordonner. Par exemple, pour 5 opérateurs utilisés sur 30 pas de contrôle, le coût est majoré par 2^{150} et dans l'exemple du filtre du $5^{ème}$ ordre, le coût est de 2^{32}. De plus, une fonction coût doit permettre de déterminer l'alignement d'opérandes optimal. Cette fonction doit prendre en compte la faculté des opérandes à partager un multiplexeur ou un bus; autrement dit, elle doit prendre en compte la compatibilité des opérandes. Cette recherche n'est pas aisée puisqu'elle est souvent modélisée par partitionnement en cliques.

Nous avons proposé une méthode d'alignement des opérandes commune à tous les styles d'architecture, fondée sur le bipartitionnement d'un graphe, en appliquant la méthode du Min-cut adaptée au problème considéré ici. Le Min-cut est une méthode itérative mais d'une complexité moindre par rapport au recuit simulé, puisqu'elle s'arrête après avoir essayé d'échanger chacune des

paires d'opérandes. Sa complexité est de $0(n^2)$.

III.2. Allocation de connexions pour chaque style

Telle qu'a été définie l'étape d'alignement des opérandes, les ressources pouvant partager la même connexion se trouvent à l'entrée de la même BNI. Si le problème d'allocation des connexions est trop complexe, il peut donc être divisé en deux sous problèmes.

La phase d'allocation des connexions dépend du style d'architecture. Pour l'ensemble des architectures, le problème est de définir les entrées de ressources pouvant utiliser des ressources d'interconnexion physiques communes.

Nous allons donc envisager les styles d'architecture successivement. Pour chacun d'eux, nous allons définir l'allocation de multiplexeurs ou de bus et, le cas échéant, les registres à dupliquer.

III.2.1. Allocation pour une architecture à base de multiplexeurs

L'allocation de multiplexeurs est résolue en deux étapes: une première phase recherche les registres communs à différentes entrées d'opérateurs et dont les opérandes sont compatibles; une deuxième phase construit des cascades de multiplexeurs.

Ces deux étapes sont réalisées de manière séquentielle en parcourant la liste des opérandes, en entrée d'une BNI, par nombre d'entrées d'opérateurs décroissant pour la recherche d'opérandes compatibles entre opérateurs, et par paires d'opérandes compatibles pour la génération de cascades de multiplexeurs à deux entrées et une sortie.

Le résultat de l'algorithme d'allocation des multiplexeurs, à partir du résultat de l'alignement des opérandes de la figure III.1, a donné une allocation optimale, illustrée figure III.8.

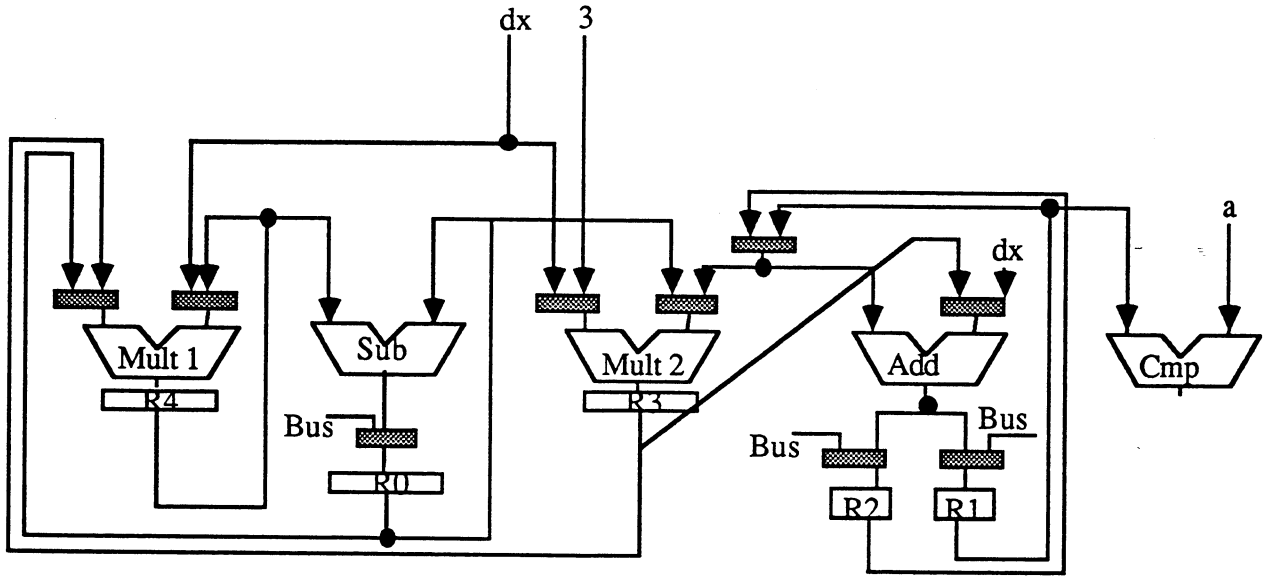


Figure III.8: Résultat de l'allocation de multiplexeurs pour l'exemple de l'équation différentielle

Pour l'allocation de multiplexeurs, comme pour l'allocation de ressources en général, le problème peut être formulé par un problème de partitionnement en cliques [PAUL 88]. L'inconvénient de cette

méthode est qu'elle ne peut regrouper que des multiplexeurs physiques et non des multiplexeurs virtuels; elle n'est applicable qu'après avoir généré une cascade de multiplexeurs à chaque entrée d'opérateur.

Wehn [WEHN 91] propose une méthode de minimisation de cascades de multiplexeurs, implantée dans CALLAS. Cette méthode élimine les multiplexeurs redondants dans des cascades de multiplexeurs. Elle modélise une cascade de multiplexeurs à deux entrées et une sortie par un arbre "if-then-else" et réalise des transformations sur chaque arbre séparément. Cette méthode a le défaut de ne pas partager de multiplexeurs entre différentes entrées de ressources.

III.2.2. Allocation pour les architectures à base de bus

Parmi les architectures à base de bus, trois architectures sont à nouveau distinguées: celle avec registres dédiés en entrée, celle avec registres dédiés en sortie et celle avec registres communs. Dans les trois cas, nous appliquerons la méthode classique d'allocation fondée sur le partitionnement en cliques pour la résolution du problème d'allocation. Pour construire le graphe de compatibilité lié à chacune des différentes architectures, des relations de compatibilité spécifiques à chaque style sont définies. Dans le cas d'architectures avec registres dédiés, une première phase consiste à dupliquer certains registres apparaissant à plusieurs entrées de ressources.

III.2.2.1. Duplication des registres

Lors de l'utilisation de registres dédiés en entrée (en sortie), il est nécessaire de dupliquer les registres apparaissant à différentes entrées (sorties) d'opérateurs, même si ces entrées concernent le même opérateur.

Pour des registres dédiés en entrée, les opérandes assignées à la même BNI sont parcourues et les registres apparaissant à plusieurs entrées d'opérateurs sont dupliqués. Pour des registres dédiés en sortie, l'ensemble des opérandes est parcouru et les registres apparaissant à plusieurs sorties d'opérateurs sont dupliqués.

Pour plus de facilité, les opérandes sont renommées en spécifiant leur provenance et leur destination.

Définition III.5: $R_D{}^{p,O_1}_{i,O_2}$ est l'opérande de SO (l'ensemble des opérandes) qui est émise par O_1 et stockée dans le registre R_i, au pas de contrôle p, et qui sera lue à l'entrée droite de l'opérateur O_2. De la même façon R_G est défini par $R_G{}^{p,O_1}_{i,O_2}$ pour l'entrée gauche.

Définition III.6: $DR_{entrée}$ le *nombre de registres à dupliquer en entrée* d'opérateurs, pour une architecture à base de registres dédiés en entrée, est défini par:

$$DR_{entrée} = \sum_{R_i \in R} sr_i - 1,$$

où sr_i, le nombre d'entrées d'opérateurs différentes en sortie du registre R_i, est défini par:

$sr_i = \| \{ R_C{}^{p_1,O_1}_{i,O_3},\ R_{C'}{}^{p_2,O_2}_{i,O_4} \in SO \,/\, O_3 \neq O_4 \text{ ou } (O_3 = O_4 \text{ et } C \neq C') \} \|$

Définition III.7: DR_{sortie} le *nombre de registres à dupliquer en sortie* d'opérateurs, pour une architecture à base de registres dédiés en sortie, est défini par:

$DR_{sortie} = \sum_{o_j \in I} so_j - 1$, où I est l'ensemble des instance d'opérateurs et

où est so_j, le nombre de registres différents en sortie de l'opérateur o_j, est défini par:

$so_j = \| \{ R_C{}^{p_1,O_j}_{i_1,O_3}, R_C{}^{p_2,O_j}_{i_2,O_4} \} \in SO / i_1 \neq i_2 \|$

Dans le cas de duplication de registres, le registre est dupliqué dès l'émission de la valeur par l'opérateur concerné. Le graphe de dépendance de données doit être modifié en dupliquant non seulement l'élément de stockage concerné à tous les pas de contrôle, mais aussi les arcs de dépendance de données.

Le résultat de la phase de duplication de registres en entrée d'opérateurs, pour l'exemple de l'équation différentielle, à partir de l'allocation de registres et d'opérateurs de la figure II.11 et de l'alignement des opérandes de la figure III.6, est donné figure III.9. Cette phase implique la duplication de 4 registres, mais ne nécessite pas de duplication pour une architecture à registres dédiés en sortie.

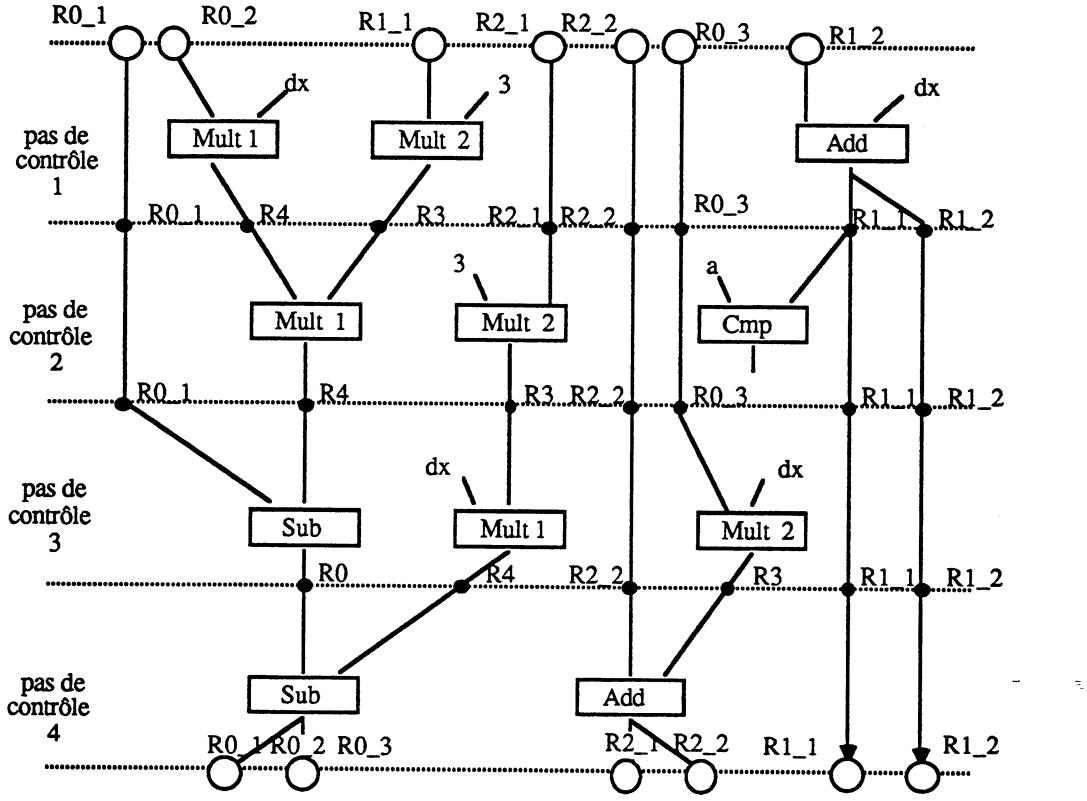


Figure III.9: Résultat de la duplication de registres pour une architecture à registres dédiés en entrée

III.2.2.2. Allocation de bus pour des architectures à registres dédiés

Une paire de bus virtuels a été allouée à chaque opérateur, pour les architectures à base de bus et de registres dédiés en entrée ou en sortie d'opérateurs. Deux bus virtuels sont compatibles si, à chaque instant, soit l'un des deux bus virtuels n'est pas utilisé pour un transfert, soit les deux bus virtuels transfèrent la même valeur vers deux destinations différentes. Dans le cas de registres dédiés en entrée

d'opérateurs, la même valeur est transférée si ses deux opérandes proviennent de la même sortie d'opérateur. Dans le cas de registres dédiés en sortie, la même valeur est transférée si les deux opérandes proviennent du même registre à la sortie du même opérateur. La définition de la compatibilité (condition (I)) donnée en section III.1.1 est reprise et adaptée à chaque style.

Définition III.8: Soient BV_1 et BV_2 deux bus virtuels dédiés respectivement à l'entrée des opérateurs O_1 et O_2,

BV_1 et BV_2 sont *compatibles* pour une architecture à base de registres dédiés en entrée si et seulement si:

(I_entrée) $\forall$ p, si $\exists$ ($R^{p,O_1}_{i_1,O_3} \in S0$) et ($R^{p,O_2}_{i_2,O_4} \in S0$) alors $O_3=O_4$

et BV_1 et BV_2 sont *compatibles* pour une architecture à base de registres dédiés en sortie si et seulement si:

(I_sortie) $\forall$ p, si $\exists$ ($R^{p,O_1}_{i_1,O_3} \in S0$) et ($R^{p,O_2}_{i_2,O_4} \in S0$) alors $i_1=i_2$, et $O_3=O_4$

En supposant que le résultat de l'allocation de registres et d'opérateurs soit celui de la figure II.11, pour une architecture à registres dédiés en sortie , le graphe de compatibilité pour cette architecture est celui de la figure III.10. Sur cette figure, les cliques sont indiquées en traits gras, le nœud Mult1_D représente le bus en entrée droite du multiplieur, Mult1_G celui en entrée gauche. Elles génèrent la partie opérative de la figure III.11.

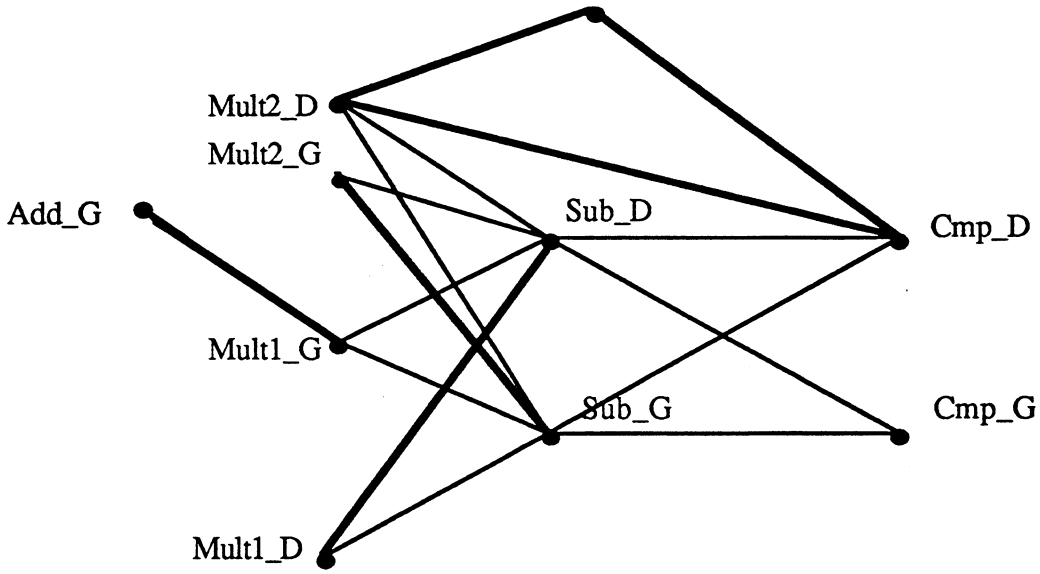


Figure III.10: Graphe de compatibilité

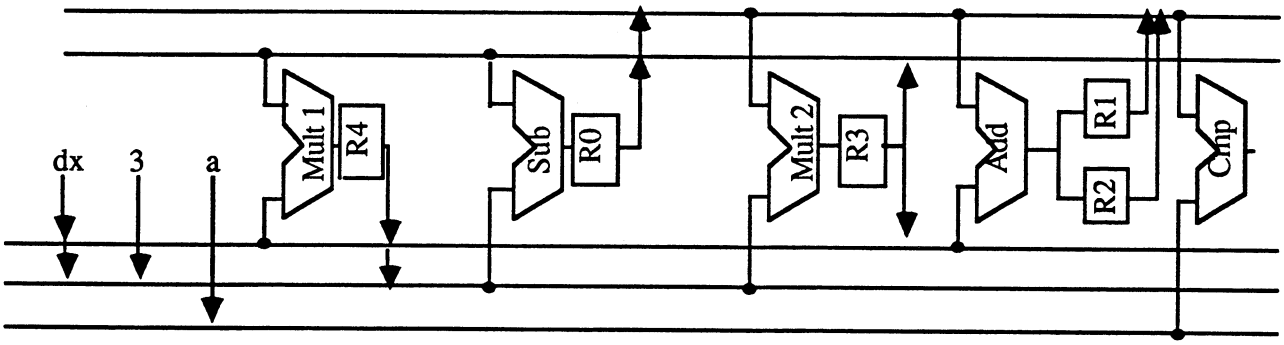


Figure III.11: Résultat de l'allocation de connexions pour une architecture avec registres dédiés en sortie

III.2.2.3. Allocation de bus pour des architectures à registres communs

Pour les architectures à registres communs, une paire de bus virtuels a été allouée par entrée d'opérateur, ainsi qu'un bus virtuel par entrée de registre. La définition de la compatibilité entre bus virtuels dépend, à présent, du fait qu'ils soient des bus virtuels en entrée ou en sortie d'opérateurs. Dans le cas de bus en entrée, la même valeur est transférée si les deux opérandes proviennent du même registre. Dans le cas de bus en sortie, la même valeur est transférée si les deux opérandes proviennent du même opérateur.

Définition III.9: Soient BV_1 et BV_2 deux bus virtuels d'entrée respectivement à l'entrée des opérateurs O_1 et O_2,

BV_1 et BV_2 sont *compatibles* pour une architecture à base de registres communs si et seulement si:

(I_commun_entrée) $\forall$ p, si $\exists$ ($R_{i_1,O_3}^{p,O_1} \in S0$) et ($R_{i_2,O_4}^{p,O_2} \in S0$) alors $i_1=i_2$

Définition III.10: Soient BV_1 et BV_2 deux bus virtuels de sortie respectivement à l'entrée des registres R_{i1} et R_{i2},

BV_1 et BV_2 sont *compatibles* pour une architecture à base de registres communs si et seulement si:

(I_commun_sortie) $\forall$ p, si $\exists$ ($R_{i_1,O_3}^{p,O_1} \in S0$) et ($R_{i_2,O_4}^{p,O_2} \in S0$) alors $O_1=O_2$

Dans le cas d'architectures avec insertion de registres tampons, les transferts vers l'entrée des opérateurs et les transferts vers l'entrée des registres ne sont pas réalisés au même cycle d'horloge. Un bus d'entrée est toujours compatible avec un bus de sortie.

Dans le cas d'architectures sans insertion de registres tampons, un bus en entrée et un bus en sortie ne peuvent jamais transférer la même valeur. Ces bus ne sont compatibles que si, à chaque instant, l'un d'eux ne transfère pas de valeur.

Définition III.11: Soient BV_1 et BV_2 deux bus virtuels respectivement en sortie de O_1 et en entrée de R_{i2},

BV_i et BV_j sont *compatibles* pour une architecture à base de registres communs sans registre tampon si et seulement si: $(R^{p,O_1}_{i_1,O_3} \in S0)$ et $(R^{p,O_2}_{i_2,O_4} \in S0)$

(I_commun) $\forall$ p, si $\exists (R^{p,O_1}_{i_1,O_3} \in S0)$ alors $\nexists (R^{p,O_2}_{i_2,O_4} \in S0) / i_1 \neq i_2$ et $O_1 \neq O_2$

et si $\exists (R^{p,O_2}_{i_2,O_4} \in S0)$ alors $\nexists (R^{p,O_1}_{i_1,O_3} \in S0) / i_1 \neq i_2$ et $O_1 \neq O_2$

En supposant que le résultat de l'allocation de registres et d'opérateurs soit celui de la figure II.11, pour une architecture à registres communs, le graphe de compatibilité pour une architecture à base de registres communs sans tampon est celui de la figure III.12. La PO correspondante est celle de la figure III.13.

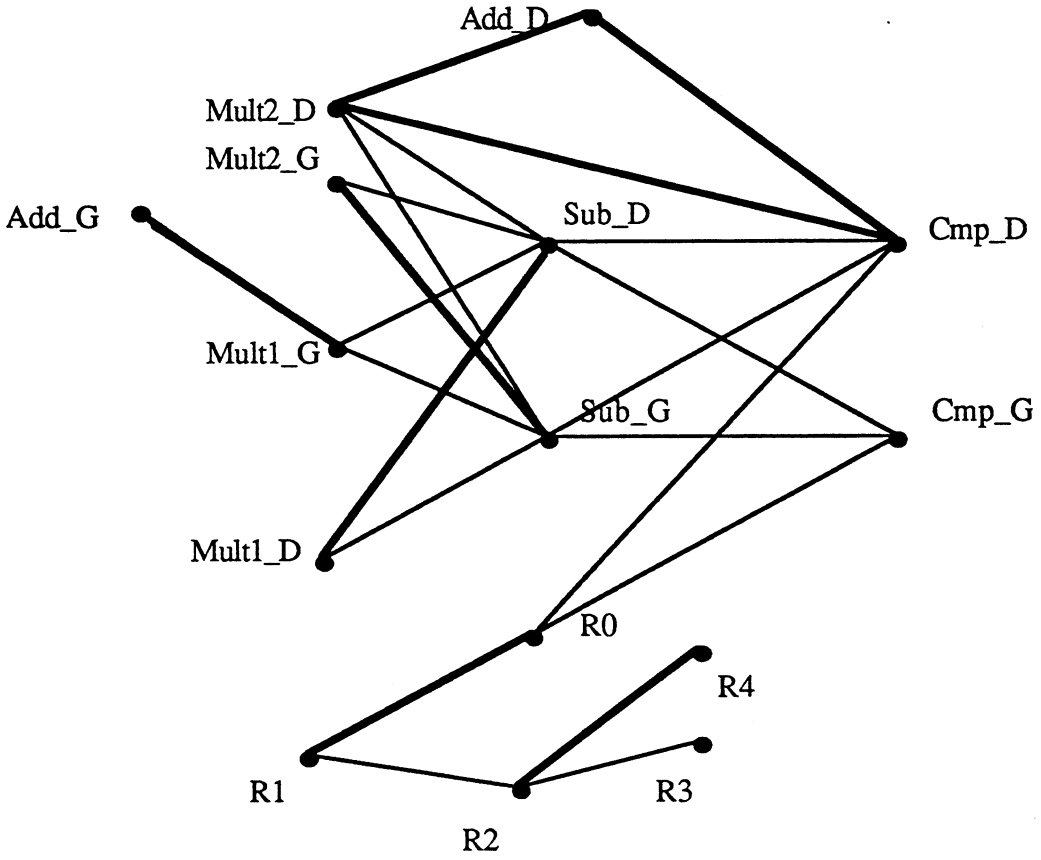


Figure III.12: Graphe de compatibilité pour une architecture à registres communs et sans tampon

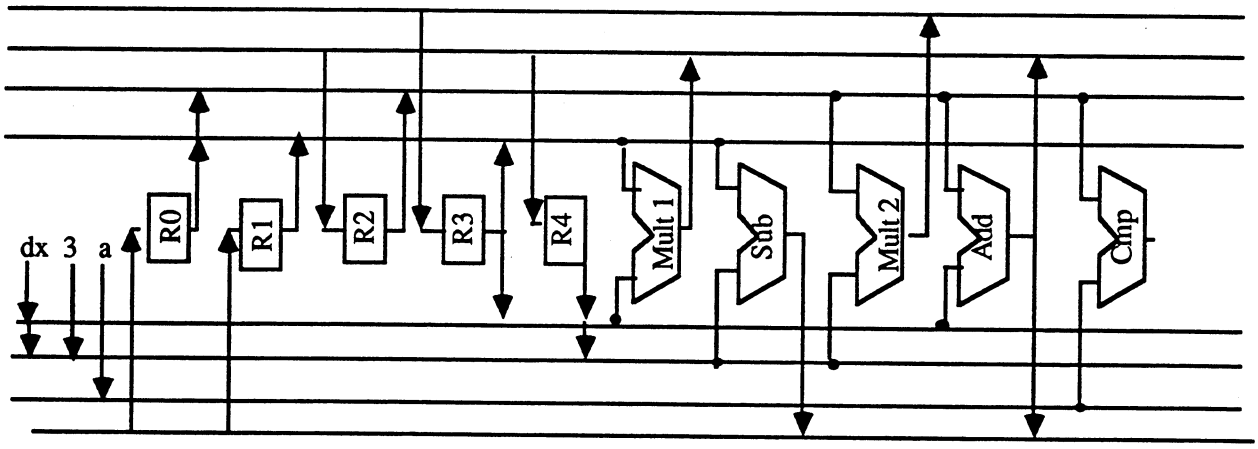


Figure III.13: PO correspondant à la figure III.12

III.3. Résultats

Ces algorithmes ont aussi été implantés en langage C sur SUN.

Les résultats concernant spécifiquement l'alignement des opérandes sont pauvres dans la littérature. La méthode proposée ici part d'une assignation de registres et d'opérateurs donnée. Il est clair que le résultat de l'alignement des opérandes dépend de cette assignation. Or, ce résultat intermédiaire n'est pour ainsi dire jamais donné dans la littérature.

Nos résultats sont donc comparés à des résultats optimaux (Gain/Min) et à des résultats obtenus de manière aléatoire (Gain/Moyen). Ceux-ci ont été calculés manuellement, ils ne sont donc pas donnés pour des exemples complexes.

De plus, chaque système utilise son propre style d'architecture; nos résultats sont comparés aux résultats des autres systèmes sur l'architecture la plus utilisée, c'est-à-dire celle à base de multiplexeurs.

Les critères d'évaluation sont le nombre de multiplexeurs générés et le nombre d'entrées de chaque multiplexeur pour estimer le nombre de connexions.

Les exemples sont les mêmes que ceux traités pour l'assignation de ressources (filtre du 5ème ordre, équation différentielle, FACET), plus deux exemples simples, définis figures III.15 et III.16, et un exemple défini dans [PAPA 90], illustré figure III.14.

