Synthèse optimisée sur les réseaux programmables de la famille Xilinx

Belgacem Babba

THESE

Présentée par

Belgacem Babba

Pour obtenir le grade de **Docteur**

de L'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

(arrêté ministériel du 30 Mars 1992)

Spécialité **Informatique**

Synthèse optimisée sur les réseaux programmables de la famille Xilinx

Date de soutenance: 20 Juin 1995

Composition du Jury:

Mesdames	**Anne Marie TRULLEMANS**	**Rapporteur**
	Professeur Gabrièle SAUCIER	
Messieurs	**Professeur Guy MAZARÉ**	**Président**
	Michel CRASTES de Paulet	
	Professeur Michel ISRAEL	**Rapporteur**

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

Remerciements

Je tiens tout d'abord à remercier Madame Gabrièle SAUCIER, Professeur à l'ENSIMAG et directrice du Laboratoire Conception de Systèmes Intégrés, pour m'avoir accueilli dans son laboratoire, pour avoir dirigé mes recherches, pour sa participation à l'élaboration des idées et le temps qu'elle m'a consacré, ainsi que le soin qu'elle a porté à la lecture de mon rapport de thèse. Son enthousiasme pour la recherche a constitué un modèle que j'ai tâché de suivre durant ces années.

Je remercie également :

Monsieur Guy Mazaré, Professeur et directeur de l'ENSIMAG, de me faire l'honneur de présider ce jury,

Madame Anne-Marie Trullemans-Anckaert, Docteur au Laboratoire de Microélectronique à l'Université de Louvain, pour l'examen et la critique qu'elle a bien voulu apporter à ce travail en tant que rapporteur et membre du jury,

Monsieur Michel Israël, Professeur et Directeur du département Informatique/mathématique de l'université d'Evry, de me faire l'honneur de participer au jury en tant que rapporteur, pour l'examen et la critique qu'il a apporté à ce travail,

Je remercie tout particulièrement Michel Crastes, en tant que membre du jury, pour avoir dirigé mes recherches, pour le grand soin qu'il a porté à la lecture de mon rapport de thèse et pour m'avoir tant aidé aussi bien sur le plan scientifique que moral.

Je remercie tous les membres du CSI pour l'agréable ambiance de travail et tous ceux qui, de près ou de loin, ont contribué à l'élaboration de ce travail.

Résumé

Cette thèse se situe dans le cadre de la synthèse logique. Elle a pour objet la synthèse logique optimisée de circuits sur réseaux programmables à base de "tables de vérité" de type "Xilinx". Ces réseaux programmables ont été à l'origine du premier succès commercial des réseaux reprogrammables à faible granularité.

Une première solution pratiquée industriellement a consisté à associer une bibliothèque équivalente de primitives logiques simples de type "cellule standard" à un réseau Xilinx. Une telle approche conduit à une très pauvre utilisation de la technologie cible car elle ne tire pas profit de la richesse de la cellule de base.

Cette thèse s'intéresse, en conséquence, à des approches plus ciblées. Il s'agit de décomposer de façon optimisée les parties combinatoires en sous-fonctions "saturant" les possibilités des cellules élémentaires. Pour ceci, le traitement des fonctions booléennes sera effectué dès l'étape de factorisation en fonction du but final.

Après un rappel de la factorisation "lexicographique", qui a comme fondement l'existence d'un ordonnancement des entrées, une méthode de décomposition en sous fonctions de k variables est proposée. Elle sert de base à des méthodes de décomposition technologique pour les séries Xilinx 3000 et Xilinx 4000.

Deux alternatives à cette factorisation lexicographique sont proposées, une factorisation utilisant une représentation par diagramme de décision binaire (ROBDD) et une factorisation algébrique classique adaptée aux caractéristiques de la cible Xilinx.

La dernière étape de synthèse concerne de façon plus fine le regroupement des sous-fonctions dans la cellule physique Xilinx et se préoccupe de l'optimisation des points de mémorisation, des buffers et des ressources d'horloge.

Une évaluation sur un ensemble d'exemples internationaux et industriels démontre l'efficacité des méthodes proposées. Ce travail a fait l'objet d'un transfert technologique vers le logiciel industriel ASYL+.

Mots Clés :

Synthèse logique - Réseaux programmables - Diagramme de décision binaire Factorisation lexicographique - Factorisation algébrique - Optimisation - Décomposition technologique

Abstract

This work concerns the optimized logic synthesis for LUT based FPGAs, namely Xilinx FPGAs. These programmable devices led to the first commercial success of small granularity FPGAs devices .

The first, easy-to-build, industrial logic synthesis solution has consisted in associating a library of simple gates equivalent to the Xilinx FPGAs. Such an approach results in very poor use of the target technology because it does not take full profit of the characteristics of basic cell.

Therefore, this thesis is focused on more dedicated approaches. The aim is to optimally decompose combinatorial parts into subfunctions "saturating" the basic cell. For this, the early processing of Boolean functions is already already takes into account the final target.

The first section recalls the principles of the "Lexicographical" factorization, based on the existence of an input order for Boolean functions. decomposition method in subfunctions of k inputs is proposed in section 2. It serves as basis to the mapping methods for Xilinx series 3000 and 4000.

Two alternatives to this lexicographical factorization are proposed in section 3: a factorization using a representation by Reduced Ordered Binary Decision Diagrams (ROBDD), and a classic algebraic factorization adapted to the characteristics of the Xilinx target.

The last synthesis step concerns the clustering of subfunctions into the physical Xilinx cells. This step deals also with the optimization of Flip-Flops, global buffers and clock resources.

Practical results on a large set of industrial and academic benchmarks demonstrate the efficiency of the methods discussed in this work. This work has made led to a technological transfer to the ASYL+ industrial software.

Keywords :

Logical synthesis - Programmable devices - Binary Decision Diagram Lexicographical factorization - Algebraic factorization - Optimization - Mapping

A BAYA,

Pour son aide, sa présence à mes côtés tout au long de la réalisation de ce travail,

Pour la qualité de son amour.

A mes Parents,

Pour les tous petits riens et les grandes choses qui ont contribués à faire de moi ce que je suis.

A toute ma famille,

A toute ma belle famille.

Introduction

Cette thèse a comme objet la synthèse logique optimisée de circuits sur réseaux programmables de type "Xilinx".

Ces réseaux programmables, qui ont été à l'origine du premier succès commercial des réseaux reprogrammables à faible granularité, sont, de façon simplifiée, constitués de cellules élémentaires appelées LUT (*Look Up Table*) pouvant implanter, par programmation, n'importe laquelle de 2^{2^k} fonctions booléennes de k variables (k étant égale à 3, 4 ou 5 suivant les séries).

Réaliser une synthèse logique sur ces réseaux, revient à identifier les zones combinatoires, à décomposer les fonctions booléennes correspondantes en sous-fonctions de k variables, puis à réaliser une optimisation finale tenant compte du regroupement en cellules et des ressources telles que point de mémorisation, horloge, buffer, etc... Une telle décomposition est très peu intuitive pour un concepteur, en particulier pour des circuits complexes. Une optimisation manuelle est impossible.

La synthèse logique est en général décomposée en deux principales étapes. La première étape est celle de la minimisation des fonctions Booléennes écrites en sommes de monômes et de la factorisation ou recherche de sous expressions communes. Cette étape est souvent considérée comme indépendante de la technologie. La deuxième étape est la décomposition technologique qui a pour but de transformer les fonctions Booléennes en un ensemble de fonction élémentaires dont chacune peut être implantée par un élément existant de la technologie cible, et de spécifier les interconnexions de ces éléments.

Une première solution pratiquée industriellement a consisté à associer une bibliothèque équivalente de primitives logiques simples de type "cellule standard" à un réseau Xilinx . Cette approche consiste à implanter sur les réseaux Xilinx, les portes de base habituellement utilisées pour les circuits à la demande et à utiliser les méthodes et les logiciels de synthèse correspondants.

Une telle approche conduit à une très pauvre utilisation de la technologie cible car elle ne tire pas profit de la richesse de la cellule de base.

Cette thèse s'intéresse, en conséquence, à des approches plus ciblées. Il s'agit de décomposer de façon optimisée les parties combinatoires en sous-fonctions "saturant" les possibilités des cellules élémentaires. Pour ceci, le traitement des fonctions booléennes sera effectué dès l'étape de factorisation en fonction du but final.