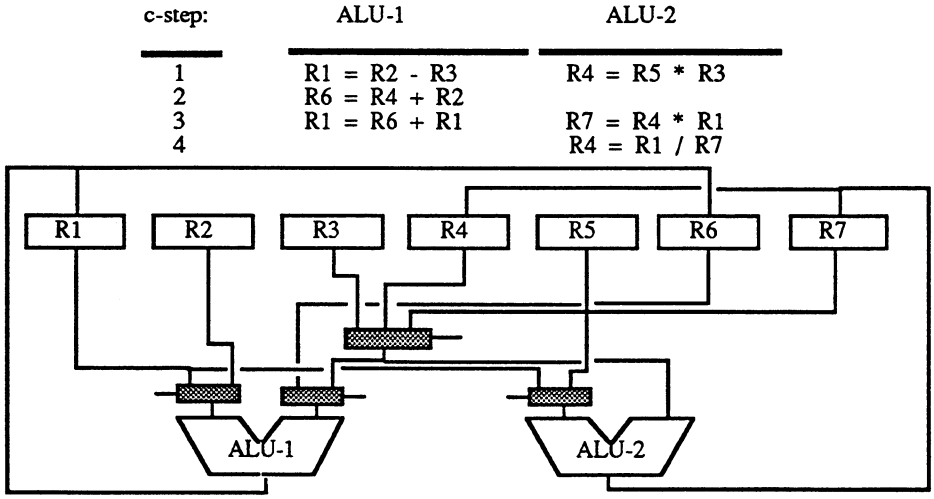


Figure III.14: Spécification et résultat pour l'exemple de [Papa90]

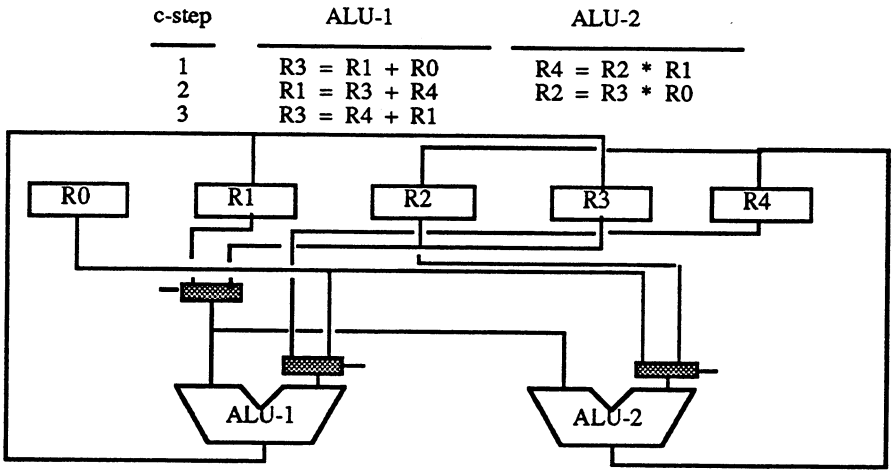


Figure III.15: Spécification et résultat pour l'exemple 1

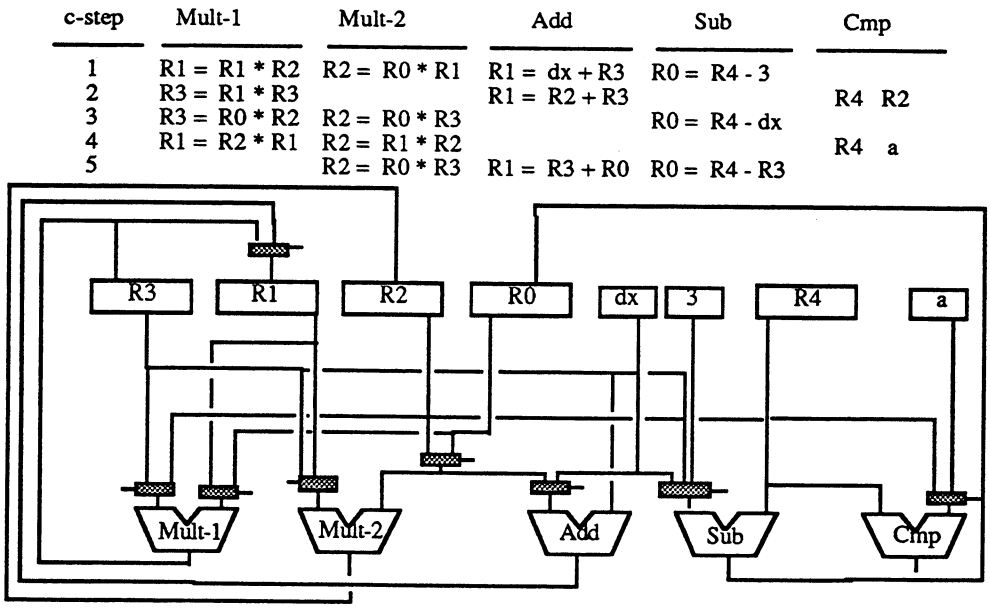


Figure III.16: Spécification et résultat pour l'exemple 2

Les caractéristiques de ces exemples sont données tableau III.1. Le tableau III.2 compare les résultats de différents systèmes à un alignement des opérandes aléatoire et à un alignement optimal. Le tableau III.3 donne les résultats de différents systèmes pour l'exemple plus complexe du filtre du 5ème ordre. Dans ces tableaux, *Mux 2-1* représente le nombre de multiplexeurs pour une architecture n'utilisant que des multiplexeurs deux entrées et une sortie et *Mux* le nombre de multiplexeurs pour une architecture utilisant des multiplexeurs avec autant d'entrées que nécessaire.

Tableau III.1: Caractéristiques des exemples

Min, *Max* et *Moyen* définissent le nombre d'entrées minimal, maximal et moyen de chacun de ces exemples

Exemple	Min	Max	Moyen
[Papa90]	8	12	10
exemple 1	6	10	7
exemple 2	14	21	20
FACET1	6	11	9
FACET2	5	10	8
Equa. Diff.	11	16	14

Tableau III.2: Résultats des systèmes ASYL, ADPS, FACET et HAL

sur les exemples 1 et 2, de [Papa 90], de FACET avec différentes allocations de registres (FACET1, FACET2, FACET3 et FACET4) et de l'équation différentielle (Eq. Diff)
en nombre de registres (Registre), de ressources différentes en entrée de ressource (Entrée), de multiplexeur (Mux) et de multiplexeur 2 entrées et 1 sortie (Mux 2-1) et en gain par rapport au nombre minimum de multiplexeurs 2-1 (Gain Min) et au nombre moyen de multiplexeurs (Gain Moy)

Système Exemple	Registre	Mux 2-1	Mux	Entrée	Temps cpu	Gain Min	Gain Moy
ASYL							
[Papa90]	7	5	4	8	0.02	0%	20%
exemple 1	4	3	3	6	0.01	0%	14%
exemple 2	8	9	8	16	0.08	- 12.5%	20%
FACET1	5	3	3	6	0.02	0%	34%
FACET2	5	3	2	5	0.02	0%	37,5%
Equa. Diff.	5	6	6	11	0.04	0%	21,5%
ADPS							
FACET3	7			10			
[Papa90]	7	5	3	8		0%	20%
FACET							
FACET4	8	6	5	11			
HAL							
FACET2	5	2+1bus	2	6		- 16%	25 %

Tableau III.3: Résultats des systèmes ASYL, EASY, ELF et HAL sur l'exemple du filtre du 5ème ordre

avec différentes contraintes globales d'ordonnancement (pas de contrôle),
différents contraintes de ressources (Registre, additionneur +, multiplieur * et multiplieur pipeliné *p)
en nombre de ressources différentes en entrée de ressource (Entrée), de multiplexeur (Mux) et de multiplexeur 2 entrées et 1 sortie (Mux 2-1)

Système	Pas de contrôle	Registre	+	* ou *p	Mux 2-1	Mux	Entrée	Temps cpu
ASYL	17	11	3	3*	26	124	32	0.18
		11		2*p	27	15	33	0.21
		12		2*p	20	12	30	0.31
EASY	17	18	3	2*p			34	
		20					22	
HAL	17	12	3	2*p			31	
ELF	17	11	3	2*p	16		28	

La méthode d'ASYL donne des résultats optimaux pour l'équation différentielle, l'exemple de FACET, l'exemple 1 et l'exemple de [PAPA 90]. Cette méthode génère deux entrées de multiplexeurs supplémentaires par rapport à l'optimum pour l'exemple 2. Pour cet exemple, la solution optimale est obtenue lorsque le même registre R_2 apparaît aux deux entrées d'un même opérateur. Ces résultats sont difficiles à comparer à ceux des systèmes ADPS [PAPA 90] et FACET [TSEN 83] car ils utilisent plus de registres.

Concernant l'exemple du filtre du 5ème ordre, les résultats d'ASYL sont meilleurs que ceux de HAL [PAUL 88], et semblent aussi meilleurs que ceux de EASY [STOK 91] puisque EASY utilise plus de multiplexeurs et plus de registres. Les résultats de ELF [LY 90] sont toujours meilleurs que ceux d'ASYL. Cependant, même manuellement, les résultats de ELF ne peuvent pas être obtenus. Il semblerait qu'ils aient été obtenus en utilisant des opérateurs libres pour le transfert d'opérandes.

Le temps cpu est toujours inférieur à une seconde.

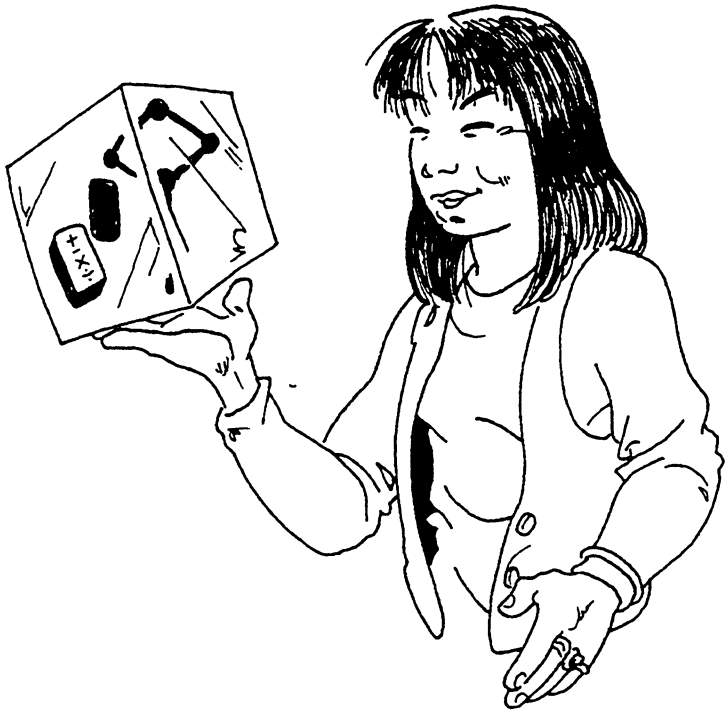
III.4. Conclusion sur l'allocation des connexions

L'alignement des opérandes, présenté dans ce chapitre, permet d'exploiter la commutativité des opérateurs pour minimiser le nombre de ressources d'interconnexions, quel que soit le style d'architecture choisi. Cet algorithme est fondé sur le bipartitionnement d'un graphe par la méthode de Min-Cut. Il est efficace, puisqu'il donne de meilleurs résultats que ceux de la plupart des autres systèmes, en moins d'une seconde.

Après cette phase, les opérandes stockées dans des registres sont assignées à une entrée déterminée d'opérateur. L'allocation des connexions affecte des ressources d'interconnexion de manière spécifique à chaque architecture. Elle alloue des multiplexeurs pour l'architecture à base de multiplexeurs et des bus, par une méthode fondée sur le partitionnement en cliques, pour les architectures à base de bus. Elle duplique les registres nécessaires pour des architectures à registres dédiés.

Chapitre IV

IMPLANTATION LOGICIELLE DANS ASYL



Les méthodes qui ont été présentées dans cette thèse sont implantées afin de s'inscrire dans le système de synthèse architecturale d'ASYL, qui est décrit dans ce dernier chapitre. Ce système est formé de deux parties: la synthèse de haut niveau et la synthèse RTL. Cette dernière peut être utilisier en phase finale de la synthèse de haut niveau pour réaliser l'allocation des connexions et la génération de la partie de contrôle ou comme un système de synthèse indépendant permettant une entrée plus pragmtique de la synthèse architecturale, laissant la maîtrise du séquencement au concepteur.

La synthèse de haut niveau part d'une description de type algorithmique, la traduit en un format interne représentant un graphe de dépendance de données et réalise l'ordonnancement et l'assignation de registres et d'opérateurs pour générer une description du circuit au niveau de transferts de registres.

La synthèse RTL implantée dans ASYL, part d'une description en termes de transferts de registres représentant un organigramme de contrôle et utilise une approche dirigée par le contrôle. Elle vise une architecture du circuit formée d'une partie opérative et d'une partie de contrôle. Ce système réalise l'allocation des connexions spécifique à chaque style d'architecture de la partie opérative et l'extraction puis la synthèse du contrôleur avec différentes cibles possibles.

IV.1. Le système de synthèse da haut niveau

Aujourd'hui, le point de départ de la synthèse de haut niveau, dans le système ASYL de synthèse, est un graphe de dépendance de données, intégrant les notions de contrôle des GFC, au format ASCIS [ASCI 91]. Dans l'avenir, une description VHDL permettra de spécifier le circuit de manière algorithmique ou de manière algorithmique sous contraintes de temps. Cette dernière pourra être similaire à celle de CALLAS pour la spécification de contraintes de temps. Dans ce cas, des macro-instructions VHDL seront définies afin de pouvoir décrire le comportement qui consiste à attendre un nombre donné de cycles d'horloge [GLUN 90].

Pour chaque opération du GDD, un type d'opérateur est déjà sélectionné. L'ordonnancement des boucles élémentaires est réalisé par l'ordonnancement orienté par les forces amélioré, présenté au premier chapitre. Les boucles sont ordonnancées les unes par rapport aux autres par un ordonnancement Au Plus Tot (APTO section I.4.1). Après l'ordonnancement, intervient la sélection du style d'architecture. Ensuite, l'assignation de registres est effectuée au regard de l'architecture choisie. L'assignation est réalisée par la méthode fondée sur la recherche d'un couplage de poids maximal dans un graphe biparti, présentée au deuxième chapitre. Cette méthode d'assignation permet de générer une partie opérative optimisée pour chaque style d'architecture défini par ses contraintes et ses critères.

Le graphe de flot de contrôle initial est alors associé à des blocs de base dont les graphes de dépendance de données sont ordonnancés et assignés. Le graphe de flot de contrôle ordonnancé et assigné est représenté par un organigramme de contrôle. Ce niveau de spécification correspond à l'entrée du système de synthèse RTL.

Le système de synthèse de haut niveau est illustré par la figure IV.1. Sur cette figure les parties tramées sont les parties non encore implantées.

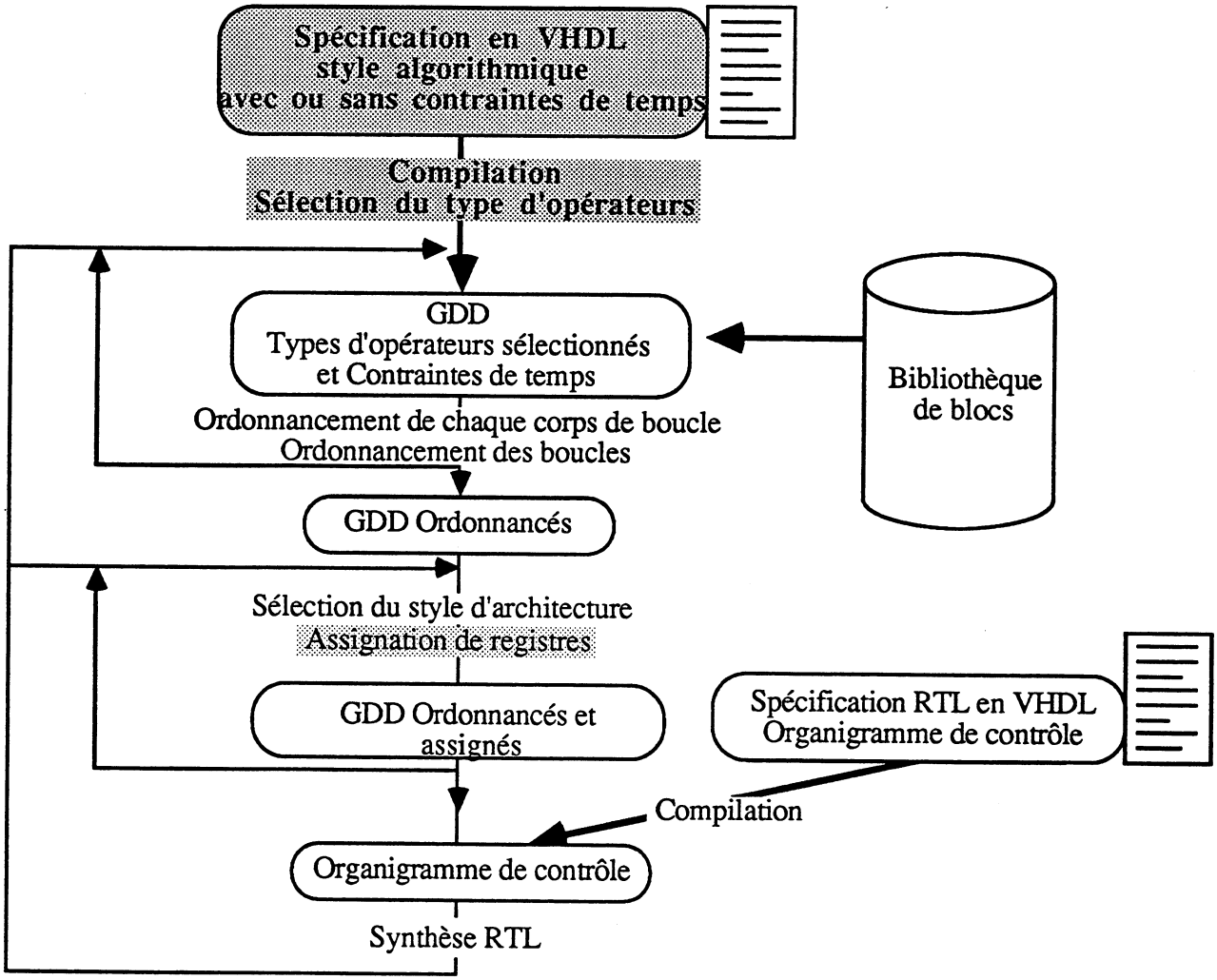


Figure IV.1: Système de synthèse ASYL

Comme l'illustre cette figure, le système de synthèse architecturale ASYL autorise une synthèse itérative afin d'essayer différentes implantations de la partie opérative pour l'assignation de registres et d'opérateurs, différentes options de synthèse RTL et différentes contraintes de temps pour l'ordonnancement.

Deux niveaux d'entrée de ce système sont possibles: le premier est classiquement utilisé en synthèse de haut niveau, il part d'un GDD; le second est un niveau en termes de transferts de registres, il part d'un organigramme de contrôle au niveau de la synthèse RTL.

IV.2. Le système de synthèse au niveau de transferts de registres (RTL)

Deux approches de synthèse RTL sont envisageables. L'approche dirigée par les données et l'approche dirigée par le contrôle.

L'approche dirigée par les données part d'une description RTL généralisée contenant des structures de branchements conditionnels et des opérations se déroulant pendant un cycle d'horloge. Cette approche synthétise le circuit en traduisant chaque constructeur de la spécification en équations booléennes au niveau du bit ou en les considérant comme des boîtes noires. Le circuit résultant est formé de blocs combinatoires et de registres banalisés.

L'approche orientée par le contrôle part d'une description RTL dans laquelle le contrôle est spécifiquement défini par un organigramme de contrôle et vise une architecture formée d'une partie opérative et d'une partie de contrôle.

Le choix d'une approche orientée par le contrôle, pour le système de synthèse RTL implanté dans ASYL, est motivé par la nécessité d'avoir une synthèse RTL et une synthèse de haut niveau cohérentes visant toutes deux une architecture formée d'une partie opérative et d'une partie de contrôle. De plus, cela a l'avantage de permettre des implantations plus complexes d'un circuit (contrôleur utilisant une ROM, contrôleur distribué ou hiérarchisé, partie opérative sur tranches de bit,...), de bénéficier des outils de synthèse performants, tels que les outils de synthèse de contrôleurs, des compilateurs de parties opératives, des générateurs de blocs, et de pouvoir avoir plusieurs cibles technologiques.

Dans la synthèse RTL d'ASYL, pour la génération de la partie opérative, un style d'architecture est sélectionné, puis l'alignement des opérandes et l'allocation des connexions, spécifiques à ce style, sont effectués. Pour la génération de la partie contrôle, un style de contrôleur est sélectionné, puis le graphe de contrôle est extrait de la représentation par organigramme de contrôle et est synthétisé. Le système de synthèse RTL est illustré figure IV.2. Sur cette figure les parties tramées ne sont pas encore implantées.

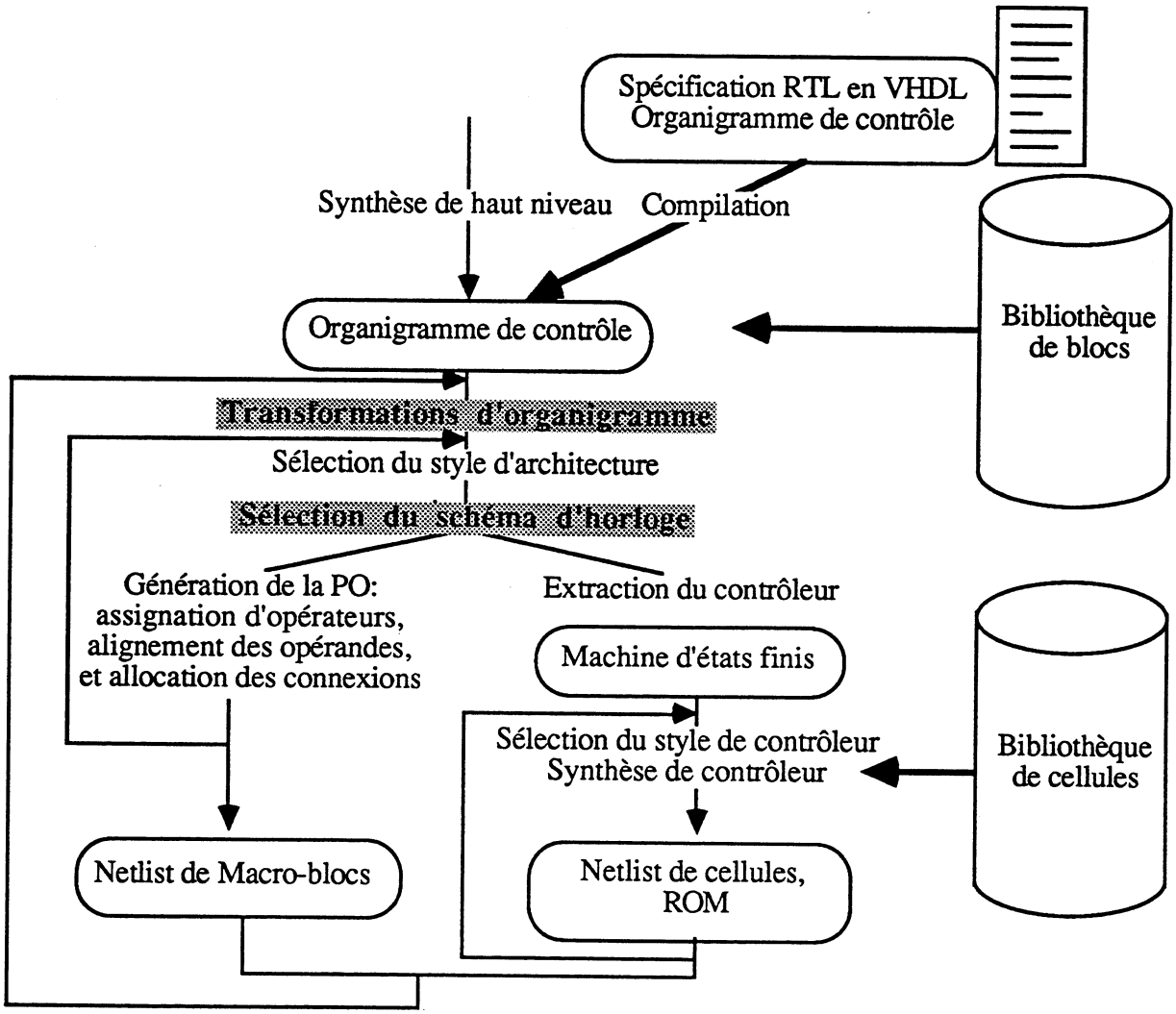


Figure IV.2: Système de synthèse RTL

Ce système est aussi hautement itératif, il permet de remettre en cause le choix du style d'architecture de la partie opérative, les options de la synthèse de contrôleur, le choix d'un schéma d'horloge. Dans l'avenir, une première étape réalisera des transformations sur un organigramme de contrôle initial, par regroupement ou découpage d'états [Mign 92c].

IV.2.1. Organigramme de contrôle

Un *organigramme de contrôle* est extrait du GFC de la spécification initiale, dont les GDD correspondants sont ordonnancés et les registres assignés.

Le GFC initial est transformé de telle sorte que chaque sommet représente l'ensemble des opérations élémentaires réalisées à un pas contrôle donné. Un organigramme de contrôle est un graphe orienté formé de sommets opératifs et de sommets de branchement conditionnel. Chaque boucle élémentaire est représentée par autant de sommets opérations que la date limite de fin de l'ordonnancement de son GP. Un *sommet opératif* représente l'ensemble des opérations s'exécutant durant un pas de contrôle p déterminé. Les opérations sont allouées à des opérateurs et les registres sont assignés. Par conséquent, les instructions sont décrites par des opérations sur des opérateurs existant en bibliothèque; elles prennent en entrée des registres et mettent à jour un registre. Un sommet de

branchement conditionnel de l'organigramme de contrôle correspond à un sommet de branchement conditionnel du GFC initial, ce sommet est indexé par le signal testé (un signal étant une donnée de 1 bit qui n'est pas forcément stockée dans une bascule ou un registre). Un arc a_{12} existe entre deux sommets opératifs Rtl_1 et Rtl_2, si les opérations qu'ils représentent ont été ordonnancées juste les unes (celles de Rtl_2) après les autres (celles de Rtl_1).

L'organigramme de contrôle de l'équation différentielle correspondant à la figure II.11, en supposant que "cmp" et "Sub" soient les opérations "<" et "-", réalisées sur un même opérateur UAL, est donné figure IV.3. Sur cet organigramme, le sommet en losange est le seul sommet de test de l'organigramme; il représente un test sur la variable X. Si cette variable vaut 1, le prochain sommet est le sommet 4, sinon le prochain sommet est le sommet 8. Une opération RTL: "R_i = Opérateur(Op, R_j, R_k)", signifie que l'opération "Op" est réalisée sur l'opérateur "Opérateur" existant en bibliothèque et que cette opération prend les registres R_j et R_k en entrée et met à jour le registre R_i.

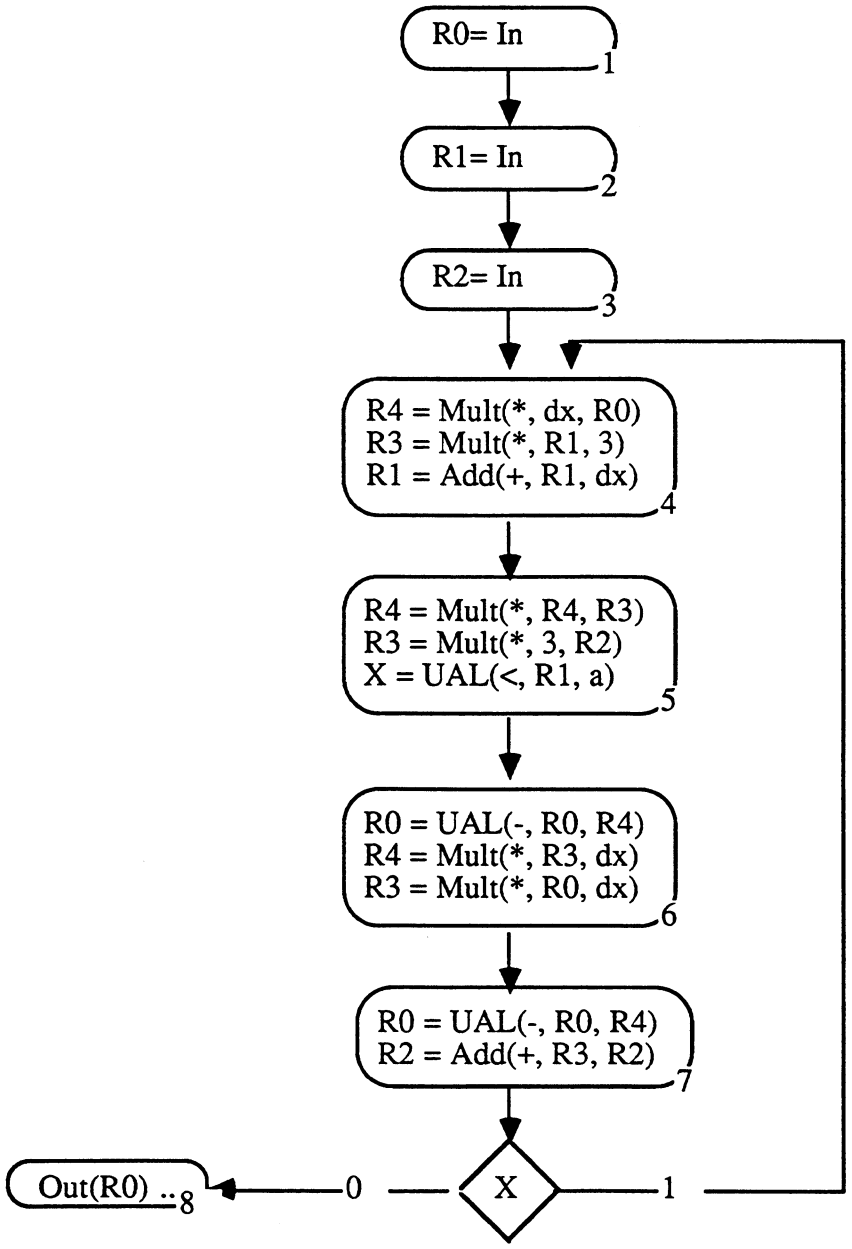


Figure IV.3: Organigramme de contrôle de la figure II.11

Une entrée textuelle en VHDL permet de définir le comportement d'un circuit directement au niveau RTL par un organigramme de contrôle.

L'avantage de VHDL est qu'il peut être utilisé pendant toute la durée de vie d'un circuit. Cependant, cette généralité impose des restrictions durant l'usage de VHDL pour un domaine particulier. Deux types de restrictions sont imposées: la première limite les types de constructions utilisées, la seconde limite la façon dont ces constructions sont utilisées. La première touche donc à la syntaxe et à la définition d'un sous-ensemble de VHDL alors que la deuxième touche à la sémantique et à la définition d'un modèle.

Nous allons donc présenter succinctement les définitions d'un sous-ensemble de VHDL et d'un modèle applicable à la synthèse RTL pour l'expression d'un organigramme de contrôle utilisant des macro-blocs existant en bibliothèque. Une définition plus complète du sous-ensemble VHDL et la comparaison de ce sous-ensemble avec les sous-ensembles choisis par les systèmes SYNOPSYS, CALLAS, COMPASS et SILCSYN peuvent être trouvées dans [BERT 92]. La définition syntaxique du sous-ensemble VHDL peut être trouvée dans [VHSU 92]. Ce sous-ensemble a été défini pour pouvoir à la fois simuler et synthétiser la même description: "ce qui est synthétisé est ce qui a été simulé à chaque instant". En plus des restrictions usuelles dues à l'utilisation de VHDL pour la synthèse, les options de synthèse pour notre système impliquent des restrictions spécifiques. Le choix de la représentation sous forme d'organigramme de contrôle implique que l'utilisateur a déjà défini une partition en états, un registre d'état est donc explicitement défini. Dans chaque état, des opérations s'exécutent en parallèle. Les transitions entre états sont étiquetées par des fonctions booléennes. Le modèle choisi a trois caractéristiques principales: la possibilité de définir un "reset" synchrone ou asynchrone, l'utilisation d'un processus monocycle basé sur la définition d'un choix sur la valeur du registre d'état et chaque opération est associée à une fonction correspondant à une opération d'un opérateur existant en bibliothèque.

IV.2.2. Génération de la partie opérative

A partir de cet organigramme, la génération de la partie opérative est réalisée par une assignation d'opérateurs, avec la méthode décrite au deuxième chapitre, par un alignement des opérandes et par une allocation des connexions, avec les méthodes décrites au troisième chapitre. Ces étapes sont réalisées de manière spécifique pour chaque style d'architecture, défini section II.1. Nous allons donc proposer des fonctions d'estimation de surface permettant de sélectionner un style d'architecture sans avoir besoin de réaliser toute la synthèse RTL et le placement routage définitif du circuit.

IV.2.2.1. Sélection du style d'architecture

Nous proposons d'aider un concepteur dans la sélection d'un style d'architecture, en estimant le gain en surface d'une architecture par rapport à une autre. Il ne s'agit pas ici de donner une estimation de la surface de chaque implantation possible de la partie opérative, mais bien d'être capable de les comparer. L'architecture de référence est celle à base de multiplexeurs. Des fonctions d'estimation de surface de chaque style d'architecture permettent de réaliser des comparaisons de surfaces.

L'assignation de ressources a été réalisée pour une architecture à base de multiplexeurs et un

multiplexeur a été généré par entrée ayant plusieurs origines, sans chercher à les partager.

- Soient R l'ensemble des r registres, I celui des o instances d'opérateurs, N celui des n multiplexeurs, divisé en deux sous-ensembles N_r et N_o de taille respective n_r et n_o, représentant respectivement les multiplexeurs en entrée de registres et en entrée d'opérateurs,
- soit la fonction in qui donne le nombre d'entrées d'un multiplexeur,
- soient so_i le nombre de registres différents en sortie de l'instance d'opérateur o_i, sr_i le nombre d'opérateurs différents en sortie du registre R_i et la fonction Surf donnant le coût en surface de chaque ressource et des connexions.

Si l'implantation topologique de la PO est réalisée par des blocs durs, il n'y a pas (ou peu) d'interconnexion à travers les blocs de la PO. Une estimation pour une implantation de la PO en tranches de bit sera donnée ultérieurement.

Le nombre d'interconnexions bipoints de la PO est estimé. Pour simplifier, une équipotentielle à plusieurs terminaisons est modélisée par plusieurs équipotentielles bipoints.

Un modèle des interconnexions pour une architecture à base de multiplexeurs est donné figure IV.4(a).

Cette architecture utilise $\sum_{n_i \in N} \lceil \log_2(in(n_i)) \rceil$ commandes pour les multiplexeurs. Une estimation de la surface de la PO, intégrant les connexions des commandes des multiplexeurs, est donc C_Mux, définie par:

$$C\_Mux = \sum_{X_i \in I,R} Surf(X_i) + \sum_{n_i \in N} Surf(n_i) + Surf(Conn) * \left(\sum_{o_i \in I} so_i + \sum_{R_i \in R} sr_i + n + \sum_{n_i \in N} \lceil \log_2(in(n_i)) \rceil \right)$$

Avec la même allocation de ressources, une implantation de la PO pour une architecture à base de bus avec registres dédiés en entrée est maintenant estimée. La surface dépend du nombre de registres qu'il a fallu dupliquer $DR_{entrée}$. Le cardinal de $DR_{entrée}$ est égal à $\sum_{n_i \in N_r} (in(n_i)-1)$.

Les n_o multiplexeurs en entrée d'opérateurs sont conservés. Une paire de bus par entrée d'opérateur est assignée. Il faut $\sum_{o_i \in I} so_i$ portes trois états.

Le modèle des interconnexions de la PO pour une architecture à base de bus avec registres dédiés en entrée est donné figure IV.4(b).

Le nombre de commandes pour les portes trois états est $\sum_{o_i \in I} so_i$. Le nombre d'interconnexions est donc I, défini comme suit:

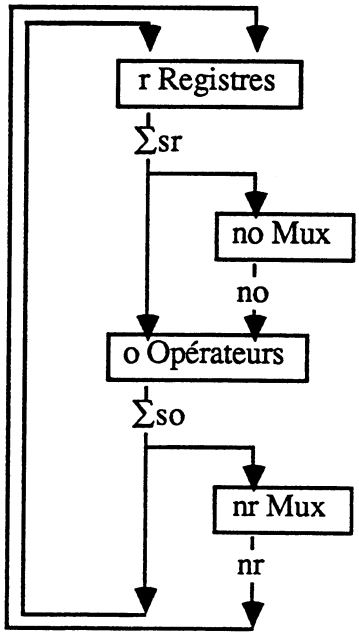
$$I = \sum_{R_i \in R} sr_i + 2o + n_o + 2 * \sum_{o_i \in I} so_i + \sum_{n_i \in N_o} \lceil \log_2(in(n_i)) \rceil$$

La surface de la PO peut être estimée par C_Entrée de la manière suivante:

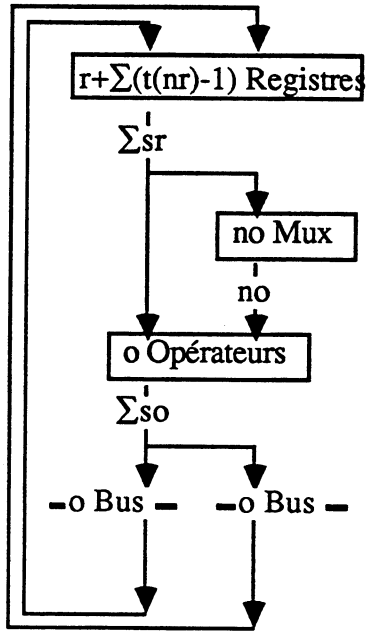
$$C\_Entrée = \sum_{X_i \in I,R,DR_{entrée}} Surf(X_i) + \sum_{n_i \in N_o} Surf(n_i) + Surf(Porte) * \sum_{o_i \in I} so_i + Surf(Conn) * I$$

Avec la même allocation de ressources, l'implantation de la PO pour une architecture à base de bus avec registres dédiés en sortie est estimée. La surface dépend du nombre de registres qu'il a fallu dupliquer DR_{sortie}. Le cardinal de DR_{sortie} est égal à $\sum_{n_i \in N_o} (in(n_i)-1)$.

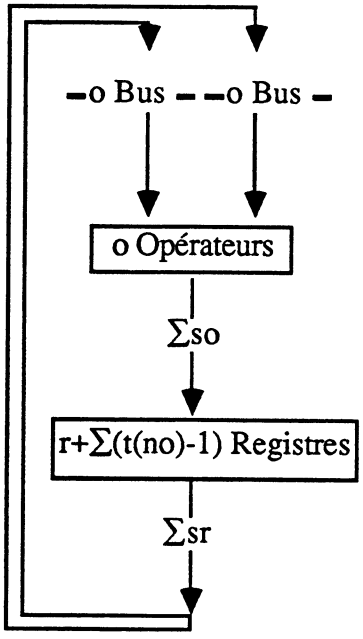
$\sum_{R} sr_i$ portes trois états sont nécessaires, et autant d'interconnexions pour les commander. Le modèle d'une PO à base de bus et de registres dédiés en sortie est donné figure IV.4(c).



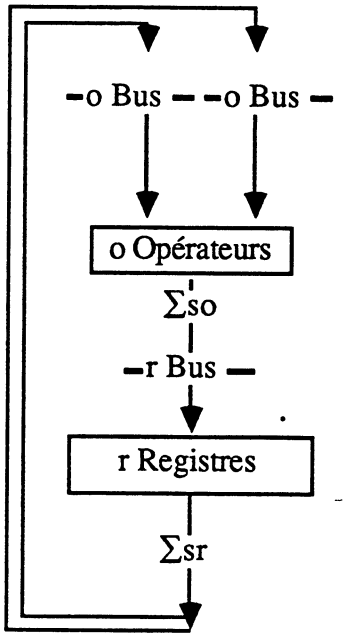
(a) Multiplexeurs



(b) Bus et Registres dédiés en entrée



(c) Bus et Registres dédiés en sortie



(d) Bus et Registres communs

Figure IV.4: Modèles d'interconnexion de PO pour différents styles

La surface d'une PO à base de registres dédiés en sortie est estimée par C_Sortie, défini par:

$$C\_Sortie = \sum_{X_i \in I,R,DR_{sortie}} Surf(X_i) + Surf(Porte) * \sum_{R_i \in R} sr_i + Surf(Conn) * (2 * \sum_{R_i \in R} sr_i + \sum_{o_i \in I} so_i + 2o)$$

Pour une architecture avec registres communs et pas de tampon, 2o bus en entrée d'opérateurs et r bus en entrée de registres sont a priori alloués. Les interconnexions d'une telle architecture sont

illustrées figure IV.4(d). Une estimation de la surface de la PO est donnée par C_Commun, défini comme suit:

$$C\_Commun = \sum_{X_i \in I,R} Surf(X_i) + Surf(Porte)*(\sum_{R_i \in R} sr_i + \sum_{o_i \in I} so_i) + Surf(Conn)*(2* \sum_{R_i \in R} sr_i + 2* \sum_{o_i \in I} so_i + 2o + r)$$

Pour une implantation de la partie opérative en tranches de bit, le nombre d'interconnexions à l'intérieur de la PO n'intervient pas, puisque les interconnexions se font par dessus les cellules de la PO. Seule la surface des interconnexions entre la PO et la PC intervient, c'est-à-dire le nombre d'interconnexions pour les commandes de multiplexeurs et de portes trois états. Les estimations de surface sont données par DP_Mux, DP_Entrée, DP_Sortie et DP_Commun de la manière suivante:

$$DP\_Mux = \sum_{X_i \in I,R} Surf(X_i) + \sum_{n_i \in N} Surf(n_i) + Surf(Conn)*(\sum_{n_i \in N} \lceil \log_2(in(n_i)) \rceil)$$

$$DP\_Entrée = \sum_{X_i \in I,R,DR_{entrée}} Surf(X_i) + \sum_{n_i \in N_o} Surf(n_i) + Surf(Porte)* \sum_{o_i \in I} so_i + Surf(Conn)*(\sum_{o_i \in I} so_i + \sum_{n_i \in N_o} \lceil \log_2(in(n_i)) \rceil)$$

$$DP\_Sortie = \sum_{X_i \in I,R,DR_{sortie}} Surf(X_i) + Surf(Porte)* \sum_{R_i \in R} sr_i + Surf(Conn)*(\sum_{R_i \in R} sr_i)$$

$$DP\_Commun = \sum_{X_i \in I,R} Surf(X_i) + Surf(Porte)*(\sum_{R_i \in R} sr_i + \sum_{o_i \in I} so_i) + Surf(Conn)*(\sum_{R_i \in R} sr_i + \sum_{o_i \in I} so_i)$$

Pour toutes les architectures, la surface de la partie de contrôle a été négligée. Ceci se justifie par le fait que tous les contrôleurs, quel que soit le style d'architecture, auront le même nombre d'états, les mêmes entrées et les mêmes transitions. Le seul paramètre qui change est le nombre de sorties du contrôleur (et donc le calcul de ces sorties). Ce nombre est estimé par le nombre d'interconnexions, entre la PO et la PC, dues spécifiquement à l'architecture choisie. Par conséquent, sachant que l'on ne souhaite pas une estimation exacte de la surface du circuit, mais une possibilité de comparer ces surfaces les unes par rapport aux autres, le fait de négliger la PC n'a pas d'influence sur la décision finale.

Pour comparer les différentes implantations d'architecture de la partie opérative, deux exemples ont été implantés selon plusieurs styles d'architecture sur les outils de COMPASS [COMP 91]. Il s'agit de celui de l'équation différentielle et de celui du filtre du 5ème ordre, présentés précédemment.
Pour l'exemple de l'équation différentielle sur 4 pas de contrôle et avec 5 registres, une implantation à base de multiplexeurs a pratiquement la même surface qu'une implantation à base de bus avec registres dédiés en sortie. L'architecture avec registres dédiés en sortie ne nécessite pas de duplication de registres. Un gain de 1,5% sur la surface du circuit est obtenu pour une architecture à base de multiplexeurs par rapport à une architecture à base de bus, sans optimiser le nombre de bus. Un gain de 6% est obtenu pour une architecture à base de bus optimisée, par rapport à une architecture à base de multiplexeurs. Cet exemple, du fait de sa faible complexité, ne donne pas de résultats très significatifs.

Pour ce qui concerne l'exemple du filtre du 5ème ordre, les résultats obtenus sont résumés dans le tableau IV.1. Ces résultats ont été obtenus pour la même assignation de registres et d'opérateurs, sans chercher à optimiser le nombre de bus, pour un ordonnancement sur 21 pas de contrôle, utilisant 2 additionneurs, 1 multiplieur et 11 registres. Dans ce tableau, chaque ligne donne les résultats obtenus pour chaque style d'architecture implanté: architecture à base de multiplexeurs, architecture à base de bus et de registres dédiés en entrée, architecture à base de bus et de registres dédiés en sortie et architecture à base de bus et de registres communs sans registre tampon. L'outil COMPASS ne contient malheureusement pas de bibliothèque de macro-blocs exécutant des opérations arithmétiques sur plusieurs bit. Un chemin de données se décrit schématiquement au niveau du bit et deux implantations sont alors possibles: l'une sur tranches de bit et l'autre sur cellules standards. Pour chacune de ces architectures, une ligne du tableau donne donc les résultats pour une implantation sur cellules standards et l'autre pour l'implantation en tranches de bit. Nous avons implanté ce circuit sur cellules standards pour éprouver l'estimation des connexions (en négligeant les connexions à travers les cellules standards "feedthrough").

Tableau IV.1: Résultats d'implantation du "filtre" pour différentes architectures: Architecture à base de multiplexeurs (Mux) ou de bus avec registres dédiés en entrée (Entrée) ou en sortie (Sortie) ou registres communs(Com). L'implantation finale est sur cellules standards (SC) ou sur tranches de bit (DP). Pour chacun de ces circuits est donnée la surface réelle totale (Totale), celle du corps (Corps), de la PC ainsi que le nombre de registres dupliqués (Reg. Dupli).

	Totale (µm2)	Corps (µm2)	PC (µm2)	PO (µm2)	Reg Dupli
Mux SC	20 886	11 903			0
Mux DP	24 967	12 442	1 895	4 070	0
EntréeSC	23 728	13 706			13
EntréeDP	37 297	17 525	2 140	13 768	13
Sortie SC	20 426	11 294			2
Sortie DP	33 374	15 703	1 650	11 949	2
Com. SC	20 807	11 755			0
Com. DP	33 595	17 711	1 747	12 857	0

Le tableau IV.2 donne le gain estimé calculé par les fonctions d'estimation, définies précédemment, et la différence entre le gain estimé et le gain réel.
Les schémas de ces différents circuits sont donnés en annexe III.

Tableau IV.2: Résultats d'implantation du "filtre" pour différentes architectures: Architecture à base de multiplexeurs (Mux) ou de bus avec registres dédiés en entrée (Entrée) ou en sortie (Sortie) ou registres communs(Com). L'implantation finale est sur cellules standards (SC) ou sur tranches de bit (DP). Pour chacun de ces circuits est donnée la surface estimée (Estim), l'ordre estimé des surfaces (Ordre esti) pour l'implantation SC et pour l'implantation DP, l'ordre des surfaces réelles (ordre réel) pour l'implantation SC et pour l'implantation DP. Le gain estimé (Gain esti) entre l'architecture à base de multiplexeurs et l'architecture concernée, et le gain réel (Gain Réel) entre ces architectures sont comparés (Diff).

	Surface Estim	Gain Estim (%)	Ordre esti SC	Ordre esti DP	Gain Réel (%)	Ordre réel SC	Ordre réel DP	Diff
Mux SC	503,1		2			2		
Mux DP	303,4			1			1	
EntréeSC	569,4	+ 12 %	4		+ 14 %	4		2 %
EntréeDP	425,9	+ 40 %		4	+ 41 %		4	1 %
Sortie SC	457,9	- 9 %	1		- 5 %	1		4 %
Sortie DP	346,9	+ 13 %		2	+ 21 %		2	8 %
Com. SC	553,7	+ 9 %	3		- 1%	2		10 %
Com. DP	398,3	+ 24 %		3	+ 25 %		3	1 %

En aucun cas les fonctions d'estimation de la surface du circuit, pour chaque style d'architecture, n'ont été définies afin d'estimer la surface du circuit, mais uniquement afin d'estimer si une architecture donnée est mieux adaptée à une description initiale qu'une autre. Ces fonctions d'estimation sont des aides au choix d'une architecture. Dans le tableau IV.2, elles impliquent des gains estimés, par rapport à une architecture à base de multiplexeurs, proches des gains réels; l'erreur d'estimation varie entre 1 et 10 %. L'ordre des surfaces estimées est identique à l'ordre des surfaces réelles à l'exception de l'architecture à base de bus et de registres communs qui donne une surface réelle similaire à celle de l'architecture à base de multiplexeurs, alors que la surface estimée est 9 % plus grande. Ces fonctions d'estimation permettent de définir rapidement si un style d'architecture est plus approprié qu'un autre, pour une application donnée.

IV.2.2.2. Espace de solutions

Les différentes implantation de la partie opérative permettent l'exploration d'un espace de solutions surface/vitesse. Pour l'exemple du filtre du 5ème avec une implantation sur cellules standards, cet espace est représenté figure IV.5.

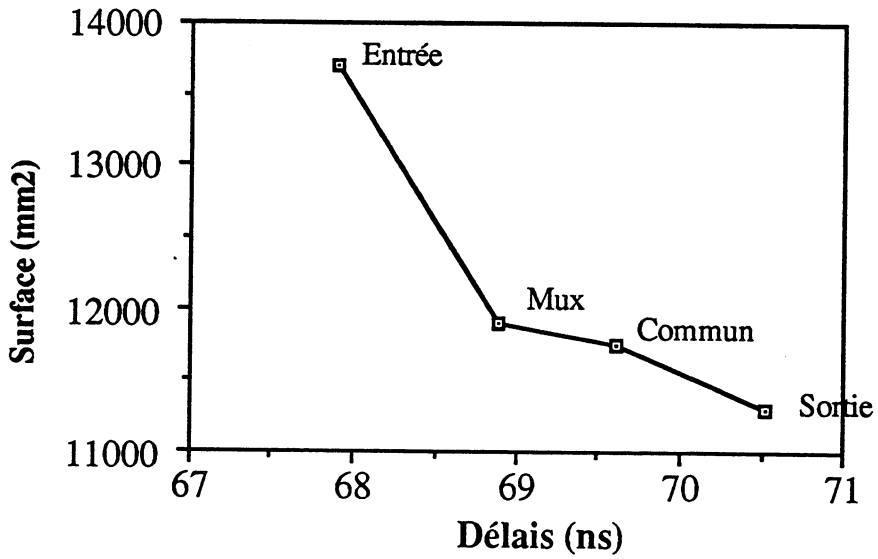


Figure IV.5: Espace de solutions pour les différentes architectures cibles pour l'exemple du filtre du 5ème ordre

IV.2.3. Extraction de la partie de contrôle

L'extraction de la partie de contrôle s'effectue par la mise à plat des cascades de sommets de test de l'organigramme de contrôle et par l'extraction des signaux de contrôle et des rapports de la partie opérative, en consultant la bibliothèque d'opérateurs [MIGN 92b], associée à l'organigramme de contrôle.

Après l'extraction du graphe de contrôle, le système génère une machine d'états finis, représentée par un graphe de Moore paramètré. Ce graphe est transmis à la synthèse de contrôleurs qui génère la partie de contrôle selon plusieurs architectures: architectures partitionnées (sur cellules standards, sur réseau de cellules programmables, sur ROM ou sur PLA) ou non partitionnées (sur cellules standards ou réseau de cellules programmables) [GERB 92] et avec plusieurs possibilités de codage (un parmi n ou compact) [DUFF 91] et différents critères (surface ou vitesse ou compromis entre la surface et la vitesse) pour la génération des blocs combinatoires. Ce choix permet de parcourir un espace de solutions surface/vitesse des contrôleurs. Un exemple d'espace de solutions, pour une spécification initiale dominée par le contrôle représentant un multiplieur série/parallèle sur 16 bit, est donné dans [MIGN 92c] et est représenté en figure IV.6.

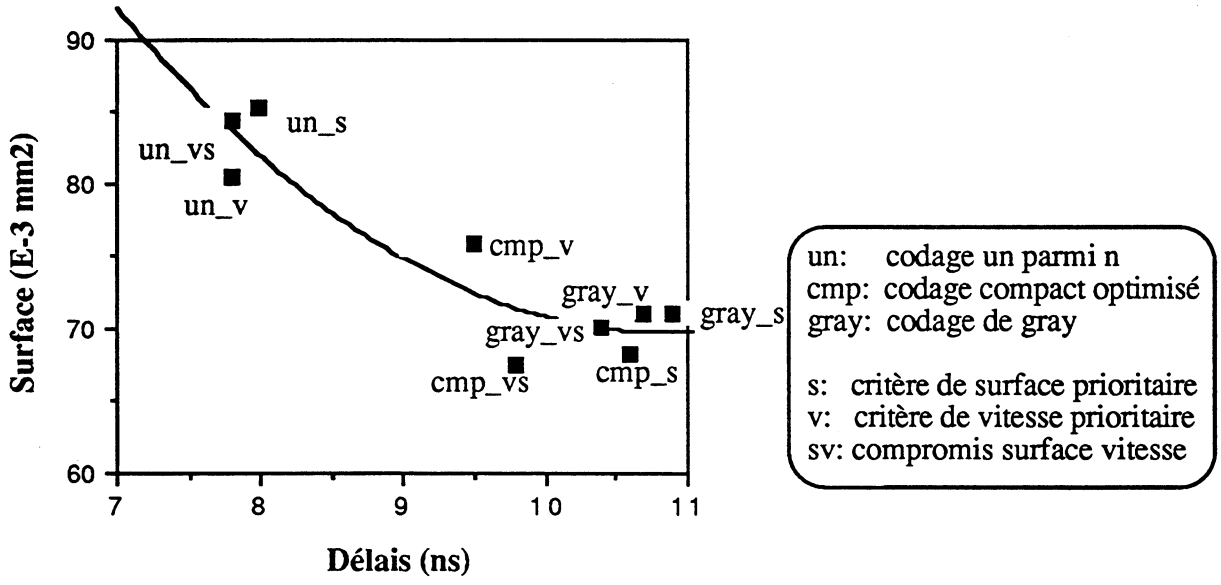


Figure IV.6: Espace de solutions, avec différents codage du contrôleurs et différents critères pour la synthèse logique, pour l'exemple du multiplieur

IV.2.4. Schéma d'horloge

Le schéma d'horloge implicitement défini pour la synchronisation entre la partie opérative et la partie de contrôle est un schéma d'horloge conventionnel. En un cycle d'horloge, la PC prend en entrée les entrées du contrôleur et la valeur de l'état courant et calcule l'état suivant et la valeur des sorties du contrôleur (les commandes de la PO), puis la PO travaille.