Le chapitre 2 de la thèse sera consacré au rappel de la factorisation "lexicographique" proposée dans [Abo92]. A partir de cette factorisation, qui a comme fondement l'existence d'un ordonnancement des entrées, une méthode de décomposition en sous fonctions de k variables est proposée au Chapitre 3. Elle sert de base à des méthodes de décomposition technologique pour les séries Xilinx 3000 et Xilinx 4000.

Dans le chapitre 4, des alternatives à cette factorisation lexicographique sont proposées. En premier lieu, une factorisation utilisant une représentation par diagramme de décision binaire (ROBDD) est utilisée. La construction des ROBDD est basée sur l'ordre des variables introduit par la factorisation lexicographique. Cette méthode permet, entre autre, d'augmenter le partage de la logique et est spécialement intéressante dans l'optimisation de surface.

Puis, on réutilise une factorisation algébrique plus classique mais néanmoins adaptée aux caractéristiques de la cible Xilinx : cette factorisation appelée factorisation algébrique restreinte, utilise des filtres spécifiques pour reinjecter les sous-fonctions partagées à granularité trop fine pour respecter la cible technologique et le critère d'optimisation.

Les résultats de ces trois approches de factorisation sont analysés après placement/routage, pour dégager l'espace des solutions couvertes tant pour l'optimisation surface que pour l'optimisation chemin critique.

Le cinquième chapitre concerne de façon plus fine le regroupement des sous-fonctions dans la cellule physique Xilinx et se préoccupe de l'optimisation des points de mémorisation, des buffers, des ressources d'horloge etc...

Le dernier chapitre donne la synthèse de l'ensemble des résultats qui sont comparés avec succès aux résultats d'outils commerciaux. Ce travail a fait l'objet d'un transfert technologique vers un logiciel industriel.

Chapitre I
Introduction aux réseaux programmables

1.1 Les réseaux programmables

Les réseaux programmables sont des circuits intégrés personnalisables par des techniques électriques ne nécessitant pas d'interventions au niveau de la fabrication. Ces circuits peuvent être programmés une ou plusieurs fois et sont utilisés soit comme produits définitifs pour de petites séries soit à des fins de prototypage.

Les mémoires ROM ou PROM (*Read Only memory*) peuvent être considérées comme les premiers circuits programmables. Elles peuvent être utilisées pour implanter des fonctions Booléennes en stockant les tables de vérité correspondants aux formes canoniques des fonctions. Les PROMs sont des supports peu efficaces pour l'implantation de la logique et sont plutôt utilisées pour le stockage d'informations.

Les premiers réseaux programmables utilisés pour l'implantation de la logique sont les réseaux de type PLD (*Programmable Logic Device*). Un PLD est typiquement constitué d'un ensemble de portes ET (étage ET) connectées à un ensemble de portes OU (étage OU).

On distingue deux types de PLD:

• les PAL (*Programmable Array Logic*) pour lesquels le plan OU est figé et seul le réseau ET est programmable.

• les PLA (*Programmable Logic Array*) pour lesquels les plans ET et OU sont tous deux programmables et ceux-ci permettent le partage de monômes entre fonctions.

La figure 1.1 montre ces deux types de PLD. Un point entre deux lignes signifie que l'on programme le PLD pour qu'une connexion s'établisse entre les deux pistes pour introduire une variable dans un monôme ou un monôme dans une fonction.

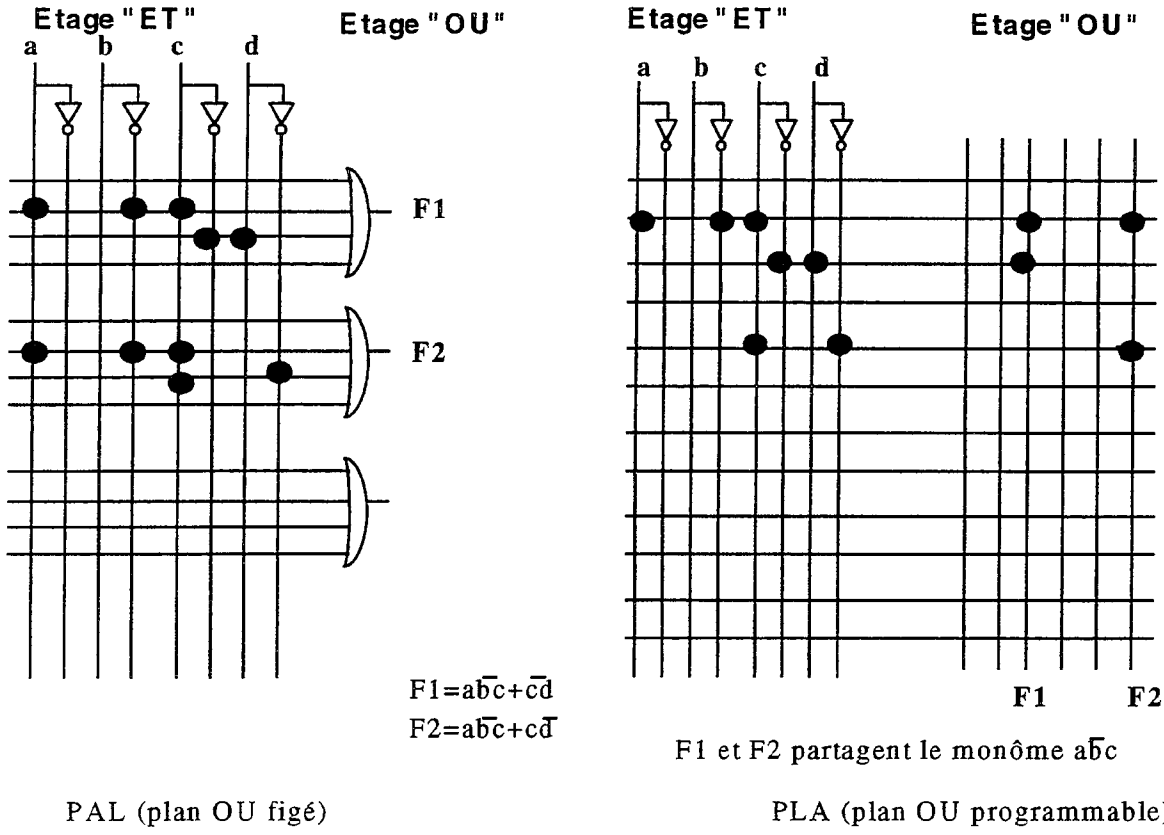


Figure 1.1: Structure d'un PLD.

Ces réseaux programmables à forte granularité sont très efficaces pour implanter des fonctions écrites sous forme de somme de monômes et sont utilisés pour implanter en particulier les contrôleurs ou les séquenceurs. Les PAL les plus connues sont les PAL22V10 de AMD.

Les PLD sont actuellement étendus à des réseaux à multiples plan ET/OU et sont appelés multi-PLD ou CPLD (*Complex PLD*). Un exemple de multi-PLD est le MACH210 de AMD.

La deuxième génération de réseaux programmables est plus adaptée à l'implantation de la logique. Elle est de granularité plus fine et interconnecte des cellules pouvant implanter un ensemble significatif de fonctions Booléennes élémentaires. Les cellules et les interconnexions sont alors personnalisables. Ces réseaux sont appelés "*Field programmable Gate Array*" ou FPGA.

Les familles de réseaux programmables sont caractérisées par la structure de la cellule de base et par les réseaux d'interconnexions.

Les cellules de bases des FPGA appartiennent essentiellement à deux familles:

- les cellules à base de multiplexeurs; les cellules de base implantent un arbre de multiplexeurs et les fonctions réalisées sont obtenues en fixant les entrées des multiplexeurs (Actel, Quicklogic). Les FPGA à base de multiplexeurs les plus connus sont les circuits commercialisés par la société Actel. La figure 1.2 représente la cellule de base de la famille Actel2 [Act94].

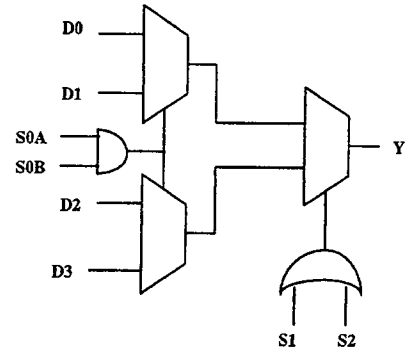


Figure 1.2: Cellule à base de multiplexeurs de Actel2.