D'autres schémas d'horloge sont envisageables. Ce sont des schémas d'horloge dédiés soit à une PO lente, soit à un bloc de calcul de l'état suivant lent, soit à un bloc de calcul des sorties lent [SAFI 92].

Ces schémas d'horloge ont un comportement pipeliné. Ils impliquent un recouvrement entre le calcul de la PO et celui des blocs de la PC. De ce fait, des états factices sont insérés dans la machine d'états finis du contrôleur et des registres tampons sont ajoutés entre la PO et la PC. Ces schémas d'horloge ont une surface plus grande que celle d'un schéma conventionnel pour de meilleures performances.
Pour les sections précédentes, l'espace de solutions surface/vitesse n'était donné que pour un schéma d'horloge fixé. Le changement de schéma d'horloge permet de faire des "sauts" dans cet espace de solutions, comme l'illustre la figure IV.7 sur l'exemple du multiplieur déjà cité.

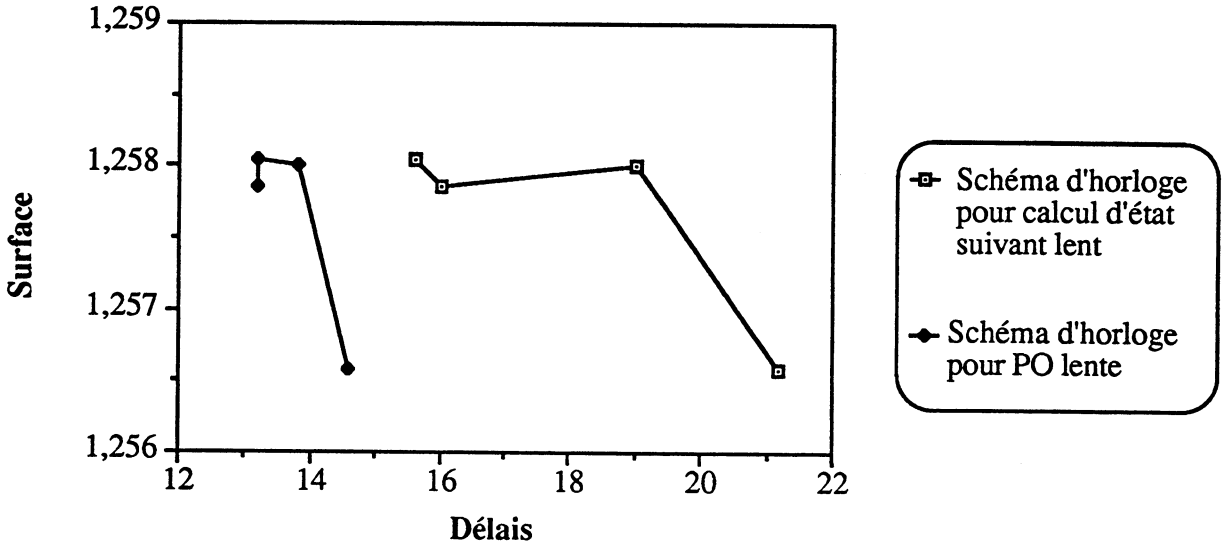


Figure IV.7: Espace de solutions pour différents schémas d'horloge pour l'exemple du multiplieur

IV.3. Conclusion sur le système de synthèse

Nous avons défini un système de synthèse à deux niveaux.
Une entrée du système sous forme de spécification algorithmique de haut niveau permet de traiter des spécifications dominées par les données souvent générées pour des processeurs dédiés au traitement numérique du signal. Les méthodes d'ordonnancement et d'assignation de ressources, définies dans cette thèse, laissent la possibilité à un concepteur de parcourir un espace de solutions pour différentes contraintes de temps et différentes architectures cibles de la partie opérative.
Une entrée au niveau de transferts de registres permet de réaliser les dernières étapes de la synthèse ou de traiter des spécifications dominées par le contrôle, par l'allocation des connexions et l'extraction d'un contrôleur. Elle autorise un parcourt de l'espace de solutions pour différentes architectures de la PO, différentes implantations du contrôleur et différents schémas d'horloge.
A ce niveau, des fonctions d'estimation de surface aident un concepteur dans le choix d'une architecture cible de la partie opérative. De la même façon l'expérience sur la synthèse de contrôleur permettrait de conseiller un concepteur dans le choix d'une architecture cible et d'un codage de la partie contrôle.

Conclusion

Dans cette thèse, nous avons proposé des heuristiques raisonnables pour réaliser une synthèse architecturale de circuits intégrés.

Nous nous sommes concentrés sur une méthode précise d'ordonnancement qui semblait intéressante dans le cas de spécification initiale représentant une boucle unique. En effet, ce cas est caractéristique du domaine d'application du traitement du signal et celui-ci est le plus susceptible d'aboutir à une utilisation fructueuse de la synthèse de haut niveau. Nous avons donc étudié cette méthode en profondeur et proposé quelques améliorations, tout en gardant comme objectif principal l'obtention de bons résultats en un temps réduit.

Concernant l'allocation de registres et d'opérateurs, nous avons proposé un algorithme de faible complexité s'appuyant sur la structuration en pas de contrôle définie au cours de l'ordonnancement et prédisant l'allocation des connexions. Cette méthode est compétitive par rapport à d'autres méthodes, elle a l'avantage d'être flexible par rapport aux styles d'architecture de partie opérative que nous avons répertoriés.

L'allocation des connexions recherche une optimisation en exploitant la commutativité des opérateurs et alloue les ressources nécessaires en correspondance avec chaque style d'architecture.

Comme l'ensemble de ce travail cherche à proposer des méthodes pragmatiques, il était important de conclure, dans une dernière étape, par la description d'une phase finale opérationnelle aujourd'hui et qui a l'intérêt d'être bien connectée au monde des outils réels. En effet, les opérateurs sont extraits de bibliothèques existantes, les compilateurs de chemins de données utilisés à bon escient et les contrôleurs finaux efficacement synthétisés.

L'ensemble de ce travail est donc inspiré par le souci de pragmatisme et d'efficacité. Les méthodes, tout en étant efficaces, sont de complexité contrôlée et enfin la connexion au monde réel de la conception de circuits intégrés est assurée.

Nous estimons que la synthèse de haut niveau ne deviendra une réalité que si ces précautions sont prises et nous pensons avoir démontré l'applicabilité de ce travail.

Annexe I

BIBLIOGRAPHIE

[AHO 88] A. Aho, R. Sethi, J. Ullman
"Compilers: Principles, Techniques and Tools"
Addison-Wesley Publishing Company, March 1988 (Edition originale 1986)

[ASCI 91] J. Van Eijndhoven, G. De Long, L. Stok
"The ASICIS Data Flow Graph: Semantics and Textual Format"
EUT Report 91 E 251, Eindhoven University of Technology Netherlands, June 1991

[AUTO 92] "Autologic VHDL User Manual" and Autologic Software version 8.1 Mentor graphic 1992

[BALA 88] M. Balakrishnan
"RT-Level Synthesis Based on Integrated Scheduling and Binding"
Technical report of the institut für informatik und Praktische Mathematik, Kiel, December 1988

[BERG 91a] R. Bergamaschi, R. Camposano, M. Payer
"Area and Performance Optimizations in Path-Based Scheduling"
European Design Automation Conference (EDAC 91), Feb 25-28 1991, Amsterdam, The Netherland, p 304-310

[BERG 91b] R. Bergamaschi, R. Camposano, M. Payer
"Scheduling under Resource Constraints and Module Assignment"
Integration, The VLSI journal, Vol 12, N°1, November 1991, pp 1-19

[BERG 91c] R. Bergamaschi, D. Allerton, K. Nichols, C. Jong, A. Currie
"SCHOLAR: A High Level Synthesis System for Concurrent VLSI Systems"
Internal Report, IBM Research Division, Yorktown Height NY USA, 1991

[BERT 92] M-C Bertrand, M. Crastes, A. Mignotte
"VHDL Subset for Control Dominated RTL Synthesis using Library Blocks"
IFIP Workshop on Control Dominated Synthesis from a RTL Description, Grenoble France, 3 au 4 Sept 1992, p. 162 à 169

[BLAZ 87] J. Blazewicz
"Selected Topics in Scheduling Theory"
Annals of Discrete Mathematics 31, 1987, Elsevier Science Publishers B.V., North Holland.

[BLAZ 91] J. Blazewicz
"Mathematical Programming Formulations fo Machine Scheduling : A Survey"
European Journal of Op. Res. 51, 1991, pp. 283-300.

[BOUB 90] A. Boubeker, G. Saucier
"Automatic generation of massively parallel low-level image processing ASICs"
IEEE/ACM 5th International Workshop on High-level Synthesis, March 3-6, 1991, Bühlerhöle, Germany.

[BOUB 92a] A. Boubeker, A. Mignotte, G. Saucier
"Synthesis of Low Level Image Processing Circuits"
AFECT/GCIS Workshop sur la synthèse et compilation d'architectures: Méthode de spécification et de conception, Grasse France, January 23-24, 1992, pp 17-18

[BOUB 92b] A. Boubeker
"Conception d'architectures intégrées de traitement d'image de bas niveau"
Thèse de Docteur de l'INPG, spécialité Microélectronique, 4 Mars 1992.

[CALL 92] CALLAS Reference Manual, 1992

[CAMP 88] R. Camposano
"Design Process Model in the Yorktown Silicon Compiler"
25th Design Automation Conference (DAC), June 1988, Anaheim CA

[CAMP 89] R. Camposano, W. Rosentiel
"Synthesizing Circuits From Behavioral Descriptions"
IEEE Trans on Computer Aided Design, vol 8, N°2, Feb 1989

[CAMP 90a] R. Camposano, R. Bergamaschi
"Synthesis Using Path-Based Scheduling: Algorithms and Execirces"
27th Design Automation Conference (DAC), Orlando FL, June1990, p450-455

[CAMP 90b] R. Camposano, R. Bergamaschi
"Redesigning Using State Splitting"
European Design Automation Conference (EDAC 90), March 12-15 1990,
Glasgow, Scotland, p157

[CARL 89] J. Carlier, F. Pinson
"An algorithm for solving the job shop problem"
Management Science, Feb. 89, p.164.

[CATT 90] F. Catthoor
"The effect of the Hardware Sharing Factor on the Selection of an Architectural Style for Real-time signal processing"
Internal report IMEC

[CHU 89] C-M. Chu, M. Potkonjak, M. Thaler, J. Rabaey
"HYPER: An Interactive Synthesis Environment for High-Level Real Time Applications"
IEEE Int. Conf. on Computer Design: VLSI in Computer & Processors (ICCD),
Cambridge MA, Oct 1989, p432

[CLOU 88] E. Cloud
"The Geometric Arithmetic Parallel Processor"
The 2nd Symp on the Frontier of Massively Parallel Computation, FairFax (VA),
pp 373-381, Oct. 1988.

[COMP 91] COMPASS Design Automation, "ASIC Synthesizer for VHDL Design", "Logic Assistant", "Chip Assistant", "Datapath Compiler", Reference Manual, v8r3, 1991.

[CUSH 88] B. Cushman
"GaAs Technology meets RISC Architectures"
VLSI System Design, Sept 88, pp 68-77

[DEMA 86] H. DeMan et al
"Cathedral II : A silicon compiler for digital signal processing."
IEEE design and test, Dec 86, p. 13-25.

[DEPU 90a] F. Depuydt
"Clustering Techniques for Register Optimization and Scheduling in High Level Synthesis"
Internal Report, IMEC 1990

[DEPU 90b] F. Depuydt, G. Goosens, J. van Meerbergen, H. DeMan
"Scheduling of Large Signal Flow Graphs Based on Metric Graph Clustering"
IFIP Working Conference on Logic and Architecture Synthesis,
Paris, May 1990

[DEVA 87] S. Devadas, S.Newton
"Data Path Synthesis from Behavioral Description : An Algorithmic Approach."
ICSAC, 1987, pp.398-401.

[DUFF 91] C. Duff, G. Saucier
"State assignment based on reduced dependency theory and recent experimental results"
IEEE International Conference on Computer Aided Design (ICCAD 91), Nov. 11-14, 1991, pp. 222-225.

[ERSC 79] J. Erschler,G. Fontan, F. Roubellat
"Potentiels sur un graphe non conjonctif et analyse d'un problème d'ordonnancement à moyens limités"
RAIRO Op Res, vol. 13, No 4, Nov. 79, pp. 363-378.

[EWER 90] C. Ewering
"Automatic High Level Synthesis of Partioned Busses"
IEEE Int Conf on Computer Aided Design (ICCAD 90), Santa Clara California, Nov 90, pp3004-307

[FARL 86a] M. McFarland
"BUD : bottom-up design of digital systems"
23rd Design Automatomation Conference, June 86, Las Vegas, NV, p 474-479.

[FARL 86b] M. Mc Farland
"Using Bottom-Up Design Techniques in the Synthesis of Digital Hardware from Abstract Behavioral Descriptions"
23rd Design Automation Conference (DAC), Las Vegas NV, June 1986, pp 744-751

[FARL 87] M. Mc Farland
"Reevaluating the Design Space for Register-Transfer Hardware Synthesis"
IEEE Int Conf on Computer Aided Design (ICCAD), Santa Clara CA, Novemb. 1987, pp 262-265

[FARL 88] M.Mc Farland, A.Parker, R.Camposano
"Tutorial on High level synthesis"
25th Design Automation Conference (DAC), June1988, Anaheim CA

[FAUR 76] R. Faure, C. Roucairol, P. Tolla
"Chemins, Flots, et Ordonnancements"
Ed. Gauthiers villars, Paris, 1976.

[FONK 89] A-B. Fonkoua
"Une application de l'intelligence artificielle à la synthèse architecturale des circuits intégrés"
Thèse de Docteur de l'INPG, 4 Octobre 1989.

[GAJS 83] D. Gajski, R. Kuhn
"Guest Editors' introduction: New VLSI Tools", IEEE COmputer, vol 16, N° 12, Dec 1983, pp 11-14

[GAJS 91] D. Gajski
"Essential issues and possible solutions in high level synthesis"
Technical Report, University of California Irivine, No 91-18, 1991

[GARE 79] M.R. Garey, D.S. Johnson
"Computers and Intractability: A Guide to the Theory of NP-Completeness"
Ed. W.H. Freeman and Co., New-York, 1979.

[GERB 92] L. Gerbaux, G. Saucier
"Automatic synthesis of large Moore sequencers"
IEEE European Conference on Design Automation (EDAC'92), March 16-19, 1992, pp. 237-244, Brussels, Belgium.

[GIRC 85] E.F. Girczyc, R.J. Buhr, J.P. Knight
"Applicability of a Subset of ADA as an Algorithmic Hardware Description Language for Graph Based Hardware Compilation"
IEEE Trans. on Computer Aided Design, vol. CAD-4, no. 2, Avril 1985.

[GIRC 87] E. F. Girczyc
"Loop Winding, A Data flow Approach to Functional Pipelining"
Int'l Symposium on Circuits and systems (ISCAS 87), Philadelphia May 1987, p382-385

[GLUN 90] W. Glunz and G. Umbreit, "VHDL for High Level Synthesis of digital systems", 1rst European Conference on VHDL Methods, 1991.

[GOND 79] M. Gondran, M. Minoux
"Graphes et Algorithmes"
Eyrolles 1979 édition 1985, Collection de la direction des Etudes et Recherches d'Electricité de France.

[GOOS 87] G. Goossens, J. Rabaey, J. Vandewalle, H. DeMan
"An Efficient Microcode-Compiler for Custom DSP Processors"
IEEE Int Conf on Computer Aided Design (ICCAD), Santa Clara CA, Novemb. 1987, p24-27

[GOOS 89a] G. Goossens, J. Vandewalle, H. DeMan
"Loop optimization in register-transfer scheduling for DSP-systems"
26th Design Automation Conference (DAC), June 1989, Las Vegas, p 826-831

[GOOS 89b] G. Goossens
"Optimisation Techniques for Automated Synthesis of Application Specific Signal Processign Architectures"
PhD Thesis, IMEC, University of Leuven, June 1989

[GOUL 90] A. Goulahsen
"GENARC Un système expert en synthèse architecturale pour circuits dédiés au traitement du signal"
Thèse de Docteur de l'INPG, 3 Juillet 1990.

[GRAS 90] W. Grass
"A Branch and Bound Method for Optimal Transformation of Data Flow Graphs for Observing Hardware Constraints"
European Design Automation Conference (EDAC 90), March 12-15 1990, Glasgow, Scotland, p 73

[GROV 84] S. Groves
"A Comparison of the Internal Architectures of the Motorola MC6800, 68881 and 86020"
IEEE Int. Conference on Computer Design (ICCD 84), Oct 84, pp 211-215

[HEIJ 90] M. Heijligers
"Time Constrained Scheduling for High Level Synthesis"
Master Thesis, Eindhoven University, May 1991

[HITC 83] C. Hitchcock, D.E. Thomas
"A Method for Automatic Data Path Synthesis"
20 th Design Automation Conference (DAC), July 1983, p 483-489

[HU 61] T.C. Hu
"Parallel Sequencing and Assembly Line Problems"
Opns. Res. 9, 1961, pp. 841-848.

[HUAN 90] C-Y. Huang, Y. Chen, Y. Lin, Y-C Hsu
"Data Path Allocation Based on Bipartite Weighted Matching"
27th Design Automation Conference (DAC), Orlando FL, June 1990, p 499-504

[HWAN 90] C-T. Hwang, Y. Lin, Y-C. Hsu
"Optimum and Heuristic Data Path Scheduling under Resource Constraints"
27th Design Automation Conference (DAC), Orlando FL, June 1990, pp 65-70

[HWAN 91] C-T. Hwang, J-H. Lee, Y-C. Hsu
"A Formal Approach to the Scheduling Problem in High Level Synthesis"
IEEE Trans. on Computer Aided Design Vol 10, No 4, April 1991, pp 464-475

[JAMI 85] R. Jamier, A. Jerraya
"Appolon A Datapath Silicon Compiler"
IEEE Int Conf on Computer Aided Design (ICCAD 85)

[JANA 90] J. Janakiraman, B. Mitra
"Automated Synthesis of Data Path Based on Integer Programming Algorithm"
IEEE-CS TC-MICRO, MICROARCH, Vol 5, N°1, April 1990

[JELE 89] J. Jelemensky, V. Goler, B. Burgess, J. Eifert, G. Miller
"The MC68332 Microcontroller"
IEEE MICRO Chips, Systems, Software and Applications, vol 9, No 4 August 1989, pp 31-50

[KERN 70] B. W. Kernighan, S. Lin
"An efficient Heuristic Procedure for partitioning graphs"
Bell System Technical Journal, February 1970, pp 291-307

[KIRK 83] Kirkpatrick, Gelatt, Vecchi
"Otimization by Simulated Annealing"
Science, 13 May 1983, Vol 220, N° 4598, pp 671-680

[KOHN 89] L. Kohn, N. Margulis
"Introducing the Intel i860 64-bit Microprocessor"
IEEE MICRO Chips, Systems, Software and Applications, vol 9, No 4 August 1989, pp 15-30

[KRAM 88] H. Krämer, M. Neher, G. Riestsche, W. Rosenstiel
"DataPath and Control Synthesis in the CADDY System"
Int. Workshop on Logic and Arcitecture Synthesis for Silicon Compilers
Grenoble, May 1988

[KRAM 89] H. Krämer, W. Rosenstiel
"Synthesis of Multi-Processor Architectures from Behavioral Descriptions"
IEEE/ACM 4th Workshop on High Level Synthesis, Portland Maine, October 1989

[KRAM 90] H. Krämer, W. Rosenstiel
"System Synthesis using Behavioural Descriptions"
European Design Automation Conference (EDAC 90), March 12-15 1990,
Glasgow, Scotland, p 277

[KUCU 89] K. Küçükçakar, A. C. Parker
"Data Path Tradeoffs using MABAL"
27 th Design Automation Conference (DAC), Orlando FL, June 1990, p 511-516
IEEE/ACM 4th Workshop on High Level Synthesis, Portland Maine, October 1989

[KUNG 85] S. Kung, H. Whitehouse, T. Kailath
"VLSI and Modern Signal Processing"
Prentice Hall Information and Systems Sciences Series, 1985, pp 258-264

[KURD 87] F. J. Kurdahi, A. C. Parker
"REAL: A Program for Register Allocation"
24th Design Automation Conference (DAC), June 1987, pp 210-215

[LAHR 82] A. Lahrichi
"Ordonnancements. La notion de "parties obligatoires" et son application aux problèmes cumulatifs"
RAIRO Op. Res., vol.16, No 3, Août 1982, pp.241-262.

[LAWE 76] L. Lawer
"Sequencing to minimize the weighted number of tardy jobs"
Rev Française Automat. Informat. Recherche Opérationnelle Ser., Bleue 10.5, 22-33 A5.1, 1976

[LEE 89] J. Lee, Y. Hsu, Y. Lin
"A New Integer Linear Programming Formulation for the Scheduling Problem in Data Path Synthesis"
Digest of Technical Papers of the International Conference of Computer Aided Design 1989, Novembre 1989, pp 20-23

[LENS 78] J. Lenstra, A. Rinnooy Kan
"Complexity of Scheduling under Precedence Constraints"
Operations research, vol 26, No1, Feb 78, pp 22-35

[LIPP 88] P. Lippens
"Force directed scheduling as an allocation algorithm"
Internal report Methods Team IMEC, Dec 1988

[LUI 91] T. Lui, Y-L Lin
"FLORA: A Data Path Allocator Based on Branch and Bound Search"
The VLSI Journal 11, 1991, pp 43-66

[LY 90] T.A. Ly, W.L. Elwood, E.F. Girczyc
"A Generalized Interconnect Model Data Path Synthesis"
27th Design Automation Conference (DAC), Orlando FL, June 1990, p 168-173

[MAHM 92] M. Mahmood, B. Sharma, C. Kingsley
"A Datapath Synthesizer for High Performance ASICs"
IEEE Custom Integrated Circuits Conference, Boston MA, May 3-8 1992

[MAHM 92] M. Mahmood, B. Sharma, C. Kingeley
"A Datapath Synthesizer for High Performance ASICs"
IEEE Custom Integrated Circuits Conference, Boston MA, May 3-6 1992

[MARW 86] P. Marwedel
"A new synthesis algorithm for the MIMOLA software system."
23rd Design Automatomation Conference, Las Vegas, NV, July 86, p.271-277.

[MARW 92] P. Marwedel
"Implementation of IF Statements in the TODOS microarchitecture synthesis systems"
IFIP Workshop on Control Dominated Synthesis from a Register Transfert Level Desciption, Grenoble France, 3-4 Sept 1992, pp 107-116

[MARZ 89] S. März, K. Buchenrieder, P. Duzy, R. Kumar, T. Wecker
"CALLAS A system for Automatic Synthesis of Digital Circuits from Algorithmic Behavioral Descriptions"
IEEE conference EuroASIC 89, Grenoble, January 89, pp 131-142.

[MIGN 90a] A. Mignotte, G. Saucier
"Matching method for concurrent operator, register and multiplexer allocation"
SASIMI '90, 23-25 Oct. 90, p. 103-110, Tokyo, Japon.
also published in
VLSI Logic Synthesis and Design, Edited by R.W. Dutton, IOS Press, 1991

[MIGN 90b] A. Mignotte, G. Saucier
"A Generalized Model for Resource Assignment"
IEEE/ACM 5th Int. Workshop on High Level Synthesis, Bühlerhöhe Germany, Marsh 90

[MIGN 91] A. Mignotte, M. Crastes de Paulet
"Resource Assignment with Different Target Architectures"
IEEE conference EuroASIC'91 International Conference, May 27-31, 1991, Paris, France, pp. 172-177

[MIGN 92a] A. Mignotte
"Operator and Register Assignment with Differerent Architecture Styles"
AFECT/GCIS Workshop sur la synthèse et compilation d'architectures: Méthode de spécification et de conception, January 23-24, 1992, Grasse France, pp 25-26

[MIGN 92b] A. Mignotte, M. Crastes de Paulet
"Format for Macro Block Description and Connection to a RTL Synthesis Tool"
WG 10.5 IFIP Workshop on Synthesis, Generation and Portability of Library Blocks for ASIC Design, 12-13 March 1992, Grenoble France, pp 220-226

[MIGN 92c] A. Mignotte, M-C Bertrand, M. Crastes de Paulet, J. Fron, J. Rampon
"Interactive RTL Synthesis Using Macro Block Library"
IEEE conference EuroASIC'92 International Conference, Paris France, May 30, 1992

[NOTE 89] S. Note, J. V. Meerbergen, F. Catthoor, H. DeMan
"hardwired Data Path Synthesis for High Speed DSP Systems with the Cathedral III Compilation Environment", Logic and Architecture Synthesis for Silicon Compilers, G. Saucier et P. M. McLellan (eds), North Holland, 1989, pp 243-254

[NOTE 91] S. Note, W. Geurts, F. Catthoor, H. DeMan
"Cathedral III: Architecture-Driven High Level Synthesis for High Throughput DSP Applications"
28th Design Automation Conference (DAC), June 1991, San Francisco

[PANG 87] B.M. Pangrle, D.D. Gajski
"Slicer : a state synthesizer for intelligent silicon compilation"
Digest of the IEEE International Conference on Computer Aided Design 1987,p.42,1987.

[PAPA 82] C. H. Papadimitriou, K. Steiglitz
"Combinatorial Optimization"
Prentice Hall, 1982

[PAPA 90] C. Papachristou, H.Konuk
"A Linear Program Driven Scheduling and Allocation Method Followed by An Interconnect Optimization Algorithm."
27th Design Automation Conference (DAC), Orlando FL, June1990, paper 5.4.

[PARK 86] A.C. Parker, J. Pizarro, M. Mlinar
"MAHA: A Program for Data Path Synthesis"
23rd Design Automation Conference (DAC),Las Vegas NV, June 1986, pp 744-751

[PARK 88] N. Park, A.C. Parker
"SEHWA: A Software Package for Synthesis of Pipelines from Behavioral Specifications"
IEEE Trans on Computer Aided Design, vol7, N°3 March 1988, pp. 256-370

[PARK 92] I. Park, K. O'Brien, A. Jerraya
"Tutorial: Amical: Interactive Architectural Synthesis Based on VHDL"
Internal report, TIM3 INPG, Grenoble France, March 1992

[PAUL 86] P. Paulin, J. Knight
"HAL a Multi-Paradigm approach to automatic dataPath Synthesis"
23rd Design Automation Conference (DAC), Las Vegas NV, June 1986, pp263-270

[PAUL 87] P. Paulin, J. Knight
"Force Directed Scheduling in automatic dataPath Synthesis"
24th Design Automation Conference (DAC), 1987, Miami Beach, FLA, pp 195-202

[PAUL 88] P. Paulin
"High Level Synthesis of Digital Circuits Using global Sheduling and Binding Algorithms"
PhD Thesis, faculty of engineering, Carleton University, Canada, January 1988

[PAUL 89a] P. Paulin, J. Knight
"Algorithms for High Level Synthesis"
IEEE Design and Test of Computers, Dec. 1989, pp 18-31

[PAUL 89b] P. Paulin, J. Knight
"Force Directed Scheduling for Behavioral Synthesis of ASICs"
IEEE Transaction on Computer Aided Design of Integrated Circuit, June 1989, vol 8, No 6, pp 661-679

[PAUL 89c] P. Paulin
"Scheduling and Binding Algorithms for High Level Synthesis"
26th Design Automation Conference (DAC), June 1989, Las Vegas, paper 2.1

[PAUL 90] P. Paulin
"Global Scheduling Techniques for Behavioral Synthesis"
Submitted to European Design Automation Conference (EDAC 90), March 12-15 1990, Glasgow, Scotland

[PAUL 92] P. Paulin
"A Formulation of the Force Directed Scheduling Algorithm with $O(c.n^2)$ complexity"
BNR Internal Report No 5E35-9207, July 1992

[POTA 90] R. Potasman, J. Lis, A. Nicolau, D. Gajski
"Percolation Based Synthesis"
27th Design Automation Conference (DAC), Orlando FL, June 1990, p 444-449

[POTK 89] M. Potkonjak, J. Rabaey
"A Scheduling and Resource Allocation Algorithm for Hierarchical Signal Flow Graphs"
26th Design Automation Conference (DAC), June 1989, Las Vegas, p 7

[ROY 66] B. Roy
"Prise en compte des contraintes disjonctives dans les méthodes de chemin critique"
Revue française de recherche opérationnelle, No. 38, 1966, pp. 69-84.