- les cellules implantant n'importe quelle fonction de k variables; sa fonction est alors précisée par le contenu d'une table de vérité stockée appelée "*look up table*" ou LUT. Un exemple de cellule à base de LUT est celui de la série FLEX8000 d'Altera et des séries 2000, 3000 et 4000 de Xilinx. Les réseaux Xilinx sont l'objet de cette thèse.

La structure des interconnexions (appelée aussi architecture de routage) est soit réalisée par des bandes de connexions comparables aux réseaux prédiffusés habituels soit par des réseaux à deux dimensions.

Les technologies utilisées pour programmer les interconnexions sont à base de:

• Fusibles et Anti-fusibles

Les fusibles sont des connexions existantes qui sont coupées par l'application d'une forte tension. Les anti-fusibles utilisent le même principe

mais la connexion est alors créée. Les connexions sont définitivement grillées . Cette technique de programmation ne permet pas la reprogrammation.

• transistors EPROM et transistors EEPROM

Ce mode de programmation est dit à grille flottante. Il utilise des transistors dont la grille n'est pas connectée. On charge la grille par l'application d'une forte tension. Elle peut être déchargée soit par U.V. (EPROM), soit électriquement par application d'une tension inverse (EEPROM). Pour déprogrammer une EPROM, il est donc nécessaire de la sortir de la carte.

• transistors passants contrôlés par une RAM statique

La programmation est stockée dans une RAM placée sur le circuit programmable. Par conséquent, elle doit être reprogrammée à chaque mise sous tension. Habituellement, pour permettre la reprogrammation, les RAM sont reliées soit à une ROM externe, soit à un micro-ordinateur par une liaison série.

Le tableau 1.1 résume les principales caractéristiques des réseaux programmables les plus connus.

Société	Xilinx	Actel	Quicklogic	Altera
Type de la cellule de base	LUT	MUX	MUX	bloc PLD
Fonctions logiques	toute fonction de 5 variables (3000)	La plupart des fonctions de 4 variables	toute fonction de 3 variables	Large somme de monômes, avec partage de monômes
Architecture générale	Matrice de cellule	Lignes de cellules	Matrice de cellule	PLD Hiérarchique
Technologie de programmation	RAM Static	Anti-fusible	Anti-fusible	EPROM
Technologie	1.2µm, 0.8µm	2µm, 1.2µm, 0.8µm	1.0µm	0.8µm
Nombre maximum de cellules par circuit	320 (3000) 1024 (4000)	1377	768	192

Tableau 1.1: Principales caractéristiques des réseaux programmables.

1.2 Les réseaux programmables Xilinx

Dans cette étude, les cibles considérées sont les réseaux programmables à base de LUT et plus précisément les réseaux Xilinx incluant la série 3000 et la série 4000. Dans cette architecture la cellule de base est appelée bloc logique configurable (CLB pour *configurable logic block*) [Xil94].

1.2.1 Structure du CLB des réseaux Xilinx série 3000

La cellule de base de la série 3000 de Xilinx est constituée d'un bloc combinatoire, de deux points mémoire et de la logique de commande (figure 1.3).

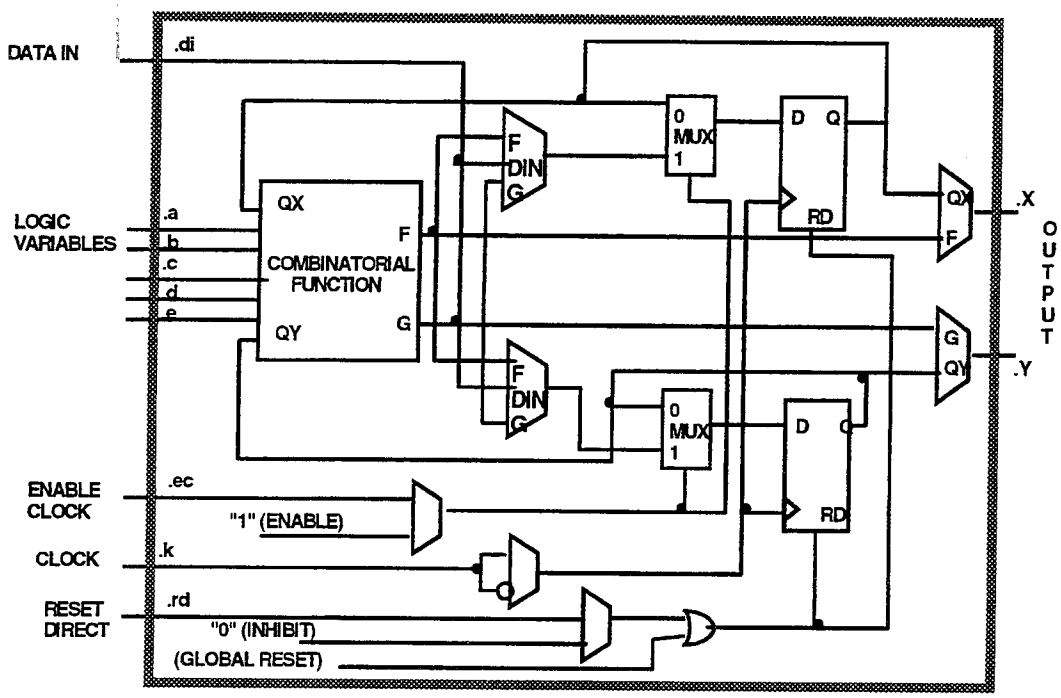


Figure 1.3: CLB de Xilinx série 3000.

La partie combinatoire peut implanter l'une des configurations suivantes:

- N'importe quelle fonction d'au plus cinq variables.

- Deux fonctions de quatre variables tout au plus chacune, de telle sorte que le nombre total des variables des deux fonctions est de cinq tout au plus.

Les deux configurations sont représentées en figure 1.4.

Le deux points mémoire sont alimentés, soit par les sorties du bloc combinatoire, soit par une entrée supplémentaire (DI *Direct input).* L'horloge des deux points mémoire doit être connectée au même signal.

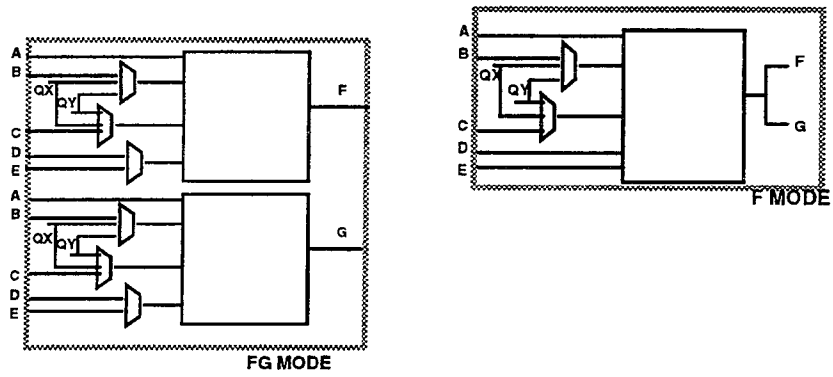


Figure 1.4: Configurations du bloc combinatoire du CLB Xilinx série 3000.

1.2.2 Bloc d'entrée sortie Xilinx série 3000

Un bloc d'entrée sortie de la série 3000 est constitué de deux points mémoire, d'un buffer d'entrée et d'un buffer de sortie. Le buffer de sortie peut être configuré en une porte trois états (figure 1.5). Cette dernière permet l'utilisation de l'entrée du bloc comme une entrée bidirectionnelle.

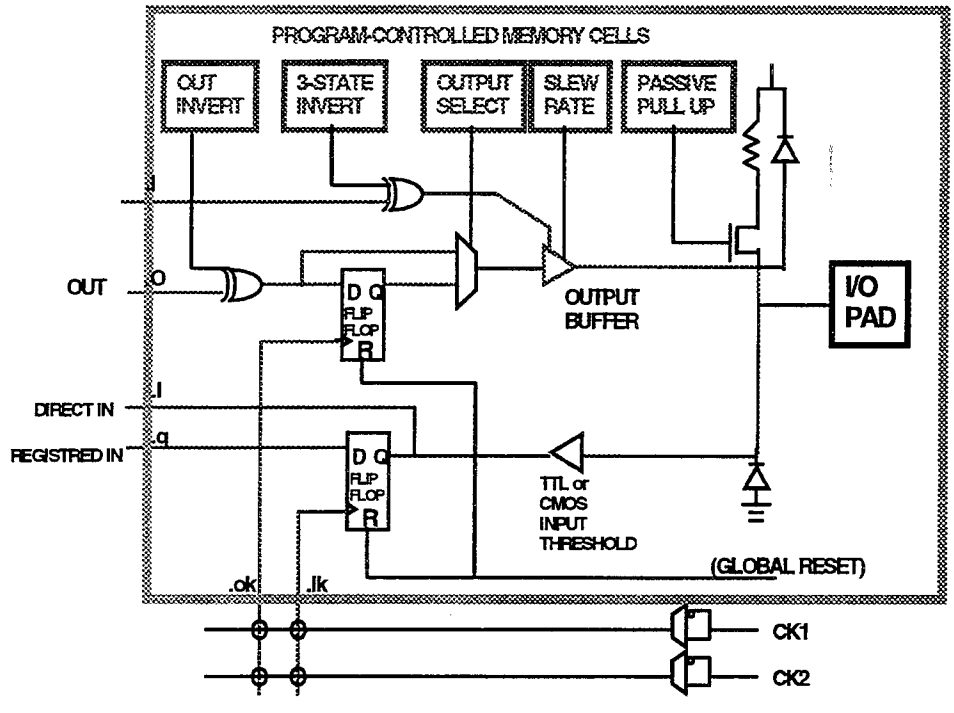


Figure 1.5: Bloc d'entrée sortie de la série 3000.

1.2.3 Organisation du réseau d'interconnexions

L'organisation des connexions dans les réseaux Xilinx est représentée en figure 1.6. Il s'agit d'un réseau à deux dimensions. Il est constitué essentiellement de:

- lignes métalliques

- points d'interconnexions programmables ("*PIPs*"). Un *PIP* est un commutateur constitué d'une cellule RAM.

- matrices de commutateurs; chaque matrice de commutateurs présente cinq points de connexions par face et permet la liaison des connexions suivant certaines configurations (figure 1.7). Cette matrice de commutateurs représente un handicap important pour le routage de la série 3000 puisque chaque matrice présente des configurations limitées et fixées.

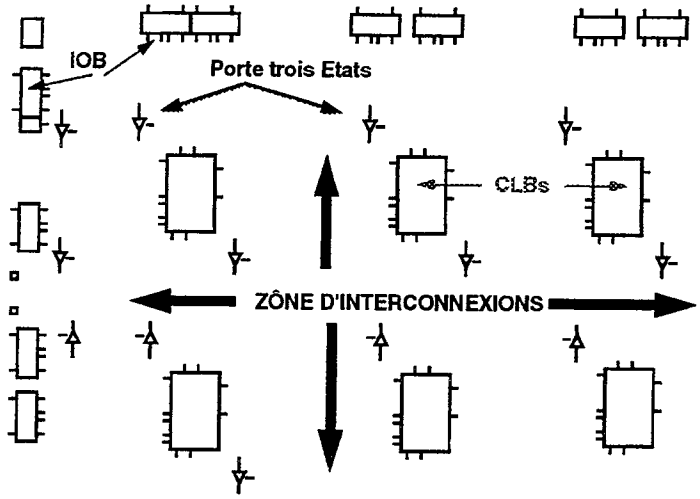


Figure 1.6: Organisation du réseau d'interconnexions Xilinx série 3000.

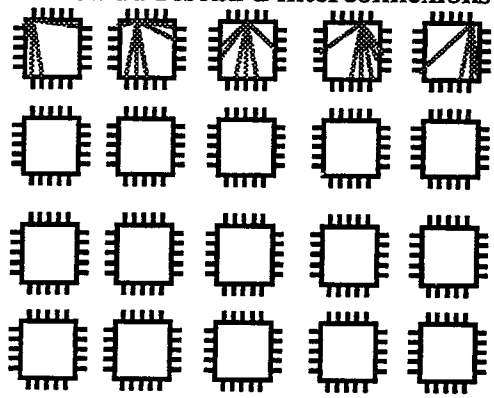


Figure 1.7: Différentes configuration des matrices de commutateurs Xilinx série 3000.

Pour la série 3000, les ressources de routage se divisent essentiellement en trois groupes d'interconnexions:

- Les *interconnexions générales* sont en grand nombre et utilisent des matrices de commutateurs. Ce groupe est le moins performant en temps de traversée.

- Les *connexions directes* permettent de connecter les sorties d'un CLB à certaines entrées des CLBs voisins. Ces connexions offrent un délai minimal mais elles n'offrent pas beaucoup de possibilités (figure 1.8).

- Les *connexions longues* parcourent le circuit entier en hauteur ou en largeur (Vertical long lines, Horizontal long lines). Ces connexions sont très performantes et peu nombreuses.

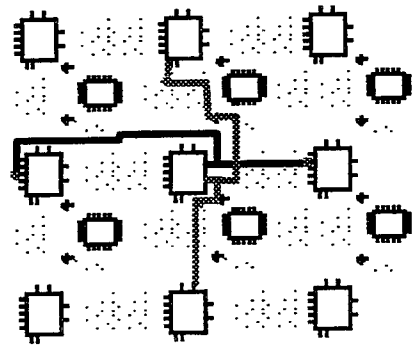


Figure 1.8: connexions directes

1.2.4 La cellule de la série 4000

La figure 1.9 représente la structure d'un bloc logique (CLB) de la série 4000. Le CLB de la série 4000 peut synthétiser trois fonctions logiques appelées "F", "G" et "H". Deux sorties de ces trois fonctions sont utilisables à l'extérieur du CLB. Le CLB comporte aussi, comme pour la série 3000, deux points mémoire utilisant soit les sorties des parties combinatoires, soit une entrée directe. Les points mémoire du CLB de la série 4000 ont un "set" ou "reset" asynchrone.

Les blocs combinatoires F et G réalisent n'importe quelle fonction d'au plus quatre variables. Le bloc H est fonction d'une variable "h" et des deux fonctions F et G.

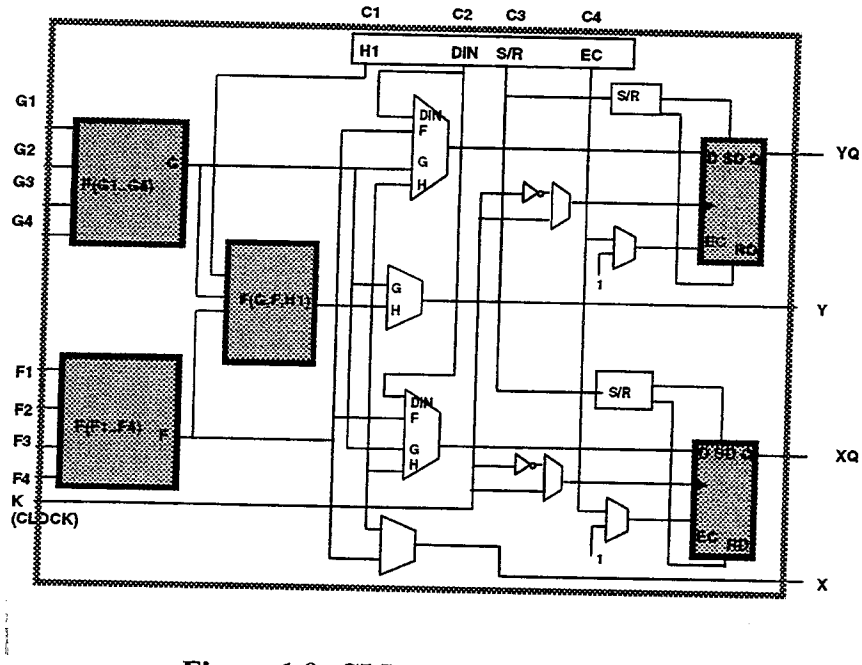


Figure 1.9: CLB de Xilinx série 4000

1.2.5 Le bloc d'entrée/sortie de la série 4000

Le bloc d'entrée sortie de la série 4000 (figure 1.10) est très similaire à celui de la série 3000.