[SAFI 90] A. Safir, B. Zavidovique
"Toward a Global Solution to High Level Synthesis problems"
European Design Automation Conference (EDAC 90), March 12-15 1990, Glasgow, Scotland, p 283

[SAFI 92] C. Safinia, R. Leveugle, A. Mignotte, G. Saucier
"Clocking Scheme Selection and its Impact on the Solution Space Exploration in RTL and High Level Synthesis"
Rapport interne CSI/INPG, Octobre 1992.

[SHAN 89] R. Shankar, E. Fernandez
"VLSI and Computer Architecture"
VLSI Electronics Microstructure Science, Volume 20, Academic Press Inc., Edited by Norman Eindpruch, 1989.

[SPRI 90] D. Springer, D. Thomas
"Exploiting the Special Structure of Conflict and Compatibility Graphs in High Level Synthesis"
IEEE Int Conf on Computer Aided Design (ICCAD), Santa Clara Ca., November 1990, pp 254-257

[STOK 88] L. Stok, R. van den Born
"EASY: Multiprocessor Architecture Optimization"
Int. Workshop on Logic and Architectural Synthesis, Grenoble France, May 1988

[STOK 91] L. Stok
"Architectural Synthesis and Optimization of Digital Systems"
PhD Thesis, Technological University of Eindhoven, 10 July 1991.

[STOL 92] A. Stoll, P. Duzy
"High Level Synthesis from VHDL with Exact Timing Constraints",
29 th DAC, 1992, pp 188-193

[SYNO 91] SYNOPSYS Design Compiler Reference Manual V2.0. March 1991.

[TSEN 83] C-J. Tseng, D. Siewiorek
"FACET: A Procedure for the Automated Synthesis of Digital Systems"
20th Design Automation Conference (DAC), July 1983, pp 490-496

[TSEN 86] C-J. Tseng, D. Siewiorek
"Automated Synthesis of Datapath in Digital Systems"
IEEE transaction on Computer Aided Design, July 1986

[TUTO 90] S. Marsz, P. Dutzi, W. Rosenstiel
"High level synthesis Principles, Algorithms and exercices"
IFIP Working Conference on logic and Archictectural synthesis
Paris 1990, Palais des congrés

[VERH 91] W. Verhaegh, E. Aarts, J. Korst, P. Lippens
"Improved Force Directed Scheduling"
European Design Automation Conference (EDAC 91), Feb 25-28 1991,
Amsterdam, The Netherland, paper 8A.1.

[VERH 92] W. Verhaegh, P. Lippens, E. Aarts, J. Korst, J van Meerbergen, A. van der Werf
"Modeling Periodicity by PHIDEO Streams"
IEEE/ACM 6th International Workshop on High Level Synthesis, Lagina Miguel,
CA, USA, Nov. 4-6 1992

[VHSU 92] "VHDL Subset for RTL Synthesis: Preliminary Version", Rapport interne
INPG/CSI, 3 Mars, 1992.

[WEHN 91] N.Wehn, J.Biesenack, M.Pilsl,
"A New Approach to Multiplexer Minimization in the CALLAS Synthesis
Environment",
VLSI 91, paper 5.2, August 91, Edinburgh Scotland.

[ZEGE 90] I. Zegers, P Six, J. Rabaey, H. DeMan
"Automatic Generation of Controllers in the CathedralII Silicon Compiler"
European Design Automation Conference (EDAC 90), p617

Annexe II

INDEX, NOTATION DES VARIABLES ET GLOSSAIRE

Table d'index

Glossaire

APTA	(ordonnancement) Au Plus TArd
APTO	(ordonnancement) Au Plus TOt
APVI	(ordonnancement) Au Plus VIte
GDD	Graphe de Dépendance de Données
GFC	Graphe de Flot de Contrôle
GP	Graphe de Précédence
GD	Graphe de Distribution
OCD	Ordonnancement sous Contraintes de Décision
OCM	Ordonnancement sous Contraintes de Ressources
OCT	Ordonnancement sous Contraintes de Temps
OPF	(ordonnancement) Orienté Par les Forces
OPF_Glob	(ordonnancement) OPF avec anticipation Globale
OPF_Loc	(ordonnancement) OPF avec anticipation Locale
OPF_Màj	(ordonnancement) OPF avec Mise à Jour successive des GD
PC	Partie de Contrôle (du circuit)
PO	Partie Opérative (du circuit)
RISC	Reduced Instruction Set Computer
RTL	Register Transfer Level (Transferts de Registres)
UAL	Unité Arithmétique et Logique

Notation des variables

0(m.n)	Ordre de la complexité pour m pas de contrôle et n opérations
0(m.r)	Ordre de la complexité pour m pas de contrôle et r registres
APTA(t)	Date d'ordonnancement au plus tard de t
APTO(t)	Date d'ordonnancement au plus tôt de t
A_R	Ensemble d'arêtes dues à l'application d'une ou plusieurs règles d'assignation de registres dans le graphe biparti d'assignation de ressources
$ASSIGN_{ins}(t)$	Fonction déterminant l'assignation de l'instance ins et qui vaut 1 si t est assignée à ins, et 0 sinon
AX	Ensemble des nœuds XOR ou AND
B_i	Bloc de base
Bl_i	Boucle élémentaire
BV_i	Bus Virtuel
c	Durée du pas de contrôle ou du cycle d'horloge
$Ch(t_o,t,p)$	Coût du chaînage créé sur t_o et dû à l'ordonnancement de t à la date p
Conn	Connexion
d(t)	Durée de l'opération t en nano-secondes
$DR_{entrée}$	Nombre de registres à dupliquer pour une architecture à base de registres dédiés en entrée d'opérateurs
DR_{sortie}	Nombre de registres à dupliquer pour une architecture à base de registres dédiés en sortie d'opérateurs
$FCh_{to}(p,t)$	Force induite sur t_o due à l'assignation de l'opération t à la date p en intégrant un coefficient de coût de chaînage
$Fr(t_o,t,p)$	Fréquence d'apparition du chaînage créé avec l'opération t_o dû à l'ordonnancement de t à la date p
$F_{to}(p,t)$	Force induite sur t_o due à l'assignation de l'opération t à la date p
FTot(p,t)	Force totale due à l'assignation de l'opération t à la date p
f_and	Nœud de type AND d'un arbre AND/XOR défini pour les branchements conditionnels
$GD_{op}(p)$	Graphe de Distribution pour l'opérateur de type op au pas de contrôle p
I	Ensemble des instances d'opérateurs
L(Ord)	Longueur totale de l'ordonnancement Ord
l(t)	Longueur de l'opération t en nombre de pas de contrôle
MB	Ensemble des macro-blocs représentant des boucles élémentaires
$m_{op}(Ord)$	Largeur totale en surface d'opérateur de l'ordonnancement Ord

Nb(op)	Nombre minimal d'instances d'opérateurs de type op
O	Ensemble des types (ou classes) d'opérateurs
op	Opérateur (type d'opérateur) existant en bibliothèque
op(t)	Fonction retournant l'opérateur sélectionné pour t
op_i, O_i ou o_i	Instance d'opérateur
Ord(t)	Date d'ordonnancement de l'opération ou de l'élément de stockage t
p	Pas de contrôle
Prec(t)	Ensemble des opérations précédentes de t dans le GP
Prob(t,p)	Probabilité d'allouer l'opération élémentaire t au pas de contrôle p
R	Ensemble des registres
$R_D{}^{p,O_1}_{i,O_2}$	Opérande qui est émise par O_1 et stockée dans le registre R_i , au pas de contrôle p, et qui sera lue à l'entrée droite de l'opérateur O_2. R_G désigne une opérande en entrée gauche.
R_i	Registre
$R_i^{p,O}$	Opérande émise par O au pas de contrôle p et stockée dans le registre R_i
S_E	Ensemble d'éléments exclusifs associés à un nœud XOR
s_e	Elément exclusif
$SELEC_{op}(t)$	Fonction déterminant l'opérateur sélectionné pour l'opération t qui vaut 1 si op est sélectionné pout t, et 0 sinon
S_i	Elément de stockage virtuel
SO	Ensemble des opérandes
so_i	Nombre de registres différents en sortie de l'instance d'opérateur o_i
$S_{op}(p)$	Ensemble des opérations ordonnancées à la date p
$S_R(p)$	Ensemble des éléments de stockage virtuels ordonnancés à la date p
sr_i	Nombre d'entrées d'opérateurs différentes en sortie du registres R_i
SSO	Sous-ensemble d'opérandes
Suiv(t)	Ensemble des opérations suivantes de t dans le GP
Surf(x)	Surface de la ressource x
T	Ensemble des opérations élémentaires (ou des sommets qui les représentent)
t ou t_i	Opération élémentaire (ou le sommet qui la représente)
U	Ensemble d'arcs, le plus souvent de dépendance de données
v_i	Variable
W	Pondération des arêtes dans le graphe biparti d'assignation de ressources
xor	Nœud de type XOR d'un arbre AND/XOR défini pour les branchements conditionnels
$x_{to}(p',t,p)$	Variation de la probabilité de l'opération t_o au pas de contrôle p' si on ordonnance t au pas de contrôle p

Annexe III

IMPLANTATION D'UN EXEMPLE POUR DIFFERENTES ARCHITECTURES

Cet exemple est le benchmark du filtre elliptique du 5ème ordre. Il a été implanté pour un ordonnancement sur 21 pas de contrôle avec 11 registres (r=11), 2 additionneurs et 1 multiplieur non pipeliné (o=3). L'assignation de registres et d'opérateurs utilisée est donnée figure III.1. Pour cette assignation, les paramètres sont:

N_O = {1 de taille 6, 2 de taille 5 et 2 de taille 4}

N_r = {7 de taille 2 et 1 de taille 3}

$\sum_R sr_i = 24$

$\sum_O so_i = 12$

2 add, 1 Mult, 11 registres

Nous avons estimé les surfaces en nombre de portes équivalentes. Nous avons pris comme valeur pour Surf(Conn), la surface d'une connexion, la valeur suivante:

Surf(Conn) = (Surf_Moyenne / Nb_Conn_Moyen) * Coef_Rout_Moyen, où Surf_Moyen est le côut moyen de l'ensemble des ressources, Nb_Conn_Moyen est le nombre moyen de connexions et Coef_Rout_Moyen est le coefficient moyen de routage sur l'ensemble des styles.

Dans les pages suivantes se trouvent, une figure du GDD du filtre du 5ème ordre ordonnancé sur 2& pas de contrôle, les figures de la connexion entre la PO et la PC, le schéma de la PO, le layout pour une implantation sur PO en tranche de bit et le layout pour une implantation sur PO en cellules standards, pour différents styles d'architecture. Les styles darchitectures essayés étant ceux du tableau 4.1 du chapitre 4, c'est à dire, une PO à base de multiplxeurs, une PO à base de bus et registres dédiés en sortie, une PO à base de bus et registres dédiés en entrée et une PO à base de bus et registres communs.

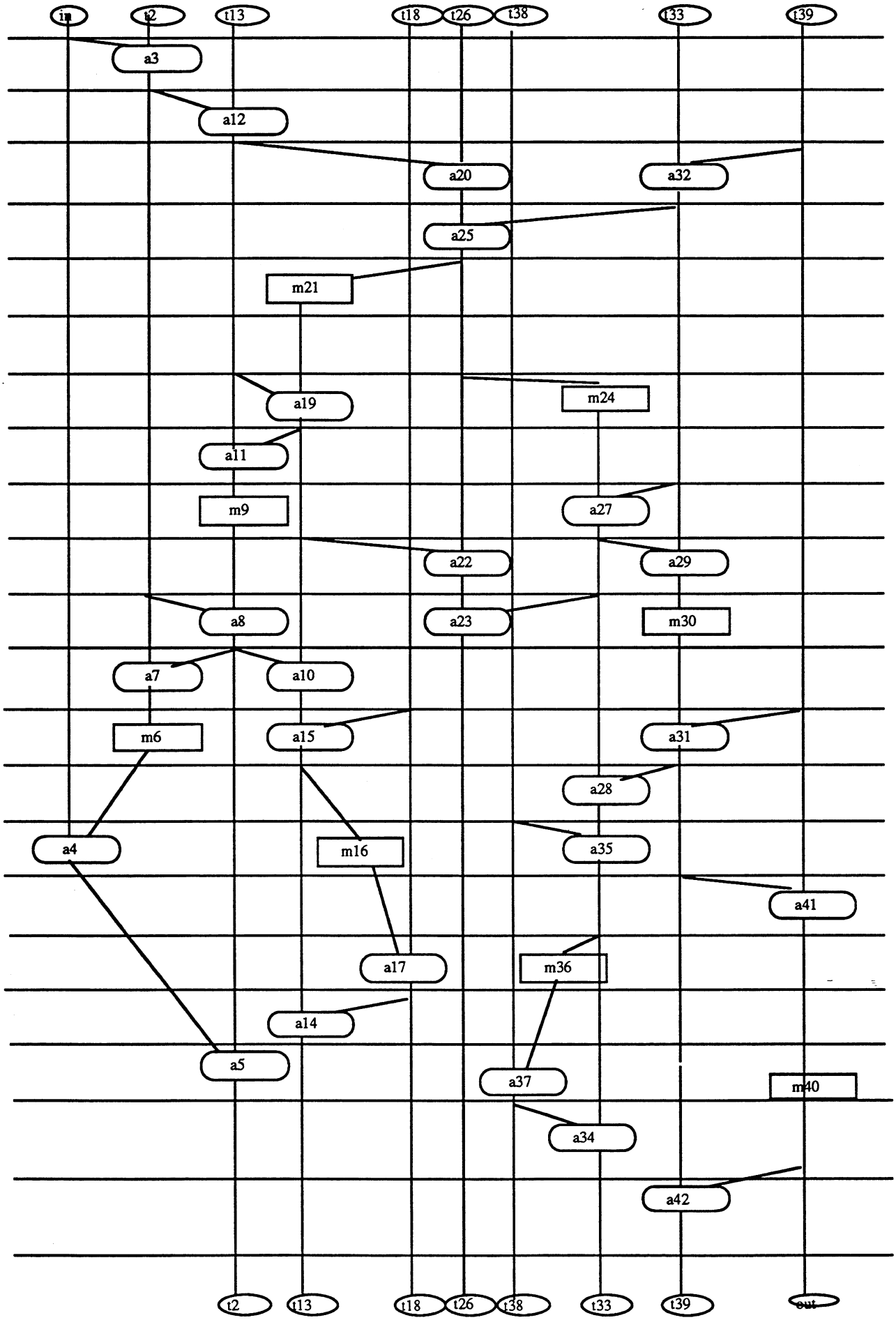


Figure III.1: GDD du filtre du 5ème ordre sur 21 pas de contrôle

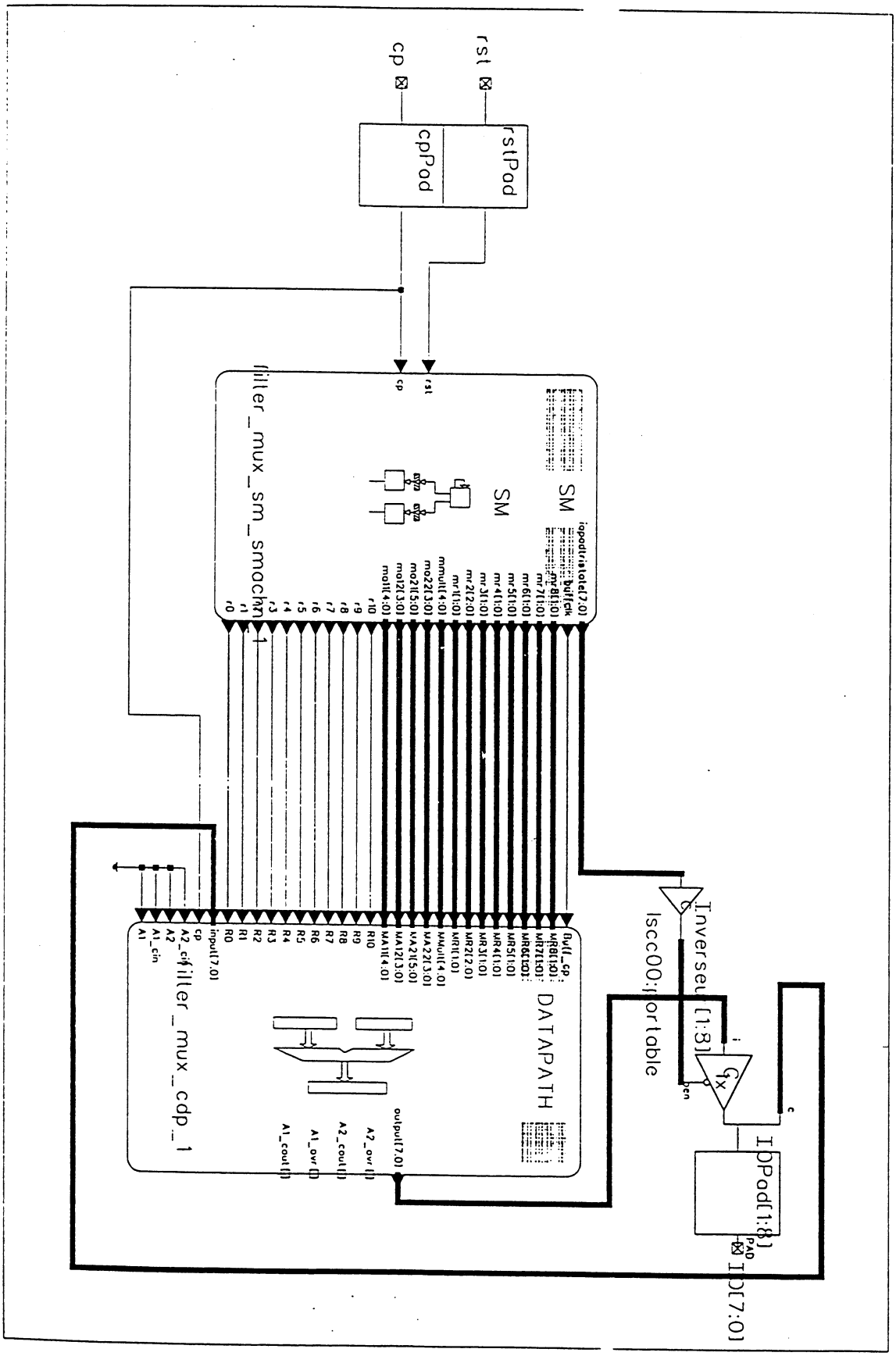
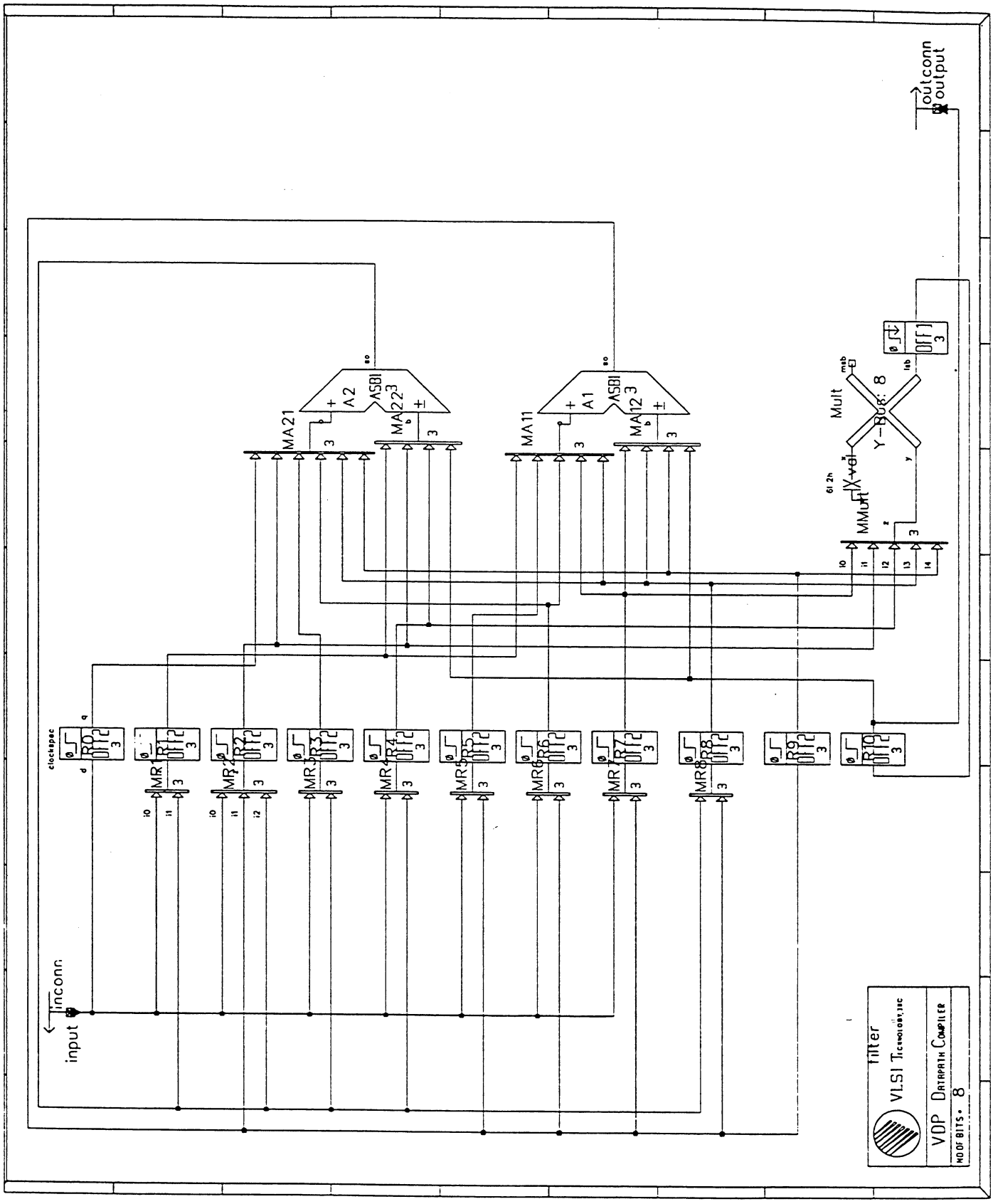


Figure III.2: Schéma PC/PO pour une architecture à base de Multiplexeurs



[la]filter_mux by anne on 10-Jun-92 at 10:27 A.M.