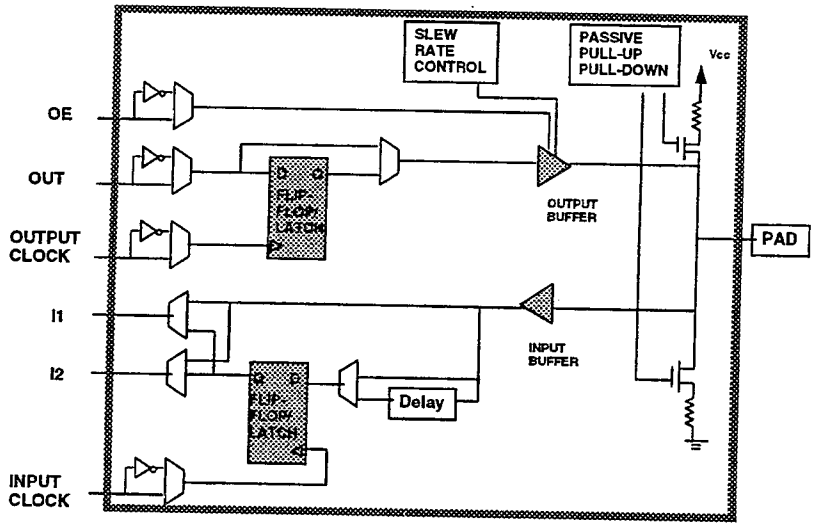


Figure 1.10: Bloc d'entrée sortie de la série 4000.

1.2.6 L'organisation des interconnexions de la série 4000

L'organisation des interconnexions pour la série 4000 est semblable à celle de la série 3000. Les interconnexions sont constituées par:

- les trois groupes d'interconnexions de la série 3000 avec un nombre plus important d'interconnexions de chaque groupe et variable suivant la capacité du boîtier.

- des *interconnexions double longueur* (figure 1.11) sont semblables aux interconnexions générales mais alternées sur deux lignes de matrices de commutateurs.

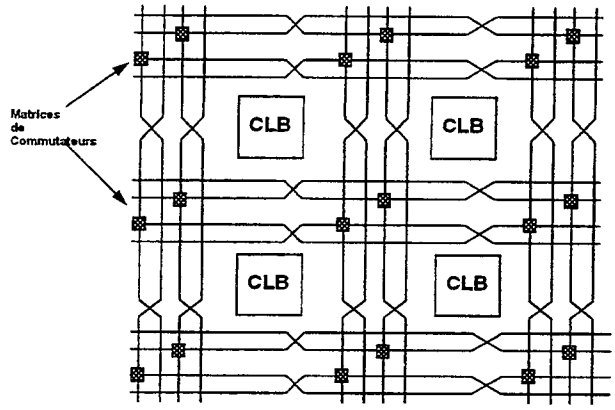


Figure 1.11: Interconnexions double longueur

La matrice de commutateurs de la série 4000 et avec le passage à la technologie 0.8µ (1.2µ pour la série 3000) est améliorée pour permettre plus de configurations (figure 1.12).

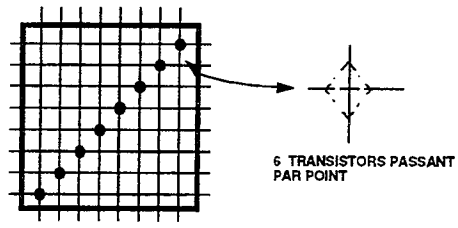


Figure 1.12: Matrice de commutateurs (Xilinx série 4000).

1.3 Flot de conception sur FPGA

Le flot de conception typique sur FPGA [Xil94] [Bro93] est présenté par la figure 1.13. On remarque ici l'importance de la phase de la synthèse logique. En effet, vue la forte granularité des FPGA la réalisation de circuit devient très complexe à gérer par un concepteur, surtout si on se place dans le cadre

d'un prototypage où la nécessité de produire rapidement un circuit est évidente. Pour contourner ce problème de complexité, on décrit généralement une bibliothèque d'éléments fictifs constituée par des portes combinatoires et séquentielles simples. La saisie schématique ou la synthèse de haut niveau se fera alors sur cette bibliothèque virtuelle et la synthèse logique se chargera de faire l'optimisation du circuit en décrivant les fonctions à implanter sur la cellules du FPGA.

La phase de placement routage a pour but d'affecter aux cellules issues de la synthèse logique des positions physiques dans le circuit puis de déterminer toutes les ressources de routage qui interviennent dans le routage de chaque signal du circuit. Les algorithmes de placement, pour FPGA, sont très proches de ceux utilisés pour les circuit VLSI, par contre le routage est différent vu l'existence de ressources d'interconnexions fixes. Le routage de 100% des signaux pour un même circuit n'est pas garantie et dépend directement du circuit en entrée de l'outil de placement-routage, La synthèse logique doit donc tenir compte de l'organisation des interconnexions et préparer la phase de placement-routage.

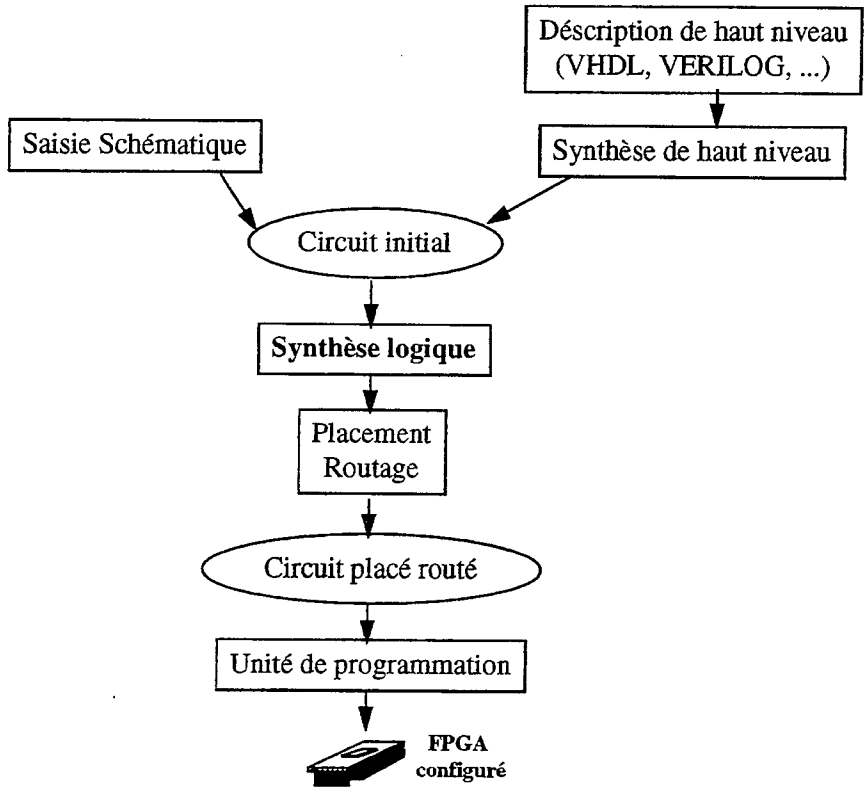


Figure 1.13: Flot de conception pour les FPGA

1.4 Etapes d'une synthèse logique de bas niveau

La figure 1.14 montre les différentes étapes de la synthèse logique [Bab92]. La première étape est celle de l'extraction des zones combinatoires d'un réseau Booléen. Cette zone combinatoire est ensuite traduite en un ensemble de fonctions Booléennes. La deuxième étape est l'optimisation logique qui peut être décomposée en deux sous-étapes, celle de la minimisation des fonctions Booléennes écrites en sommes de monômes et celle de la factorisation ou recherche de sous expressions communes. Cette étape est souvent considérée comme indépendante de la technologie. La troisième étape est la décomposition technologique qui a pour but de transformer les fonctions Booléennes en un ensemble de fonction élémentaires dont chacune peut être implantée par un élément existant de la technologie cible, et de spécifier les interconnexions de ces éléments.

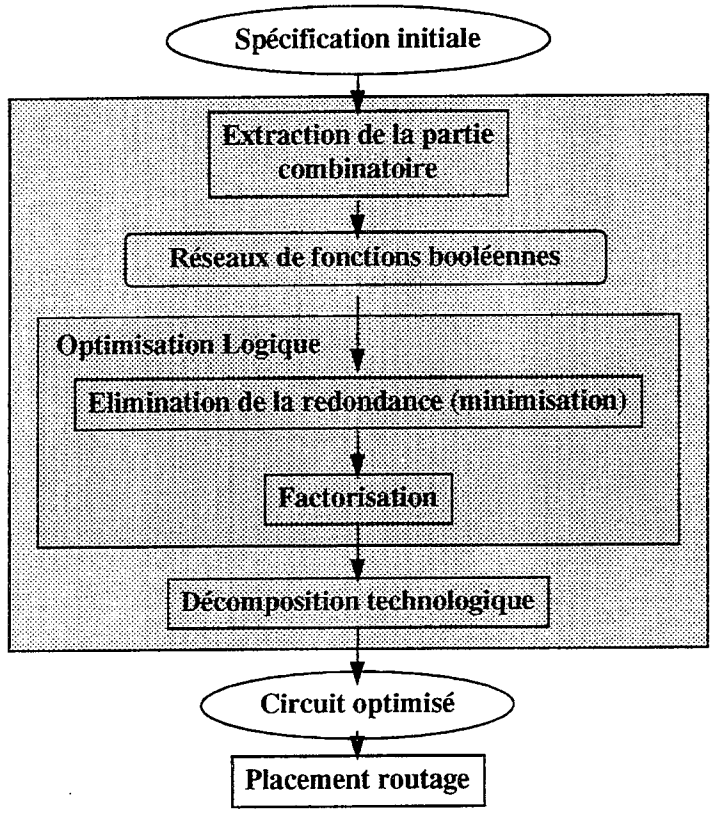


Figure 1.14: Etapes de la synthèse logique

Le but de ce travail est d'étudier des méthodes de synthèse spécifiques à ce type de réseaux. Les parties combinatoires étant extraites, on suppose que les

fonctions Booléennes ont été minimisées sous forme de somme de monômes [Sic88] [Duf91] et on s'intéressera dans les chapitres suivants à des méthodes de factorisation destinées à préparer l'implantation finale. Trois types de factorisation sont étudiés: la factorisation dite lexicographique, la factorisation à base de décomposition de Shannon et la factorisation algébrique restreinte. Les résultats de la décomposition technologique finale seront comparés sur ces trois types de factorisation.

Chapitre II
Factorisation Lexicographique

2.1 Introduction

La factorisation est un processus qui part d'un ensemble de fonctions Booléennes exprimées par des sommes de monômes et le transforme en un ensemble d'expressions factorisées. En général la complexité d'un circuit combinatoire réalisée par des cellules standard est étroitement liée au nombre de littéraux des formes factorisées représentant l'ensemble des fonctions Booléennes. Celui-ci est un bon estimateur de la surface active d'un circuit (surface occupée par les portes logiques). La factorisation a donc pour objectif essentiel de minimiser la complexité des formes factorisées et donc la logique qui sert à les réaliser.

L'étape de la factorisation doit préparer l'étape de la décomposition technologique. Il s'agit pour les réseaux considères de décomposer une fonction Booléenne en sous-fonctions de k variables. Pour ceci, une méthode de factorisation innovatrice a été proposée [Abo92] [Sau90]; il s'agit de la factorisation lexicographique qui précisément, en proposant un ordonnancement des variables d'entrées, facilitera la décomposition en sous-fonctions de k variables [Abo93].

2.2 Objectif et principe de la factorisation

Une définition des termes employés dans cette introduction se trouve dans la section 2.3 de ce chapitre.

La factorisation consiste à mettre en commun des expressions Booléennes appelées candidats diviseurs qui apparaissent plusieurs fois dans l'ensemble des fonctions Booléennes. Ces facteurs peuvent appartenir à une seule fonction Booléenne ou à plusieurs fonctions. La mise en commun de ces facteurs permet de diminuer la complexité, exprimée en terme d'occurrences de littéraux, des expressions Booléennes [Bra87a] [Sak93].

On peut énumérer essentiellement deux types de factorisation :

- Factorisation algébrique utilisant le produit algébrique (noyau algébrique).

- Factorisation Booléenne utilisant le produit Booléen (facteur ou noyau Booléen).

L'exemple suivant illustre la différence entre les noyaux algébriques et les noyaux Booléens.

Exemple :

Soient les deux fonctions Booléennes suivantes :

$F_1 = a.c.d.g + b.c.d + \bar{e}.c.d$

$F_2 = a.b.f.g + b.c.f + \bar{e}.b.f$

Le nombre de littéraux dans la forme polynomiale est égal à 20.

Les noyaux algébriques sont les suivants :

$K_1 = a.g + b + \bar{e}$ (F_1)

$K_2 = a.g + c + \bar{e}$ (F_2)

L'expression des deux fonctions Booléennes F_1, F_2 après factorisation algébrique devient :

$F_1 = c.d.(a.g + b + \bar{e})$

$F_2 = b.f.(a.g + c + \bar{e})$

$K = a.g + b.c + \bar{e}$ est un noyau Booléen et permet d'exprimer F_1, F_2 de la façon suivante :

$F_1 = c.d.(a.g + b.c + \bar{e})$

$F_2 = b.f.(a.g + b.c + \bar{e})$

L'expression $(a.g + b.c + \bar{e})$ est commune à F_1 et à F_2, et devient une sous-fonction commune notée SF.

$F_1 = c.d.SF$

$F_2 = b.f.SF$

$SF = a.g + b.c + \bar{e}$

Le nombre de littéraux dans le cas de la factorisation algébrique est de 12 littéraux. Tandis que dans le cas de génération de noyaux Booléens, on ne compte qu'une fois les littéraux de la sous-fonction commune, le nombre de littéraux est de 11.

On appelle gain associé à un noyau le nombre de littéraux gagnés en effectuant la factorisation par ce noyau. Il est égal au nombre de littéraux de la fonction avant la factorisation moins le nombre de littéraux après factorisation par ce noyau [Bra87a] [Sak93].

Une approche gloutonne de la factorisation consiste en la séquence d'étapes suivantes :

A) Génération des candidats diviseurs.
B) Tant qu'il y a des candidats diviseurs
- *Sélection des candidats de gain maximal en terme de nombre de littéraux.*
- *Division par les candidats sélectionnés.*

Fin tant que

Dans le cas Booléen, le gain associé à un facteur ne peut être connu qu'après la division par ce facteur ce qui rend difficile un choix fondé sur la notion de gain.

Dans le cas algébrique, le problème théorique revient à trouver un sous-ensemble de candidats diviseurs compatibles algébriquement entre eux et qui apporte un gain maximal. Malheureusement la sélection d'un tel sous-ensemble demande une recherche exhaustive des solutions et ne peut être envisagée dans la pratique. C'est pourquoi une approche gloutonne est souvent utilisé [Sak93].

Nous proposons dans ce chapitre une méthode originale, fondée sur la méthode de division par les conoyaux. Cette méthode restreint encore plus l'étape de factorisation en imposant un ordre sur les variables d'entrée pour l'ensemble des fonctions, afin de faciliter la connectique ultérieure. Elle permet de contrôler, d'une part, la dépendance des entrées en créant des cônes logiques structurés [Abo93] [Bab92] [Sau90] et, d'autre part, les connexions

entre cellules, en contrôlant aussi le « *fanout* » des cellules logiques utilisées (figure 2.1).

L'organisation de l'ensemble de fonctions Booléennes dans le réseau, est faite donc en cônes logiques dans lesquels les variables d'entrée sont rangées dans un ordre précis (le même pour chaque fonction). Il faut alors contrôler les intersections de ces cônes. La restriction imposée par l'ordonnancement des variables d'entrée assure une optimisation du routage, et le contrôle des intersections entre les cônes permet d'améliorer le routage entre les cellules ainsi que les *fanouts* associés à ces cellules.

La phase de décomposition technologique sur cellules de base sera réalisée couche par couche en respectant ces cônes logiques.

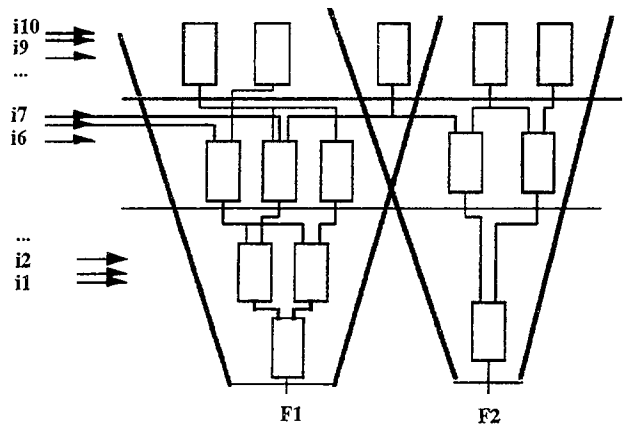


Figure 2.1. Cônes logiques virtuels avec l'ordre des entrées i1,i2...i6,i7...i9,i10

2.3 Définitions préalables

Définition 2.1: Diviseur algébrique et Booléen

D est un diviseur algébrique de F si :

F = D.H + R où D.H est un produit algébrique,

il n'existe pas H' tel que H < H' (H < H' ≠ H ≤ H' et H ≠ H') et F = D.H' + R, R étant une expression polynomiale.