Figure III.3: Schéma de la PO pour une architecture à base de Multiplexeurs

```
sm filter_mux_sm;

clock cp;
reset rst --> Init0;

latched outputs MR8[1:0] MR7[1:0] MR6[1:0]
MR5[1:0] MR4[1:0] MR3[1:0]
MR2[2:0] MR1[1:0] MA11[4:0] MA12[3:0]
MA21[5:0] MA22[3:0] MMult[4:0]
R10 R9 R8 R7 R6 R5 R4 R3 R2 R1 R0 BuffClk
IOPadTriState[7:0];

state Init0
      --> Init1 R0;
state Init1
      --> Init2 R1 MR1[1:0]=01;
state Init2
      --> Init3 R2 MR2[2:0]=001;
state Init3
      --> Init4 R3 MR3[1:0]=01;
state Init4
      --> Init5 R4 MR4[1:0]=01;
state Init5
      --> Init6 R5 MR5[1:0]=01;
state Init6
      --> Init7 R6 MR6[1:0]=01;
state Init7
      --> etat1 R7 MR7[1:0]=01;
state etat1
      --> etat2 R1 MR1[1:0]=10
MA21[5:0]=000001  MA22[3:0]=0001;
state etat2
      --> etat3 R2 MR2[2:0]=100
MA21[5:0]=000010  MA22[3:0]=0001;
state etat3
      --> etat4 R4 MR4[1:0]=10
MA21[5:0]=000010  MA22[3:0]=0100
            R6 MR6[1:0]=10 MA11[4:0]=00100
MA12[3:0]=0001;
state etat4
      --> etat5 R4 MR4[1:0]=10
MA21[5:0]=001000  MA22[3:0]=0100;
state etat5
      --> etat6 BuffClk MMult[4:0]=00100;
state etat6
      --> etat7 R10;
state etat7
      --> etat8 R8 MR8[1:0]=01
MA21[5:0]=000010  MA22[3:0]=1000
            BuffClk MMult[4:0]=00100;
state etat8
      --> etat9 R2 MR2[2:0]=100
MA21[5:0]=000010  MA22[3:0]=1000
            R10;
state etat9
      --> etat10 BuffClk MMult[4:0]=00010
            R9 MA12[3:0]=1000
MA11[4:0]=00100;
state etat10
      --> etat11 R4 MA21[5:0]=010000
MA22[3:0]=0100  MR4[1:0]=10
            R2 MA11[4:0]=00100
MA12[3:0]=0100  MR2[2:0]=010
            R10;
state etat11
      --> etat12 R6 MR6[1:0]=10
MA11[4:0]=00001  MA12[3:0]=1000
            R4 MR4[1:0]=10
MA21[5:0]=100000  MA22[3:0]=0100
            BuffClk MMult[4:0]=00010;
state etat12
      --> etat13 R8 MR8[1:0]=01
MA21[5:0]=001000  MA22[3:0]=0001
            R2 MR2[2:0]=010
MA11[4:0]=00100  MA12[3:0]= 0010
            R10;
state etat13
      --> etat14 BuffClk MMult[4:0]=01000
            R2 MR2[2:0]=100
MA21[5:0]=000100  MA22[3:0]=0010
            R8 MR8[1:0]=10 MA11[4:0]=01000
MA12[3:0]=1000;
state etat14
      --> etat15 R10
            R9  MA11[4:0]=10000
MA12[3:0]=0100;
state etat15
      --> etat16 R1 MR1[1:0]=10
MA21[5:0]=000001  MA22[3:0]=1000
            BuffClk MMult[4:0]=00010
            R9 MA11[4:0]=00010
MA12[3:0]=0100;
state etat16
      --> etat17 R7 MR7[1:0]=10
MA11[4:0]=10000  MA12[3:0]=0001
            R10;
state etat17
      --> etat18 R3 MR3[1:0]=10
MA21[5:0]=000100  MA22[3:0]=1000
            BuffClk MMult[4:0]=10000;
state etat18
      --> etat19 R2 MR2[2:0]=100
MA21[5:0]=000100  MA22[3:0]=0010
            R10;
state etat19
      --> etat20 R1 MR1[1:0]=10
MA21[5:0]=001000  MA22[3:0]=0001
            R5 MR5[1:0]=10 MA11[4:0]=00010
MA12[3:0]=1000
            BuffClk MMult[4:0]=00001;
state etat20
      --> etat21 R6 MR6[1:0]=10
MA11[4:0]=00010  MA12[3:0]=0100
            R10;
state etat21
      --> Out R7 MR7[1:0]=10 MA11[4:0]=10000
MA12[3:0]=1000;
state out
      --> Init0 IOPadTriState[7:0]=11111111;
end
```

Figure III.4: PC pour une architecture à base de Multiplexeurs

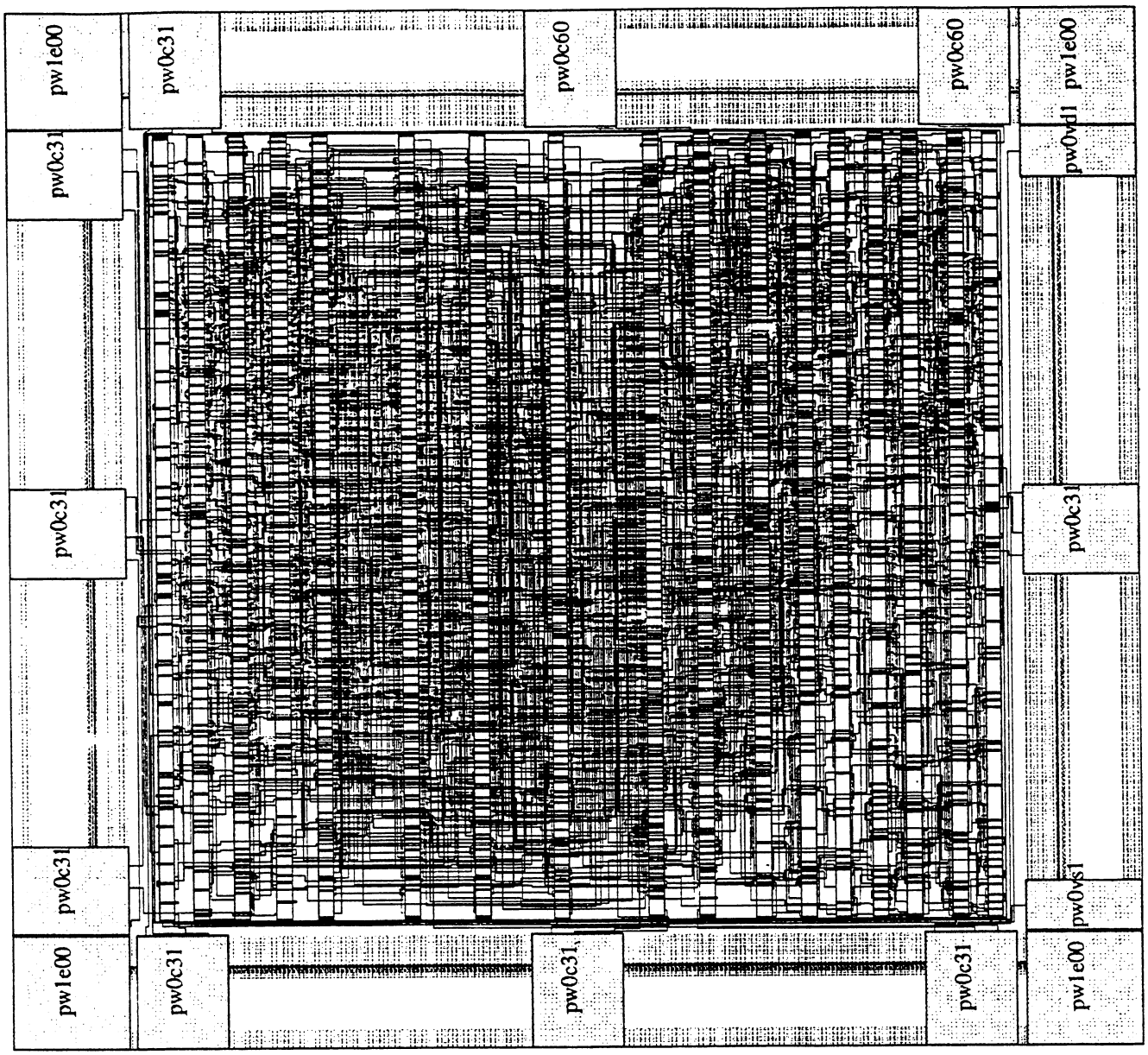
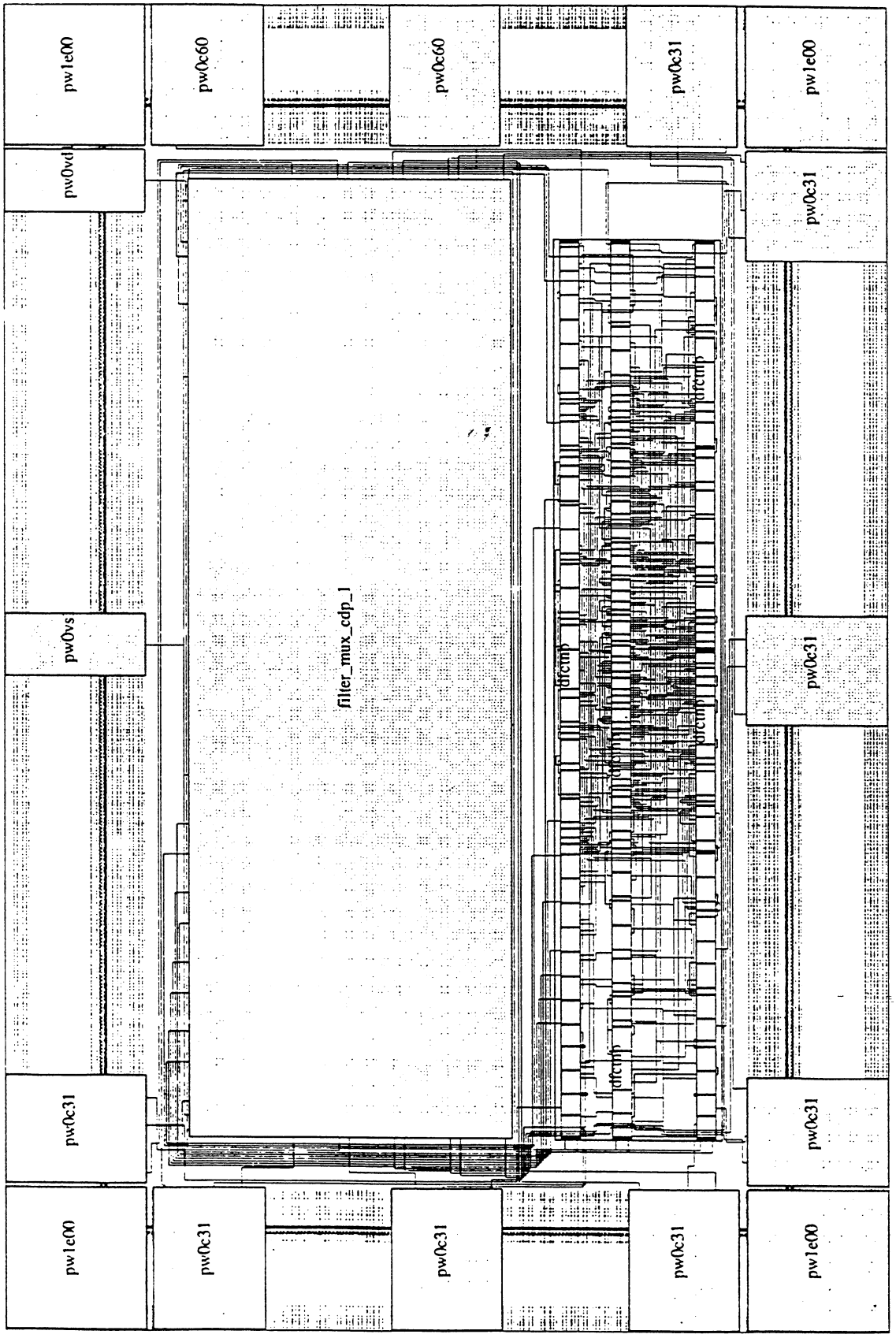


Figure III.5: Layout en cellules standards pour une architecture à base de Multiplexeurs

```
# General Information:
system_name            SPARC UNIX
user_Id                anne
home_directory         /users/csi/anne
mainsail_version       12.15
vlsi_tools_version     v8r3.1
tool                   Chip Compiler
technology             cmn16
last_operation         RouteBlock
design                 global_mux_f


# Nets And Cells:
nets                   1251
arbitrary_blocks       0
cell_areas             1
standard_cells         1233
pads                   12
pins_per_net           3.13
nets_with_weight       138
nets_with_maxcap       0
net_with_max_pins      SM.i_184.Z 34
clock_net              no clock nets


# Total Area (excluding scribe lines) :
package                global_mux_default
chip_size              4604.00 X 4573.00 lambda (145.01 X 144.03 mil)
core_size              3514.00 X 3414.50 lambda (110.68 X 107.54 mil)
transistors            12440
core_density           .1037E-02 trans/sq-lam (.2613 gates/sq-mil)


# Routing Factor:
block_routing_factor   .75
core_routing_factor    3.57
padring_routing_factor .58


# Post Route:
CM2CM2                 6412
metal                  693688 lambda
metal2                 525779 lambda
unroutes_in_cell_routing  0
unroutes_in_block_routing 0


# Cell Areas:
Area global_mux_sc
 size                  3514 X 3414.5 lambda
 routing_factor        1.72
 transistors           12440
 density               .1037E-02 trans/sq-lam (.2613 gates/sq-mil)
 standard_cells        1242
 row_end_cells         34
 feed_thru_cells       68
end_area
```

Figure III.6: Caractéristiques du Layout de la figure III.5



Compass Design Automation VGS plot global_mux by anne on 5-Nov-92 at 5:03 A.M.

Figure III.7: Layout en tranches de bit pour une architecture à base de Multiplexeurs

```
# General Information:
system_name          SPARC UNIX
user_Id              anne
home_directory       /users/csi/anne
mainsail_version     12.15
vlsi_tools_version   v8r3.1
tool                 Chip Compiler
technology           cmn16
last_operation       RouteBlock
design               global_mux_f


# Nets And Cells:
nets                 214
arbitrary_blocks     1
cell_areas           2
standard_cells       192
pads                 12
pins_per_net         3.33
nets_with_weight     0
nets_with_maxcap     0
net_with_max_pins    SM.i_184.Z 34
clock_net            no clock nets


# Total Area (excluding scribe lines) :
package              global_mux_default
chip_size            6546.00 X 3845.00 lambda (206.17 X 121.10 mil)
core_size            5240.00 X 2393.50 lambda (165.04 X 75.39 mil)
transistors          1884
core_density         .1502E-03 trans/sq-lam (.0379 gates/sq-mil)


# Routing Factor:
block_routing_factor     1.46
core_routing_factor      1.82
padring_routing_factor   .26


# Post Route:
CM2CM2               1115
metal                159630 lambda
metal2               367897 lambda
unroutes_in_cell_routing   0
unroutes_in_block_routing  0


# Cell Areas:
Area global_mux_sc
 size                4322 X 442 lambda
 routing_factor      1.99
 transistors         1602
 density             .8386E-03 trans/sq-lam (.2113 gates/sq-mil)
 standard_cells      175
 row_end_cells       4
 feed_thru_cells     7
end_area

Area global_mux_sc1
 size                113 X 1410 lambda
 routing_factor      .54
 transistors         282
 density             .1770E-02 trans/sq-lam (.4460 gates/sq-mil)
 standard_cells      17
 row_end_cells       2
end_area
```

Figure III.8: Caractéristiques du layout de la figure III.7

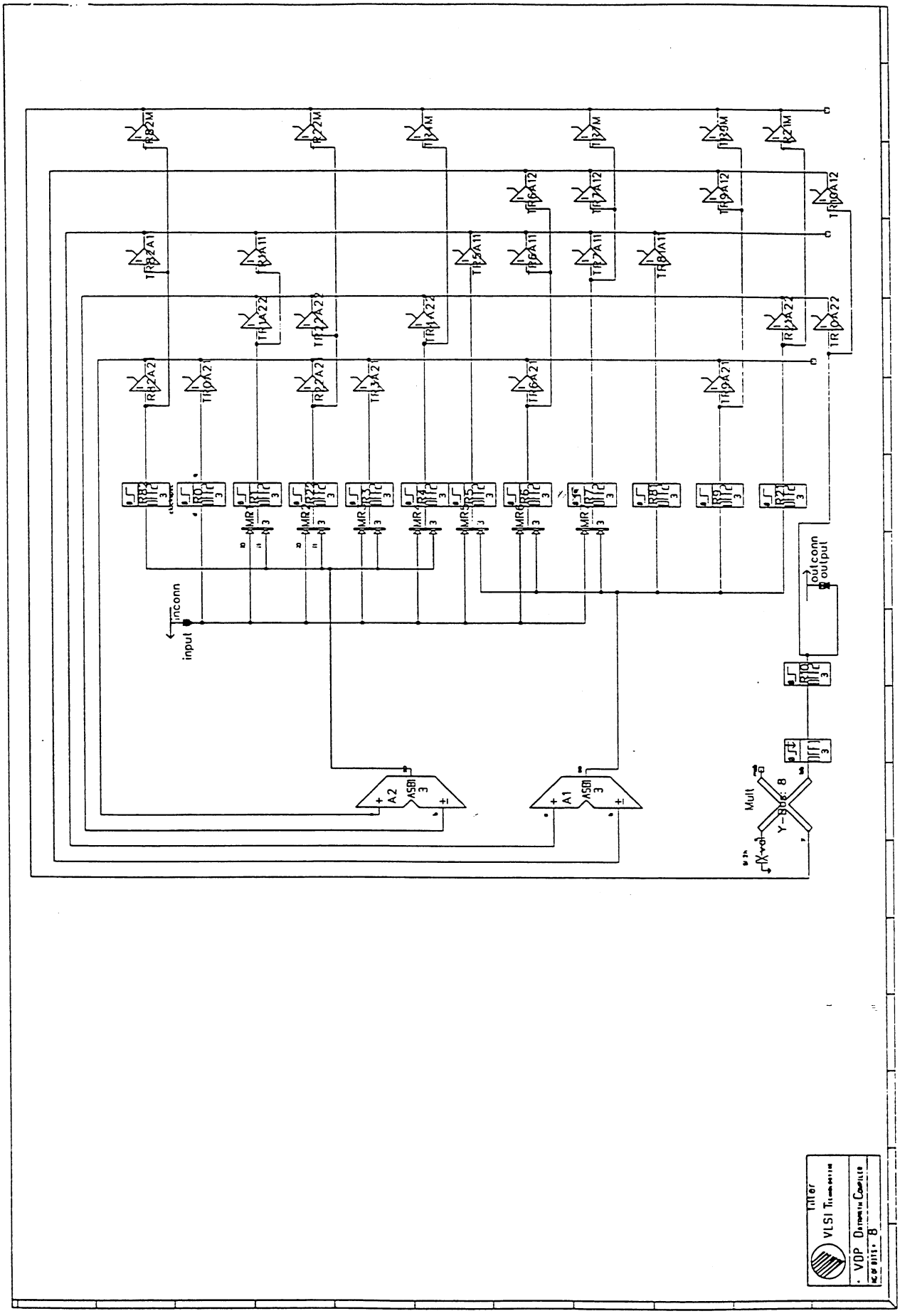


Figure III.9: Schéma PC/PO pour une architecture à base de bus et registres dédiés en sortie

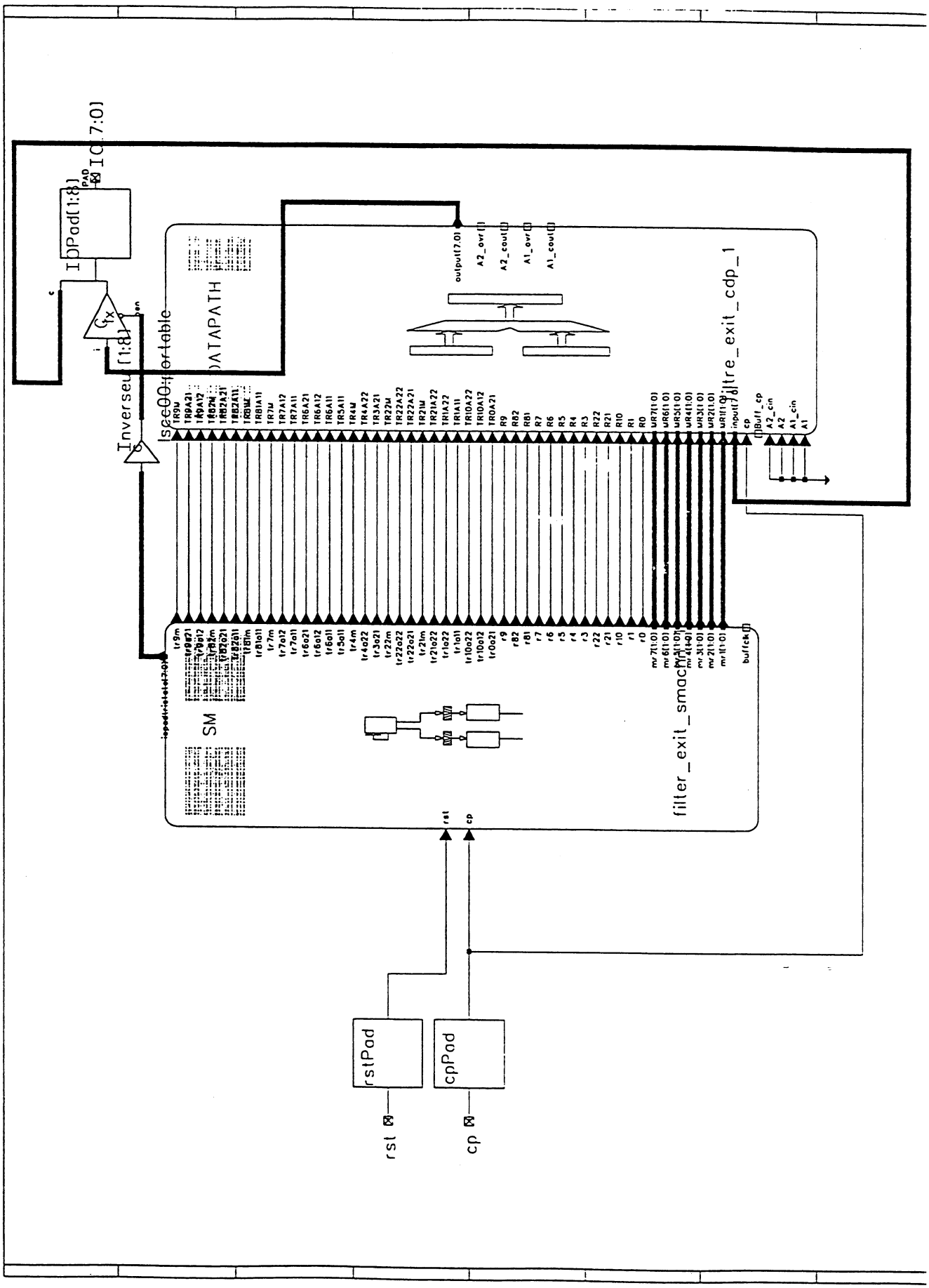


Figure III.10: Schéma de la PO pour une architecture à base de bus et registres dédiés en sortie

```
sm filter_mux_sm;

clock cp;
reset rst --> Init0;

latched outputs MR7[1:0]=10 MR6[1:0]=10 MR5[1:0]=10
MR4[1:0]=10 MR3[1:0]=10 MR2[1:0]=10 MR1[1:0]=10
TR21M TR22M TR4M TR7M TR81M TR82M TR9M
TR82A21 TR0A21 TR22A21 TR3A21 TR6A21 TR9A21
TR1A22 TR22A22 TR4A22 TR21A22 TR10A22
TR82A11 TR1A11 TR5A11 TR6A11 TR7A11 TR81A11
TR6A12 TR7A12 TR9A12 TR10A12
R10 R9 R81 R82 R7 R6 R5 R4 R3 R21 R22 R1 R0
BuffClk IOPadTriState[7:0];

state Init0
    --> Init1 R0;
state Init1
    --> Init2 R1 MR1[1:0]=01;
state Init2
    --> Init3 R22 MR2[1:0]=01;
state Init3
    --> Init4 R3 MR3[1:0]=01;
state Init4
    --> Init5 R4 MR4[1:0]=01;
state Init5
    --> Init6 R5 MR5[1:0]=01;
state Init6
    --> Init7 R6 MR6[1:0]=01;
state Init7
    --> etat1 R7 MR7[1:0]=01;
state etat1
    --> etat2 R1 TR0A21 TR1A22;
state etat2
    --> etat3 R22 TR1A22 TR22A21;
state etat3
    --> etat4 R4 TR4A22 TR6A21
        R6 TR6A11 TR7A12;
state etat4
    --> etat5 R4 TR4A22 TR6A21;
state etat5
    --> etat6 BuffClk TR4M=1;
state etat6
    --> etat7 R10;
state etat7
    --> etat8 R82  TR10A22 TR22A21
        BuffClk TR4M=1;
state etat8
    --> etat9 R22 TR82A21 TR22A22
        R10;
state etat9
    --> etat10 BuffClk TR22M
        R9 TR10A12 TR6A11;
state etat10
    --> etat11 R4 TR82A21 TR4A22
        R21 TR9A12 TR6A11
        R10;
state etat11
    --> etat12 R6 TR1A11 TR10A12
        R4 TR4A22 TR9A21
        BuffClk TR21M;
state etat12
    --> etat13 R82 TR1A22 TR6A21
        R21 TR6A12 TR82A11
        R10;
state etat13
    --> etat14 BuffClk TR82M
        R22 TR21A22 TR3A21
        R81 TR10A12 TR7A11;
state etat14
    --> etat15 R10
        R9 TR9A12 TR81A11;
state etat15
    --> etat16 R1 TR0A21 TR10A22
        BuffClk  TR22M=1
        R9 TR5A11 TR9A12;
state etat16
    --> etat17 R7 TR81A11 TR7A12
        R10;
state etat17
    --> etat18 R3 TR10A22 TR3A21
        BuffClk TR9M=1;
state etat18
    --> etat19 R22 TR22A22 TR3A21
        R10;
state etat19
    --> etat20 R1 TR6A21 TR1A22
        R5 TR5A11 TR10A12
        BuffClk TR7M=1;
state etat20
    --> etat21 R6 TR5A11 TR9A12
        R10;
state etat21
    --> Out R7 TR81A11 TR10A12;
state out
    --> Init0 IOPadTriState[7:0]=11111111;
end
```

Figure III.11: PC pour une architecture à base de bus et registres dédiés en sortie

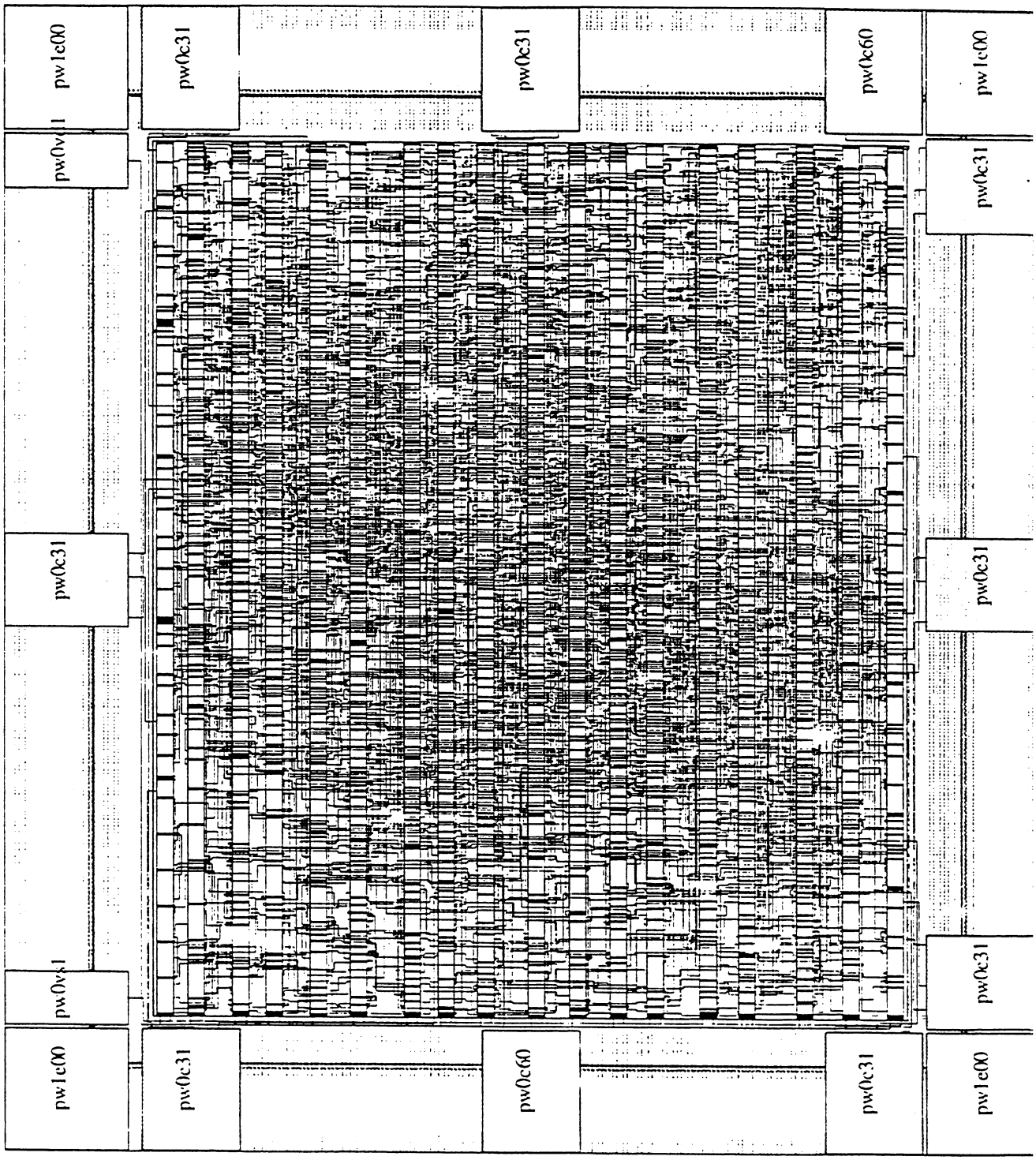
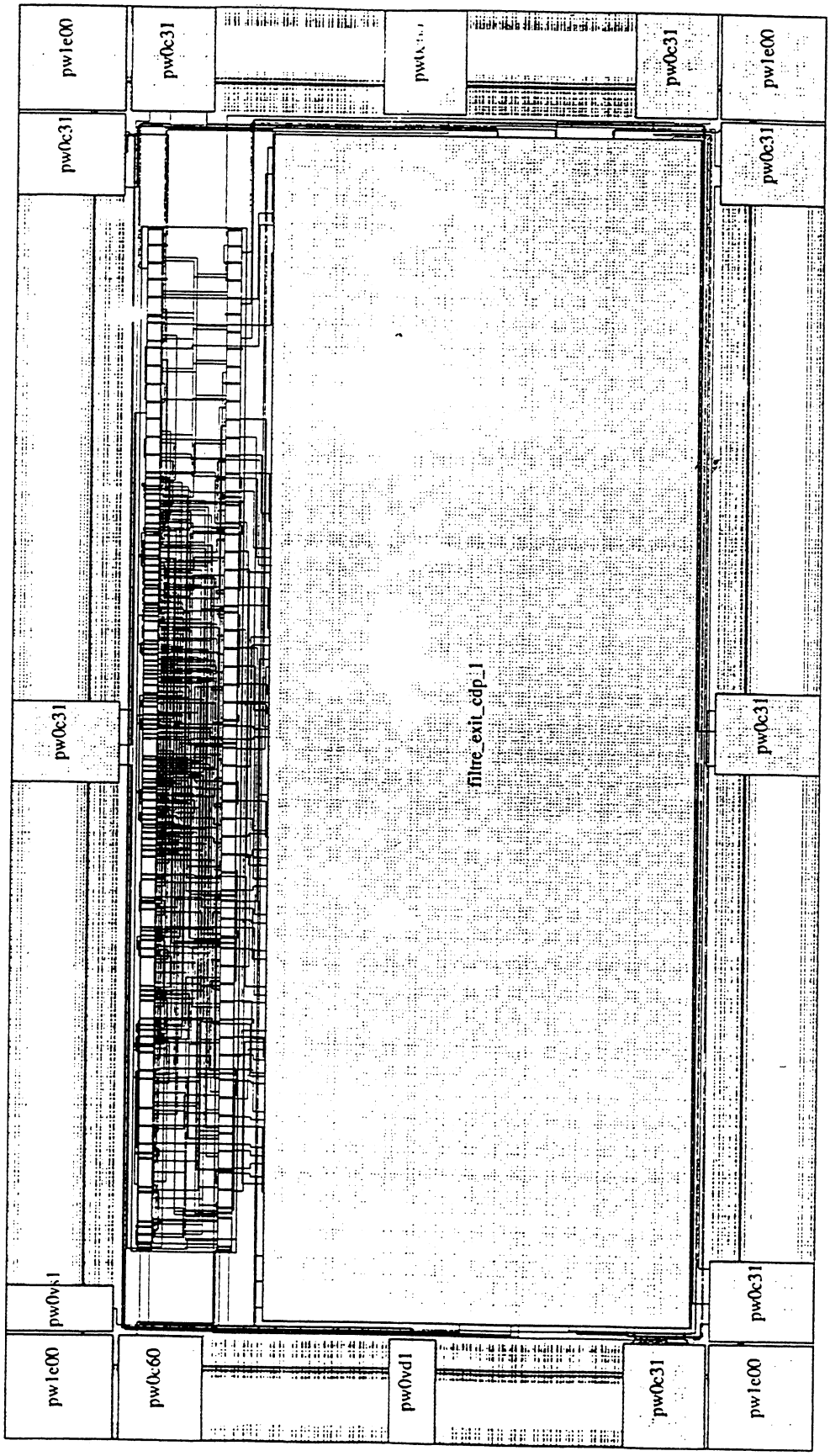


Figure III.12: Layout en cellules standards pour une architecture à base de bus et registres dédiés en sortie

```
# General Information:
system_name            SPARC UNIX
user_Id                anne
home_directory         /users/csi/anne
mainsail_version       12.15
vlsi_tools_version     v8r3.1
tool                   Chip Compiler
technology             cmn16
last_operation         RouteBlock
design                 global_exit_f


# Nets And Cells:
nets                   1163
arbitrary_blocks       0
cell_areas             1
standard_cells         1329
pads                   12
pins_per_net           3.43
nets_with_weight       138
nets_with_maxcap       0
net_with_max_pins      U5.i_178.Z 36
clock_net              no clock nets


# Total Area (excluding scribe lines) :
package                global_exit_default
chip_size              4802.00 X 4288.00 lambda (151.24 X 135.06 mil)
core_size              3682.00 X 3092.00 lambda (115.97 X 97.39 mil)
transistors            13854
core_density           .1217E-02 trans/sq-lam (.3067 gates/sq-mil)


# Routing Factor:
block_routing_factor   .81
core_routing_factor    3.03
padring_routing_factor .44


# Post Route:
CM2CM2                 6317
metal                  448401 lambda
metal2                 500111 lambda
unroutes_in_cell_routing  0
unroutes_in_block_routing 0


# Cell Areas:
Area global_exit_sc
  size                 3682 X 3092 lambda
  routing_factor       1.33
  transistors          13854
  density              .1217E-02 trans/sq-lam (.3067 gates/sq-mil)
  standard_cells       1344
  row_end_cells        36
  feed_thru_cells      86
end_area
```

Figure III.13: Caractéristiques du layout de la figure III.12

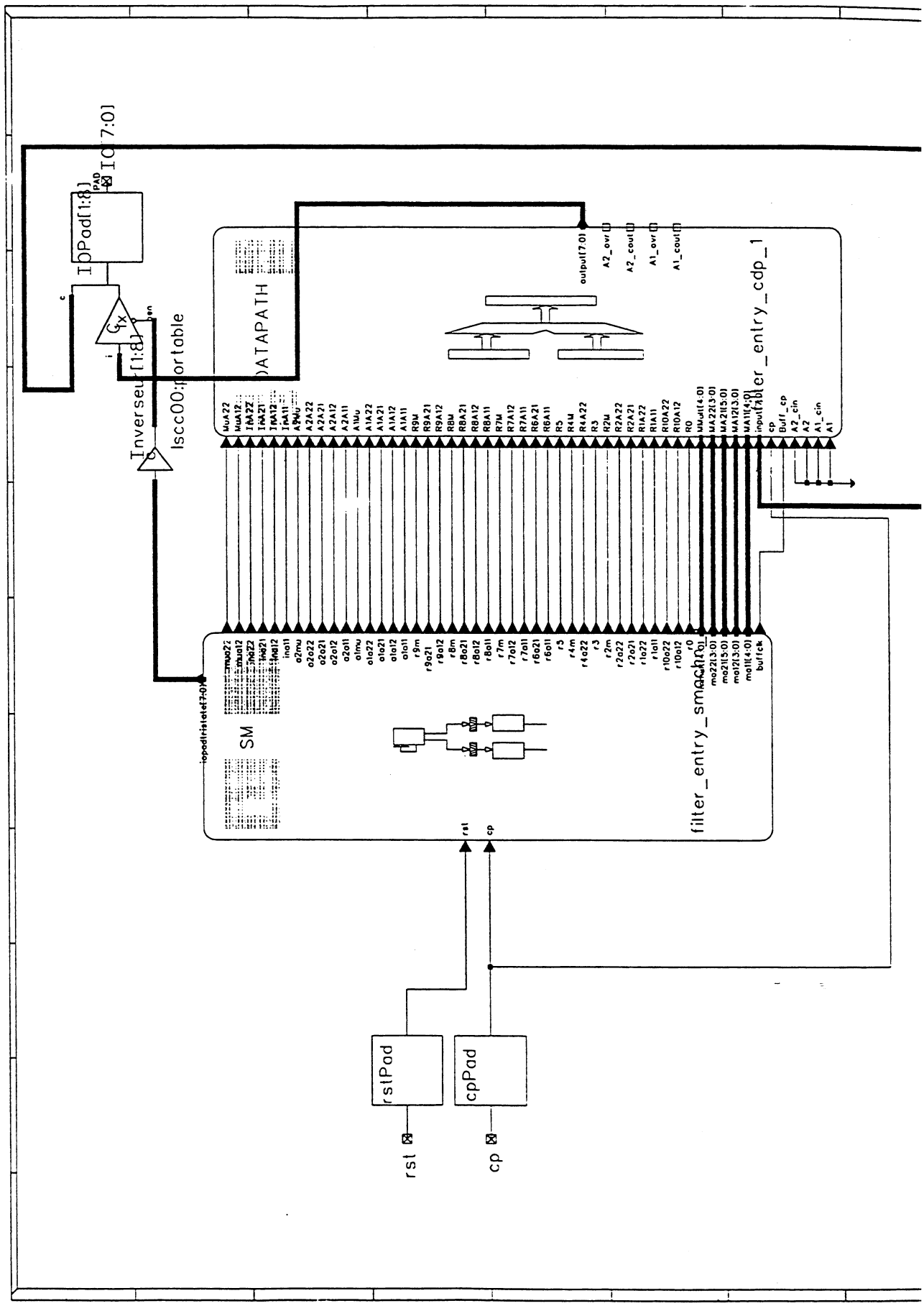


Compass Design Automation VGS plot global_exit by anne on 8-Nov-92 at 5:42 A.M.

Figure III.14: Layout en tranches de bit pour une architecture à base de bus et registres dédiés en sortie

```
# General Information:
system_name          SPARC UNIX
user_Id              anne
home_directory       /users/csi/anne
mainsail_version     12.15
vlsi_tools_version   v8r3.1
tool                 Chip Compiler
technology           cmn16
last_operation       RouteBlock
design               global_exit_f


# Nets And Cells:
nets                 207
arbitrary_blocks     1
cell_areas           1
standard_cells       185
pads                 12
pins_per_net         3.34
nets_with_weight     0
nets_with_maxcap     0
net_with_max_pins    U5.i_178.Z 36
clock_net            no clock nets


# Total Area (excluding scribe lines) :
package              global_exit_default
chip_size            8802.50 X 3822.00 lambda (277.24 X 120.38 mil)
core_size            7610.50 X 2080.00 lambda (239.70 X 65.51 mil)
transistors          1896
core_density         .1198E-03 trans/sq-lam (.0302 gates/sq-mil)


# Routing Factor:
block_routing_factor    1.43
core_routing_factor     1.60
padring_routing_factor  .33


# Post Route:
CM2CM2               1060
metal                153528 lambda
metal2               325393 lambda
unroutes_in_cell_routing  0
unroutes_in_block_routing 0


# Cell Areas:
Area global_exit_sc
  size               1482 X 1122 lambda
  routing_factor     1.18
  transistors        1896
  density            .1140E-02 trans/sq-lam (.2874 gates/sq-mil)
  standard_cells     186
  row_end_cells      14
  feed_thru_cells    31
end_area
```

Figure III.15: Caractéristiques du layout de la figure III.14



[la]global_entry by anne on 12-Jun-92 at 8:47 A.M.

Figure III.16: Schéma PC/PO pour une architecture à base de bus et registres dédiés en entrée

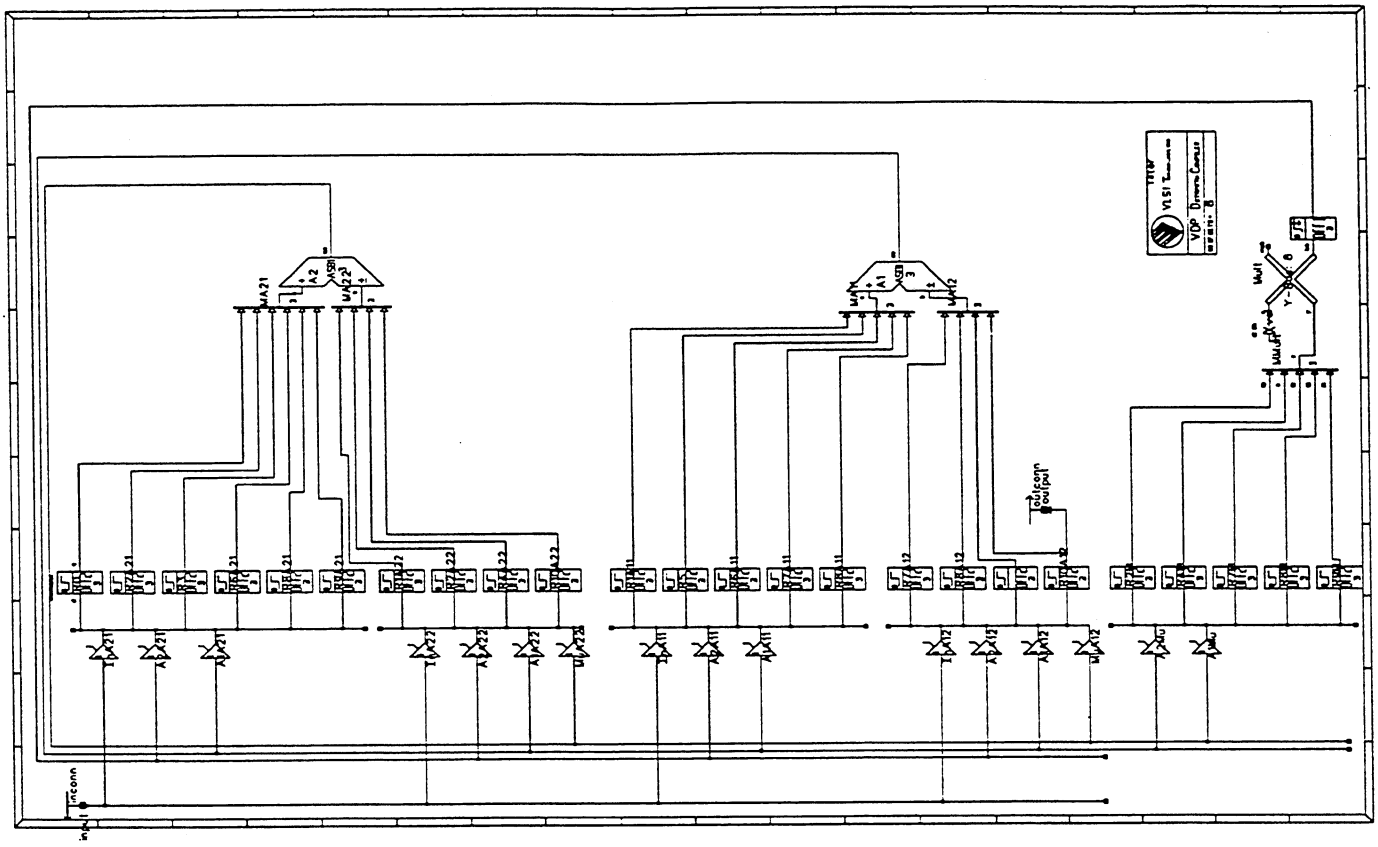


Figure III.17: Schéma de la PO pour une architecture à base de bus et registres dédiés en entrée

```
sm filter_mux_sm;

clock cp;
reset rst --> Init0;

latched outputs MA11[4:0] MA12[3:0] MA21[5:0]
MA22[3:0] MMult[4:0]
R10A12 R10A22 R9M R9A21 R9A12 R8M R8A21
R8A12 R8A11 R7M R7A12 R7A11 R6A21 R6A11
R5 R4M R4A22 R3 R2M R2A22 R2A21 R1A22 R1A11
R0 BuffClk IOPadTriState[7:0]
A2A22 A2A21 A2A11 A2A12 A2Mu A1A22 A1A21
A1A12 A1A11 A1Mu MuA12 MuA22 InA22
InA21 InA12 InA11;

state Init0
      --> Init1 R0 InA21;
state Init1
      --> Init2 R1A22 InA22;
state Init2
      --> Init3 R2A22 InA22;
state Init3
      --> Init4 R3 InA21;
state Init4
      --> Init5 R4A22 InA22;
state Init5
      --> Init6 R5 InA11;
state Init6
      --> Init7 R6A11 InA11;
state Init7
      --> etat1 R7A12 R7A11 InA12 InA11;
state etat1
      --> etat2 R1A22 R1A11 A2A22 A2A11
MA21[5:0]=000001 MA22[3:0]=0001;
state etat2
      --> etat3 R2A22 A2A22 R2A21 A2A21
MA21[5:0]=000010 MA22[3:0]=0001;
state etat3
      --> etat4 R4A22 A2A22 MA21[5:0]=000010
MA22[3:0]=0100
            R6A21 R6A11 A1A21 A1A11
MA11[4:0]=00100 MA12[3:0]=0001;
state etat4
      --> etat5 R4M A2Mu R4A22 A2A22
MA21[5:0]=001000 MA22[3:0]=0100;
state etat5
      --> etat6 BuffClk MMult[4:0]=00100;
state etat6
      --> etat7 R10A12 MuA12;
state etat7
      --> etat8 R8A21 A2A21 R8A12 A2A12
MA21[5:0]=000010 MA22[3:0]=1000
            BuffClk MMult[4:0]=00100;
state etat8
      --> etat9 R2M A2Mu MA21[5:0]=000010
MA22[3:0]=1000
            R10A12 MuA12;
state etat9
      --> etat10 BuffClk MMult[4:0]=00010
            R9A12 A1A12 R9A21 A1A21
MA12[3:0]=1000 MA11[4:0]=00100;
state etat10
      --> etat11 R4A22 A2A22 MA21[5:0]=010000
MA22[3:0]=0100
            R2M A1Mu MA11[4:0]=00100
MA12[3:0]=0100
            R10A12 MuA12;
state etat11
      --> etat12 R6A21 A1A21 R6A11 A1A11
MA11[4:0]=00001 MA12[3:0]=1000
            R4A22 A2A22 MA21[5:0]=100000
MA22[3:0]=0100
            BuffClk MMult[4:0]=00010;
state etat12
      --> etat13 R8M A2Mu MA21[5:0]=001000
MA22[3:0]=0001
            R2A22 A1A22 MA11[4:0]=00100
MA12[3:0]= 0010
            R10A12 MuA12;
state etat13
      --> etat14 BuffClk MMult[4:0]=01000
            R2A22 A2A22 R2M A2Mu
MA21[5:0]=000100 MA22[3:0]=0010
            R8A11 A1A11 MA11[4:0]=01000
MA12[3:0]=1000;
state etat14
      --> etat15 R10A22 MuA22
            R9A12 A1A12 MA11[4:0]=10000
MA12[3:0]=0100;
state etat15
      --> etat16 R1A22 A2A22 MA21[5:0]=000001
MA22[3:0]=1000
            BuffClk MMult[4:0]=00010
            R9M A1Mu R9A12 A1A12
MA11[4:0]=00010 MA12[3:0]=0100;
state etat16
      --> etat17 R7M A1Mu MA11[4:0]=10000
MA12[3:0]=0001
            R10A22 MuA22;
state etat17
      --> etat18 R3 A2A21 MA21[5:0]=000100
MA22[3:0]=1000
            BuffClk MMult[4:0]=10000;
state etat18
      --> etat19 R2A21 A2A21 MA21[5:0]=000100
MA22[3:0]=0010
            R10A12 MuA12;
state etat19
      --> etat20 R1A22 A2A22 MA21[5:0]=001000
MA22[3:0]=0001
            R5 A1A11 MA11[4:0]=00010
MA12[3:0]=1000
            BuffClk MMult[4:0]=00001;
state etat20
      --> etat21 R6A11 A1A11 MA11[4:0]=00010
MA12[3:0]=0100
            R10A12 MuA12;
state etat21
      --> Out R7A11 R7A12 A1A11 A1A12
MA11[4:0]=10000 MA12[3:0]=1000;
state out
      --> Init0 IOPadTriState[7:0]=11111111;
end
```

Figure III.18: PC pour une architecture à base de bus et registres dédiés en entrée

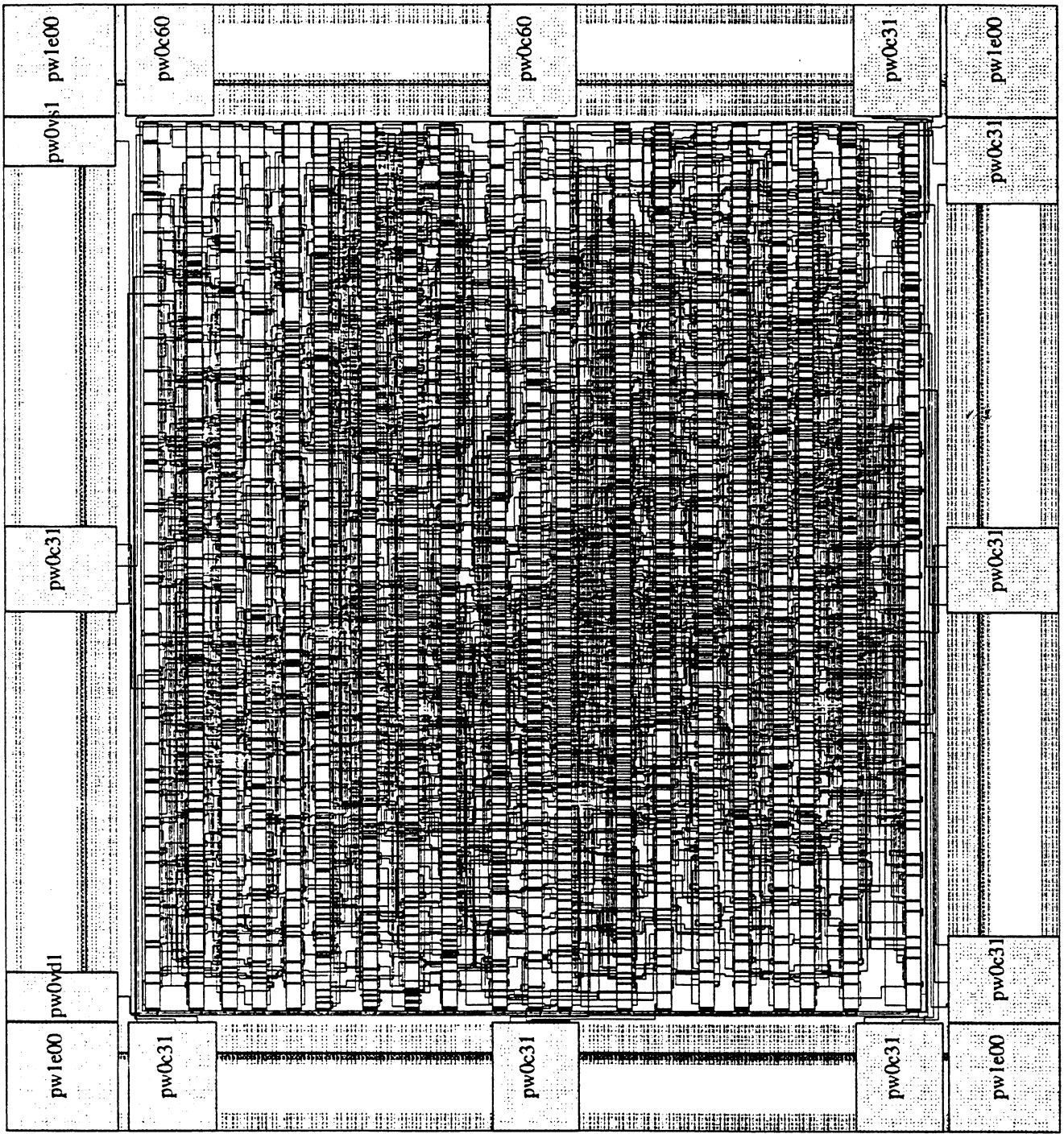
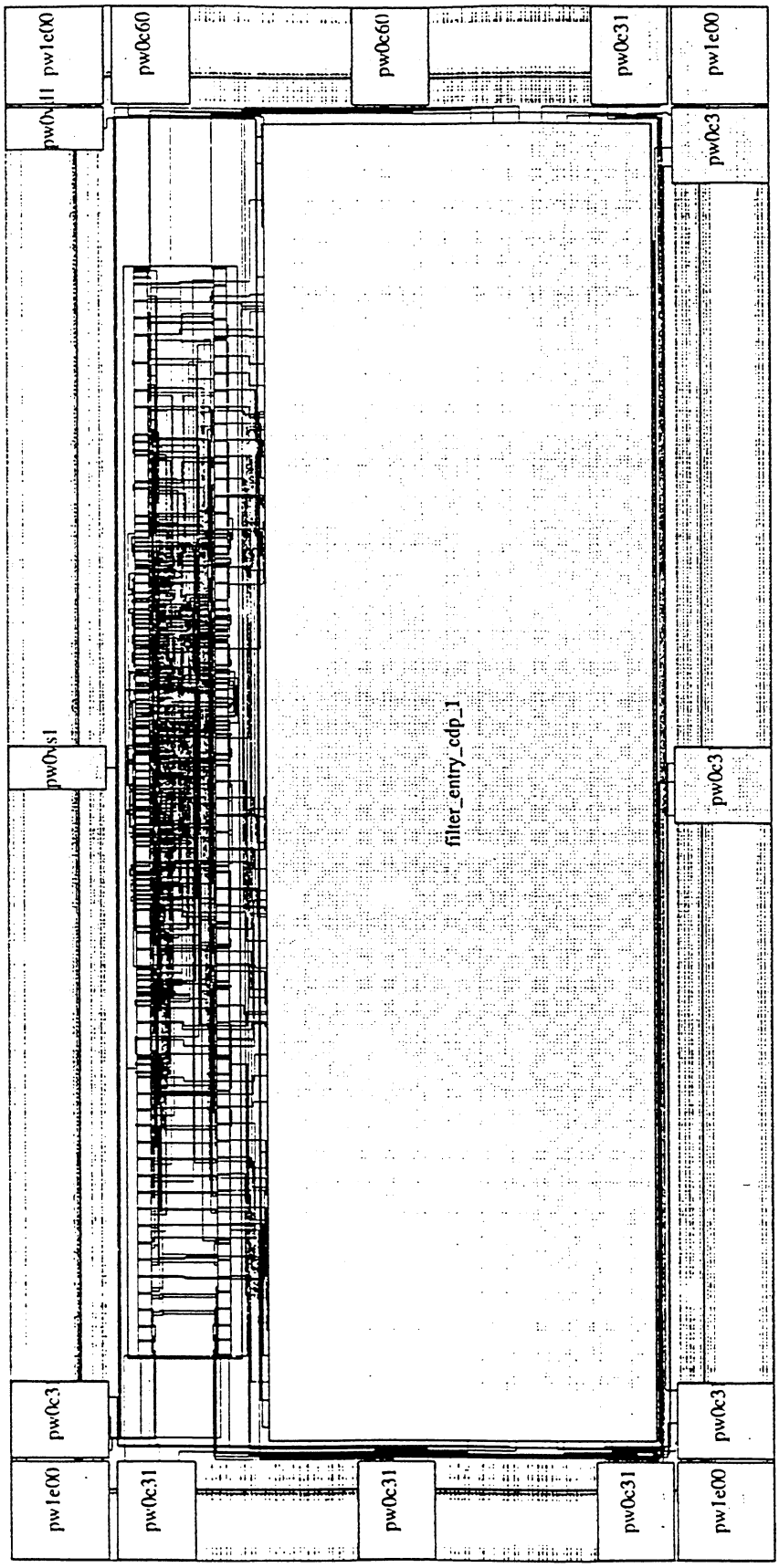


Figure III.19: Layout en cellules standards pour une architecture à base de bus et registres dédiés en entrée

```
# General Information:
system_name            SPARC UNIX
user_Id                anne
home_directory         /users/csi/anne
mainsail_version       12.15
vlsi_tools_version     v8r3.1
tool                   Chip Compiler
technology             cmn16
last_operation         RouteBlock
design                 global_entry_f


# Nets And Cells:
nets                   1468
arbitrary_blocks       0
cell_areas             1
standard_cells         1538
pads                   12
pins_per_net           3.27
nets_with_weight       138
nets_with_maxcap       0
net_with_max_pins      U6.i_212.Z 40
clock_net              no clock nets


# Total Area (excluding scribe lines) :
package                global_entry_default
chip_size              5250.00 X 4556.00 lambda (165.35 X 143.50 mil)
core_size              4122.00 X 3352.00 lambda (129.83 X 105.57 mil)
transistors            17616
core_density           .1275E-02 trans/sq-lam (.3213 gates/sq-mil)


# Routing Factor:
block_routing_factor   .73
core_routing_factor    2.78
padring_routing_factor .53


# Post Route:
CM2CM2                 7222
metal                  561185 lambda
metal2                 548840 lambda
unroutes_in_cell_routing  0
unroutes_in_block_routing 0


# Cell Areas:
Area global_entry_sc
 size                  4122 X 3352 lambda
 routing_factor        1.27
 transistors           17616
 density               .1275E-02 trans/sq-lam (.3213 gates/sq-mil)
 standard_cells        1551
 row_end_cells         40
 feed_thru_cells       91
end_area
```

Figure III.20: Caractéristiques du layout de la figure III.19



Compass Design Automation VGS plot global_entry by anne on 5-Nov-92 at 6:32 A.M.