Le quotient H et le reste R sont uniques.

Exemple :

$F = a.b.\bar{c} + a.b.e.f + a.b.g + e.\bar{f}$

$D = \bar{c} + e.f + g$ est un diviseur algébrique de F.

$F/D = a.b$ est le quotient de la division.

$R = e.\bar{f}$ est le reste de la division.

D est un diviseur Booléen de F si F = D.H + R où D.H est un produit Booléen. Un diviseur algébrique est en particulier Booléen, car la division Booléenne génère un ensemble de solutions contenant la solution algébrique. Le quotient et le reste Booléen ne sont pas uniques.

Exemple :

$F = a.b.c + c + d$

D = (a . c) est un diviseur Booléen de F, car F peut être écrite sous la forme $F = D.(a + b) + d$.

alors que D n'est pas un diviseur algébrique de F.

Définition 2.2: Facteur algébrique et Booléen

G est un facteur algébrique de F si F = G.H, tels que H est une expression Booléenne et G.H le produit algébrique de G par H.

G est un facteur Booléen de F si F = G.H, avec H une expression Booléenne et G.H le produit Booléen de G par H.

Définition 2.3: Substitution algébrique

Soient F et G deux fonctions Booléennes. La substitution algébrique de G dans F est la division algébrique de l'expression de F par l'expression de G et de son complément $\bar{G}$:

$F = H.G + H'.\bar{G} + R$

H (resp. H') est le quotient de la division algébrique de F par G (resp. $\bar{G}$)

Exemple :

$F = a.c + a.d + b.c + b.d + e.c + e.d + \bar{a}.\bar{b}.\bar{c}.f + r$

$G = a + b + e$

F peut s'écrire par substitution algébrique de G dans F :

$F = G.(c + d) + \bar{G}.f + r$

Définition 2.4: Expression libre

Une expression Booléenne polynomiale F est libre s'il n'existe pas de monôme m (avec m différent de 1) qui soit un facteur algébrique de F.

Exemples :

a.b + a.c	n'est pas libre car a est un facteur algébrique
a.b + c	est libre
a.b	n'est pas libre car a et b sont des facteurs algébriques triviaux.

Remarque : un monôme n'est pas une expression libre.

Définition 2.5: Noyaux algébriques

K est un noyau algébrique d'une expression polynomiale F si K est le quotient de la division algébrique de F par C, où C est un monôme et K est une expression polynomiale libre, C est appelé *conoyau* de K.

On note K(F) l'ensemble des noyaux algébriques de F. On définit le degré d'un noyau de la façon suivante :

– Un noyau K est dit un noyau de degré 0 s'il n'accepte comme noyau que lui-même.

– Un noyau K est de degré n s'il a au moins un noyau de degré (n-1) mais aucun noyau de degré supérieur ou égal à n excepté lui-même.

Ainsi, l'ensemble K(F) des noyaux algébriques de F peut être partitionné et ordonné en sous-ensembles $K_i(F)$, où i représente le degré du noyau. On obtient ainsi la partition suivante des noyaux de F :

$$\{K_0(F), K_1(F), \ldots\ldots K_{n-1}(F), K_n(F)\} = K(F)$$

Remarques :

– On appelle K *noyau global* s'il est noyau de plusieurs fonctions Booléennes ; un noyau local est noyau d'une seule fonction Booléenne.

– Un noyau local peut avoir plusieurs conoyaux associés.

– On peut remarquer qu'un noyau est formé de plus d'un monôme car un monôme n'est pas une expression libre.

– Si F est une expression polynomiale libre, elle est elle-même un noyau algébrique et son conoyau associé est égal à 1.

Exemple :

Soit F_1 une fonction dont l'expression polynomiale est :

$$F_1 = a.b.c + a.b.d + a.e.f + g$$

Les couples conoyaux/noyaux de F_1 sont :

{(a.b, c + d), (a, b.c + b.d + e.f)}

Soit F_2 = a.c + a.d + g.h

Le couple conoyau/noyau de F_2 est : {(a, c + d)}

On remarque donc que (c + d) est un noyau global de degré 0, ses conoyaux sont a.b (F_1) et a (F_2). Par contre, le noyau (b.c + b.d + e.f) est un noyau local de degré 1, son conoyau est a (F_1).

Définition 2.6: Portée d'un noyau

La portée d'un noyau K d'une expression de $F = \sum_{i=1}^{n} m_i$ est définie par le triplet (C,K, ξ) où :

K est un noyau de l'expression F.

C est le conoyau associé au noyau K.

ξ est l'ensemble des monômes résultat du développement de C.K.

Exemple :

F = a.b.c + a.b.d + a.c.d + e.f

K = (c + d)	est un noyau de l'expression de F
C = a.b	est le conoyau associé au noyau K
ξ = {a.b.c, a.b.d}	est l'ensemble des monômes de C.K écrit sous forme polynomiale.

Le triplet (C,K, ξ) définit la portée dans F de K.

Définition 2.7: Compatibilité algébrique

Soient deux couples de conoyaux/noyaux (C,K) et (C',K') d'une expression Booléenne F. Soit E (resp. E') l'ensemble des monômes de C.K (resp. C'.K') écrit sous forme polynomiale.

On dit que (C,K) et (C',K') sont compatibles algébriquement si et seulement si :

$$E \subset E' \text{ ou } E' \subset E \text{ ou } E \cap E' = \emptyset$$

Exemple :

F = a.b.c + a.b.d + a.c.d + e.f

K = (c + d) est un noyau de l'expression de F

C = a.b est le conoyau associé au noyau K

K' = (b + c) est un noyau de l'expression de F

C' = a.d est le conoyau associé au noyau K'

E = {a.b.c, a.b.d}

E' = {a.b.d, a.c.d}

$E \not\subset E'$ et $E' \not\subset E$ et $E \cap E' = \{a.b.d\}$

(C,K) et (C',K') sont incompatibles algébriquement.

2.4 Expressions lexicographiques de fonctions Booléennes

Définition 2.8: Factorisation élémentaire

Une factorisation élémentaire est définie par un couple (C,K) où C est un conoyau et K est un noyau.

Définition 2.9: Relation de précédence induite par une factorisation élémentaire

Une factorisation élémentaire définie par (C,K) induit une relation de précédence dans laquelle les variables du conoyau précédent celles du noyau.

Si V(C) (resp. V(K)) désigne l'ensemble des variables de C (resp. K), alors V(C) précède V(K) que l'on note V(C) « V(K).

Exemple:

La factorisation élémentaire $\bar{a}.c.(b.d+\bar{b}.\bar{d})$ implique la relation de précédence suivante: (a,c) « (b,d).

Remarque:

On appelle *ordre de référence*, noté entre accolade, un ordre total sur les variables d'entrée.

Définition 2.10: Compatibilité d'une factorisation élémentaire avec un ordre de référence donné

Une factorisation élémentaire est *compatible* avec un ordre de référence, si la relation de précédence induite par cette factorisation élémentaire respecte l'ordre de référence.

Exemple:

Dans l'expression $F = \bar{a}.b.c + c.\bar{d} + \bar{b}.\bar{d} + \bar{a}.b.d$, la factorisation élémentaire $\bar{a}.b.(c+d)$ est compatible avec les ordres de référence suivants: {abcd}, {bacd}, {abdc} et {badc}.

Définition 2.11: Factorisations élémentaires lexicographiquement compatibles

Deux factorisations élémentaires (C_1,K_1), (C_2,K_2) sont lexicographiquement compatibles s'il existe au moins un ordre de référence avec lequel elles sont toutes les deux compatibles.

Exemple:

Soient $\bar{a}.b.(c.e + d)$ et $\bar{c}.\bar{e}.(\bar{d} + f)$ deux factorisations élémentaires de l'expression $F = \bar{a}.b.c.e + \bar{c}.\bar{d}.\bar{e} + \bar{b}.\bar{d} + \bar{a}.b.d + \bar{c}.\bar{e}.f$. Elles sont lexicographiquement compatibles car les ordres induits par ces factorisations élémentaires ((a,b) « (c,e,d) et (c,e) « (d,f)) respectent l'ordre de référence {bacedf}.