Figure III.21: Layout en tranches de bit pour une architecture à base de bus et registres dédiés en entrée

```
# General Information:
system_name          SPARC UNIX
user_Id              anne
home_directory       /users/csi/anne
mainsail_version     12.15
vlsi_tools_version   v8r3.5
tool                 Chip Compiler
technology           cmn16
last_operation       RouteBlock
design               global_entry_f


# Nets And Cells:
nets                 241
arbitrary_blocks     1
cell_areas           1
standard_cells       219
pads                 12
pins_per_net         3.38
nets_with_weight     0
nets_with_maxcap     0
net_with_max_pins    U6.i_211.Z 40
clock_net            no clock nets


# Total Area (excluding scribe lines) :
package              global_entry_default
chip_size            10005.00 X 3758.00 lambda (315.12 X 118.36 mil)
core_size            8811.00 X 2005.00 lambda (277.51 X 63.15 mil)
transistors          2176
core_density         .1232E-03 trans/sq-lam (.0310 gates/sq-mil)


# Routing Factor:
block_routing_factor     1.34
core_routing_factor      1.54
padring_routing_factor   .35


# Post Route:
CM2CM2                   1163
metal                    201176 lambda
metal2                   421486 lambda
unroutes_in_cell_routing 0
unroutes_in_block_routing 0


# Cell Areas:
Area global_entry_sc
 size                1474 X 1464 lambda
 routing_factor      1.49
 transistors         2176
 density             .1008E-02 trans/sq-lam (.2541 gates/sq-mil)
 standard_cells      219
 row_end_cells       16
 feed_thru_cells     23
end_area
```

Figure III.22: Caractéristiques du layout de la figure III.21

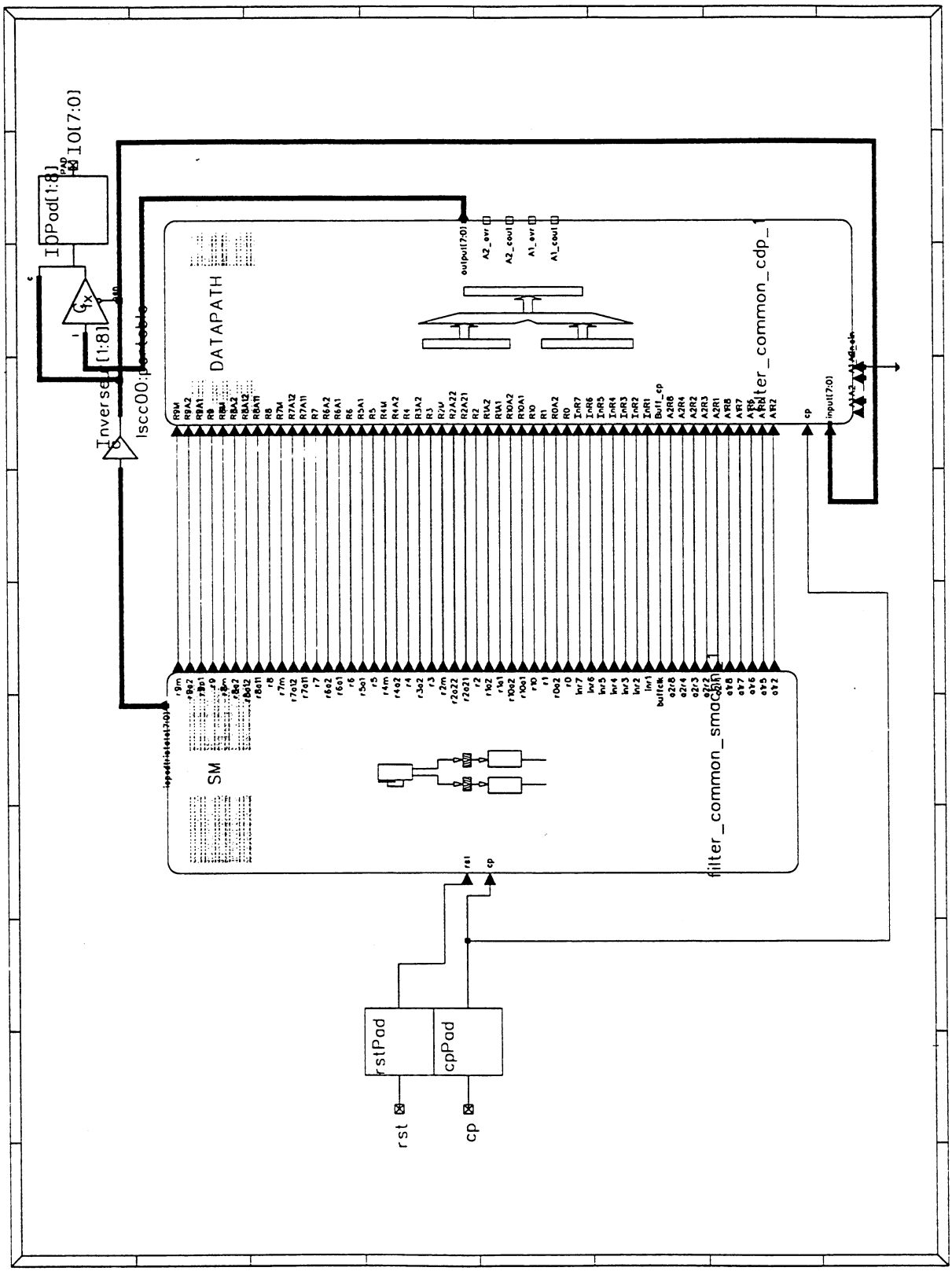


Figure III.23: Schéma PC/PO pour une architecture à base de bus et registres communs

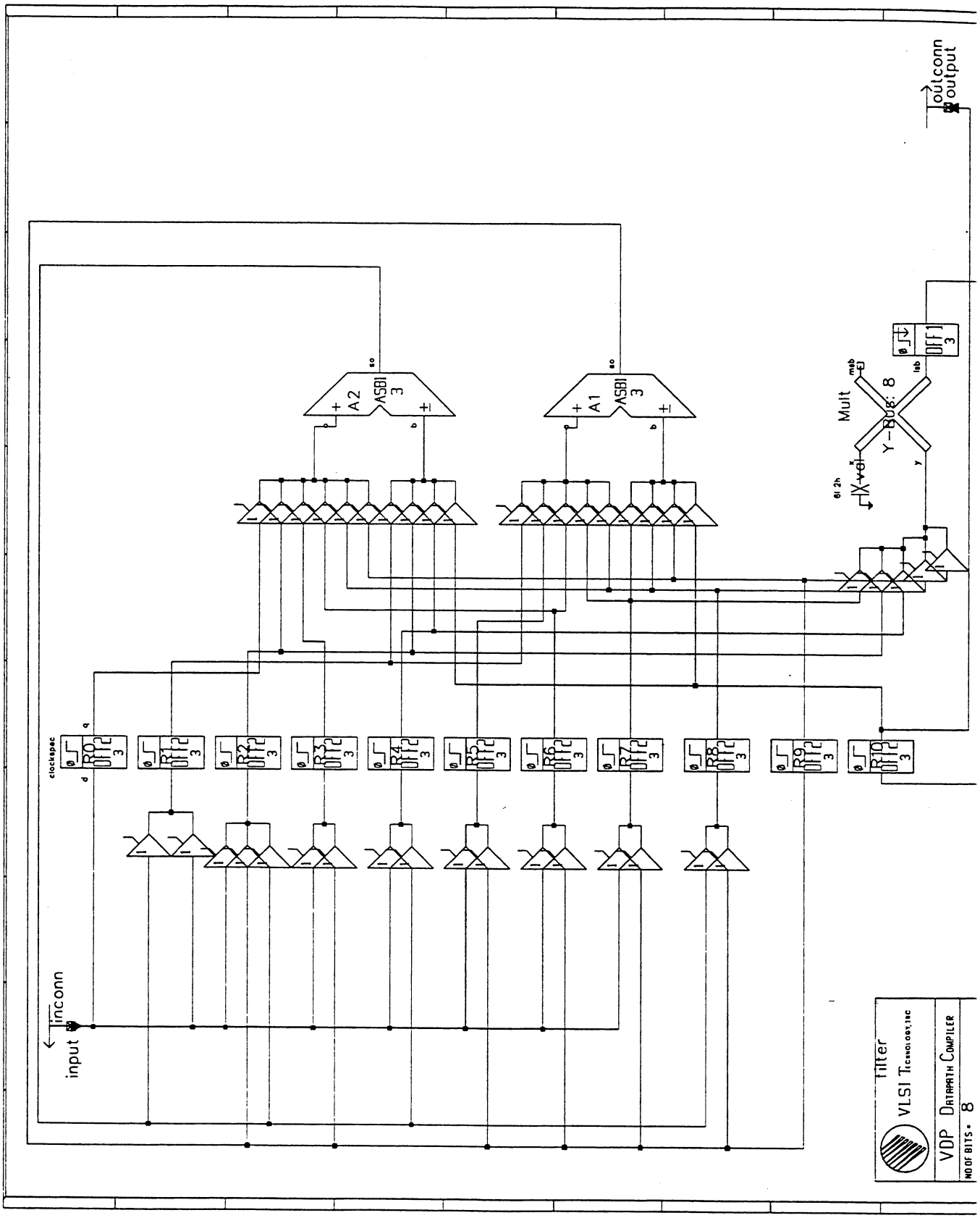


Figure III.24: Schéma de la PO pour une architecture à base de bus et registres communs

```
sm filter_mux_sm;

clock cp;
reset rst --> Init0;

latched outputs InR1 InR2 InR3 InR4 InR5 InR6
InR7 A2R1 A1R2 A2R2 A2R3 A2R4
A1R5 A1R6 A1R7 A1R8 A2R8
R10A1 R10A2 R9M R9A2 R9A1 R8M R8A2
R8A12 R8A11 R7M R7A12 R7A11 R6A2 R6A1
R5A1 R4M R4A2 R3A2 R2M R2A22 R2A21 R1A2
R1A1 R0A2 BuffClk IOPadTriState[7:0]
R0 R1 R2 R3 R4 R5 R6 R7 R8 R9 R10;

state Init0
    --> Init1 R0;
state Init1
    --> Init2 R1 InR1;
state Init2
    --> Init3 R2 InR2;
state Init3
    --> Init4 R3 InR3;
state Init4
    --> Init5 R4 InR4;
state Init5
    --> Init6 R5 InR5;
state Init6
    --> Init7 R6 InR6;
state Init7
    --> etat1 R7 InR7;
state etat1
    --> etat2 R1 R1A2 A2R1;
state etat2
    --> etat3 R2 R1A2 R2A21 A2R2;
state etat3
    --> etat4 R4 R2A21 R4A2 A2R4
        R6 R6A1 R7A12 A1R6;
state etat4
    --> etat5 R4 R4A2 R6A2 A2R4;
state etat5
    --> etat6 BuffClk R4M;
state etat6
    --> etat7 R10;
state etat7
    --> etat8 R8 A2R8 R2A21 R10A2
        BuffClk R4M;
state etat8
    --> etat9 R2 A2R2 R8A2 R2A22
        R10;
state etat9
    --> etat10 BuffClk R2M
        R9 R10A1 R6A1;
state etat10
    --> etat11 R4 A2R4 R4A2 R8A2
        R2 A1R2 R9A1 R6A1
        R10;
state etat11
    --> etat12 R6 A1R6 R1A1 R10A1
        R4 A2R4 R4A2 R9A2
        BuffClk R2M;
state etat12
    --> etat13 R8 A2R8 R1A2 R6A2
        R2 A1R2 R8A12 R6A1
        R10;
state etat13
    --> etat14 BuffClk R8M
        R2 A2R2 R2A22 R3A2
        R8 A1R8 R10A1 R7A11;
state etat14
    --> etat15 R10
        R9 R9A1 R8A11;
state etat15
    --> etat16 R1 A2R1 R0A2 R10A2
        BuffClk R2M
        R9 R9A1 R5A1;
state etat16
    --> etat17 R7 A1R7 R7A11 R8A12
        R10;
state etat17
    --> etat18 R3 A2R3 R3A2 R10A2
        BuffClk R9M;
state etat18
    --> etat19 R2 A2R2 R2A22 R3A2
        R10;
state etat19
    --> etat20 R1 A2R1 R1A2 R6A2
        R5 A1R5 R5A1 R10A1
        BuffClk R7M;
state etat20
    --> etat21 R6 A1R6 R5A1 R9A1
        R10;
state etat21
    --> Out R7 A1R7 R8A11 R10A1;
state out
    --> Init0 IOPadTriState[7:0]=11111111;
end
```

Figure III.25: PC pour une architecture à base de bus et de registres communs

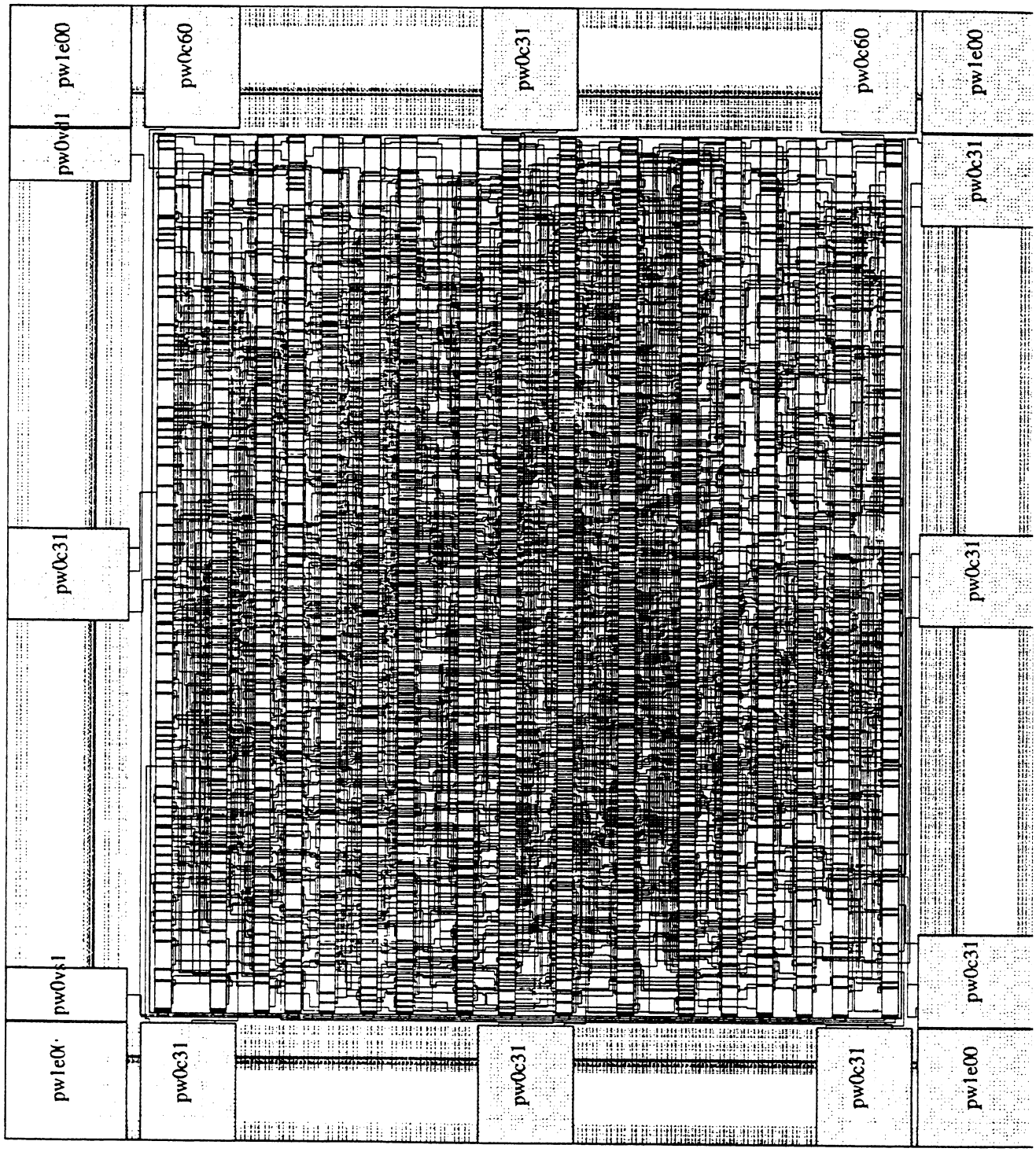
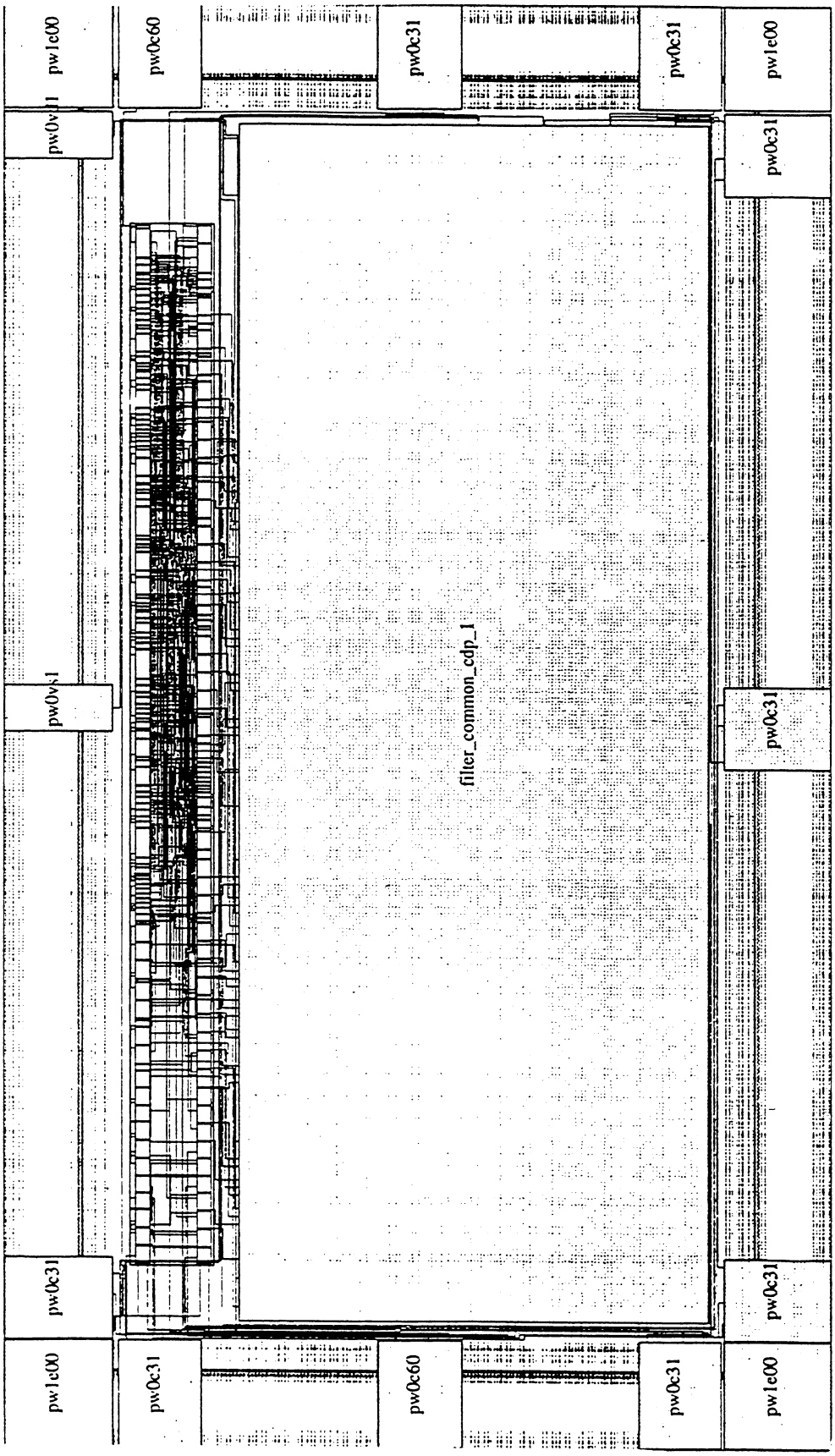


Figure III.26: Layout en cellules standards pour une architecture à base de bus et registres communs

```
# General Information:
system_name            SPARC UNIX
user_Id                anne
home_directory         /users/csi/anne
mainsail_version       12.15
vlsi_tools_version     v8r3.5
tool                   Chip Compiler
technology             cmn16
last_operation         RouteBlock
design                 global_common_f


# Nets And Cells:
nets                   1071
arbitrary_blocks       0
cell_areas             1
standard_cells         1286
pads                   12
pins_per_net           3.55
nets_with_weight       138
nets_with_maxcap       0
net_with_max_pins      U5.i_189.Z 34
clock_net              no clock nets


# Total Area (excluding scribe lines) :
package                global_common_default
chip_size              4844.00 X 4330.00 lambda (152.57 X 136.38 mil)
core_size              3762.00 X 3150.00 lambda (118.49 X 99.21 mil)
transistors            13414
core_density           .1132E-02 trans/sq-lam (.2853 gates/sq-mil)


# Routing Factor:
block_routing_factor   .77
core_routing_factor    3.26
padring_routing_factor .51


# Post Route:
CM2CM2                 5205
metal                  485241 lambda
metal2                 527126 lambda
unroutes_in_cell_routing  0
unroutes_in_block_routing 0


# Cell Areas:
Area global_common_sc
  size                 3762 X 3150 lambda
  routing_factor       1.51
  transistors          13414
  density              .1132E-02 trans/sq-lam (.2853 gates/sq-mil)
  standard_cells       1299
  row_end_cells        34
  feed_thru_cells      53
end_area
```

Figure III.27: Carctéristiques du layout de la figure III.26



Compass Design Automation VGS plot global_common by anne on 5-Nov-92 at 8:47 A.M.

Figure III.28: Layout en tranches de bit pour une architecture à base de bus et registres communs

```
# General Information:
system_name          SPARC UNIX
user_Id              anne
home_directory       /users/csi/anne
mainsail_version     12.15
vlsi_tools_version   v8r3.5
tool                 Chip Compiler
technology           cmn16
last_operation       RouteBlock
design               global_common_f


# Nets And Cells:
nets                 210
arbitrary_blocks     1
cell_areas           1
standard_cells       197
pads                 12
pins_per_net         3.42
nets_with_weight     0
nets_with_maxcap     0
net_with_max_pins    U5.i_189.Z 34
clock_net            no clock nets


# Total Area (excluding scribe lines) :
package              global_common_default
chip_size            8715.00 X 3886.00 lambda (274.49 X 122.39 mil)
core_size            7558.00 X 2229.00 lambda (238.05 X 70.20 mil)
transistors          1898
core_density         .1127E-03 trans/sq-lam (.0284 gates/sq-mil)


# Routing Factor:
block_routing_factor    1.31
core_routing_factor     1.47
padring_routing_factor  .28


# Post Route:
CM2CM2               1016
metal                159790 lambda
metal2               281456 lambda
unroutes_in_cell_routing  0
unroutes_in_block_routing 0


# Cell Areas:
Area global_common_sc
  size               1466 X 1202 lambda
  routing_factor     1.34
  transistors        1898
  density            .1077E-02 trans/sq-lam (.2714 gates/sq-mil)
  standard_cells     197
  row_end_cells      14
  feed_thru_cells    10
end_area
```

Figure III.29: Caractéristiques du layout de la figure III.28

Résumé

La synthèse architecturale consiste à obtenir automatiquement la description d'un circuit en termes de blocs interconnectés à partir de sa description comportementale. Pour la synthèse dite de haut niveau cette description comportementale est de type algorithmique. A partir d'une description algorithmique une structure interne représentant un graphe de dépendance de données est extraite. Cette structure définie les contraintes de précédence entre les opérations de la spécification initiale. Les deux étapes principales de la synthèse sont l'ordonnancement et l'allocation de ressources. Chaque étape de la synthèse de haut niveau est un problème combinatoire NP-Complet. Afin de diminuer la complexité de l'espace de solutions à explorer, ces deux étapes sont généralement réalisées séparément. Dans un souci d'universalité, des architectures cibles et des heuristiques flexibles par rapport à ces architectures ont été définies.

Dans un premier temps, l'ordonnancement affecte les tâches à des unités de temps et détermine le nombre d'opérateurs. La méthode d'ordonnancement choisie est une amélioration de la méthode orientée par les forces.

Dans un second temps, l'allocation de ressources assigne des opérateurs physiques aux opérations, des registres aux variables et des connexions aux transferts de données de la description initiale. Cette dernière est également scindée en deux étapes l'allocation de registres et d'opérateurs est réalisée en premier lieu avec comme objectif de préparer l'étape suivante d'allocation des connexions.

Une méthode d'allocation de registres et d'opérateurs originale est définie. Celle-ci a comme objectif d'être flexible par rapport à l'architecture cible du circuit. Cette architecture pourra utiliser des multiplexeurs ou des bus, des registres dédiés aux opérateurs ou des registres communs. Une phase d'alignement des opérandes exploite la commutativité des opérateurs dans le but de minimiser les interconnexions. La phase d'allocation des interconnexions est définie spécifiquement pour chaque architecture cible.

Toutes ces méthodes sont implantées dans un système complet fait de deux parties : la synthèse de haut niveau et la synthèse RTL.

Mots clefs

Synthèse architecturale, Synthèse de haut niveau, Synthèse RTL, Compilateur de silicium, Ordonnancement, Allocation de ressources, Allocation de registres, Architecture cible

Abstract

Architectural synthesis consists in the automatic block netlist generation of a design from its behavioural description. For high level synthesis, this description is an algorithmic specification. from which is extracted an internal format representing data dependencies. It defines the precedence relations between operations. The two main steps of high level synthesis are scheduling and resource allocation. Each of these steps is a NP-Complete problem. To reduce the solution space exploration, scheduling and resource allocation are realised separately. To be as independent as possible from an application domain, architecture styles and flexible heuristics dedicated to these styles have been defined.

First of all, scheduling assigns each operation to a time unit. A force directed scheduling method has been chosen.

Then, the resource allocation assigns physical operators to operations, registers to variables and connections to data transfers. This step is divided into two tasks, register and operator allocation is performed first and prepares the next task namely the connection allocation.

An original method for resource allocation has been defined. Its main objective is to be flexible with respect to architecture styles which can be either multiplexer based or bus based, with dedicated or common registers. Then an operand alignment method exploits operator commutativity. At the end, connection allocation is dedicated to each architecture style.

All these methods are implemented in a complete system made up of two parts : high level synthesis and RTL synthesis.