Théorème 2.1:

Considérons l'expression Booléenne polynomiale d'une fonction F, représentée sous forme de somme de monômes, un ordre de référence et l'ensemble des triplets (C_i,K_i,ξ_i) (cf définition 2.6) compatibles avec cet ordre de référence.

Alors:

a) Si C_i est un facteur algébrique de $C_j = \xi_j \subset \xi_i$.

b) Si C_i n'est pas un facteur algébrique de C_j et si C_j n'est pas un facteur algébrique de $C_i = \xi_j \cap \xi_i = \emptyset$.

Preuve:

a) Supposons que C_i soit un facteur algébrique de C_j, C'est à dire $C_j = q.C_i$.

$\forall m \in \xi_j$, $m = p.C_j = p.q.C_i$ où p est un monôme de K_j

$\Rightarrow m \in \xi_i$

Conclusion: $\xi_j \subset \xi_i$

b) Supposons que C_i ne soit pas un facteur algébrique de C_j et que C_j ne soit pas un facteur algébrique de C_i. Il existe alors deux littéraux a et b tels que:

$a \in V(C_i), a \notin V(C_j), b \in V(C_j)$ et $b \notin V(C_i)$

Supposons qu'il existe un monôme m tel que $m \in \xi_j \cap \xi_i$

$\Rightarrow$ $m = p'.C_i$ où p' est un monôme de K_i

$m = p''.C_j$ où p'' est un monôme de K_j

$a \in V(m)$ et (par hypothèse) $a \notin V(C_j)$

$\Rightarrow a \in p''$ et $a \ll b$ (car $b \in V(C_j)$)

$b \in V(m)$ et (par hypothèse) $b \notin V(C_i)$

$\Rightarrow b \in p'$ et $b \ll a$ (car $a \in V(C_i)$)

Ceci est impossible, donc $\xi_j \cap \xi_i = \emptyset$.

Corollaire 2.1:

Deux factorisations élémentaires lexicographiquement compatibles sont algébriquement compatibles.

Ceci Découle directement de la définition de la compatibilité algébrique (définition 2.7) et du théorème 2.1.

Cette propriété très importante permettra d'éviter le test de la compatibilité algébrique des factorisations élémentaires lexicographiquement compatibles.

Théorème 2.2:

Etant donnés une fonction Booléenne F décrite sous la forme d'une expression Booléenne polynomiale minimisée, un ordre total de référence des variables et l'ensemble des factorisations élémentaires (C_i,K_i) lexicographiquement compatibles avec cet ordre, il existe une unique expression lexicographique obtenue en divisant F par les conoyaux C_i.

Preuve:

Soit l'ensemble des conoyaux/noyaux (C_i,K_i) d'une expression Booléenne polynomiale minimisée F compatibles avec un ordre de référence donné. Cet ensemble est unique et le corollaire du théorème 2.1 nous montre que ces conoyaux/noyaux sont algébriquement compatibles. Par suite, il existe une unique expression factorisée obtenue en divisant algébriquement F par les conoyaux.

2.4.1 Graphe de conoyaux/noyaux

On définit le graphe des conoyaux/noyaux associé à une fonction Booléenne de la façon suivante:

Il s'agit d'un graphe G = <*N*,E> orienté connexe où *N* est l'ensemble des nœuds et E l'ensemble des arcs reliants ces nœuds. A chaque nœud N du graphe est associé le triplet (C,K,ξ). On notera N = (C,K,ξ).

Il existe un arc du nœud N = (C,K,ξ) (K de degré d) au nœud N' = (C',K',ξ') (K' de degré d') si et seulement si, K' est un noyau de l'expression Booléenne de K et d' = d - 1.

Le graphe de conoyaux/noyaux admet une racine qui est représentée par le nœud N_0 = (1,F,M) où F est la fonction Booléenne considérée et M l'ensemble des monômes de F.

Les nœuds feuilles représentent les factorisations élémentaires dont les noyaux sont de degré 0.

On appelle ainsi tout chemin reliant le nœud N_0 à une feuille par un chemin C du graphe.

Exemple:

soit la fonction F suivante:

$$F = \sum_{i=1}^{5} m_i = a.b.g + a.b.d + a.c.d + b.e + e.h$$

le graphe des conoyaux/noyaux de cette fonction est le suivant:

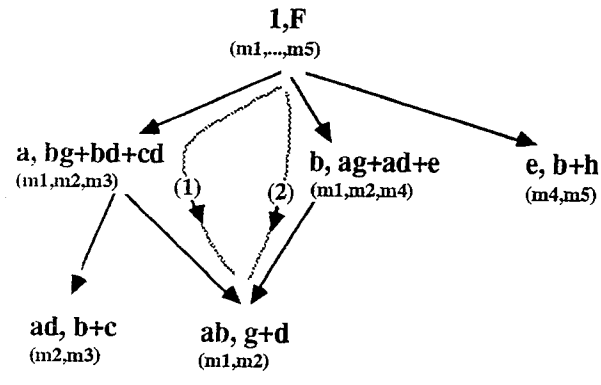


Figure 2.2. Graphe des conoyaux/noyaux de la fonction F

Théorème 2.3:

Tous les noyaux K_i associés aux nœuds N_i d'un même chemin C du graphe sont compatibles algébriquement entre eux.

Preuve:

Soit $N_1, N_2, ..., N_n$ les nœuds d'un chemin C, avec

$N_1 = (C_1,K_1,\xi_1)$, $N_2 = (C_2,K_2,\xi_2)$, ..., $N_n = (C_n,K_n,\xi_n)$, cela veut dire que:

$K_n \subset K_{n-1} \subset ... \subset K_1$ ($K \subset K' \Leftrightarrow$ K' est un noyau de l'expression Booléenne de K et d' = d-1), les noyaux sont donc compatibles algébriquement.

Remarque:

On remarque que si deux chemins C_1 et C_2 convergent vers le même nœud alors leurs noyaux sont incompatibles.

Exemple:

Les chemins (1) et (2) de l'arbre de conoyaux/noyaux de la figure 2.2 ont leurs noyaux incompatibles.

2.4.2 Arbre de conoyaux/noyaux

On définit un arbre de conoyaux/noyaux algébriquement compatibles associé à une expression Booléenne minimisée par un arbre dont:

- Les nœuds N_i sont associés de façon bijective avec un ensemble de factorisations élémentaires algébriquement compatibles caractérisées par les triplets (C_i,K_i,ξ_i),

- Les arcs représentent la relation d'inclusion entre les conoyaux (N_j est un successeur de $N_i \Leftrightarrow V(C_i) \subset V(C_j)$).

Exemple:

Soit F une fonction Booléenne formée de 7 monômes:

$$F = \sum_{i=1}^{7} m_i = \bar{a}.c.e + c.e.\bar{f} + \bar{a}.c.\bar{d} + \bar{a}.\bar{d}.f.\bar{h} + \bar{a}.\bar{e}.d.f + g.h + \bar{a}.g$$

L'ensemble des factorisations élémentaires (C_i,K_i,E_i) de cette fonction est:

$$\{(c, \bar{a}.e + e.\bar{f} + \bar{a}.\bar{d}), (c.e, \bar{a} + \bar{f}), (c.\bar{a}, e + \bar{d}), (\bar{a}, c.e + c.\bar{d} + \bar{d}.f.\bar{h} + \bar{e}.d.f + g), (\bar{a}.\bar{d}, f.\bar{h} + c), (\bar{a}.f, \bar{d}.\bar{h} + \bar{e}.d), (g, \bar{a} + h)\}$$

Il existe 5 sous-ensembles maximaux de factorisations élémentaires compatibles algébriquement:

1 - $\{(c, \bar{a}.e + e.\bar{f} + \bar{a}.\bar{d}), (c.e, \bar{a} + \bar{f}), (\bar{a}.f, \bar{d}.\bar{h} + \bar{e}.d), (g, \bar{a} + h)\}$

2 - $\{(\bar{a}, c.e + c.\bar{d} + \bar{d}.f.\bar{h} + \bar{e}.d.f + g), (c.\bar{a}, e + \bar{d}), (\bar{a}.f, \bar{d}.\bar{h} + \bar{e}.d)\}$

3 - $\{(c, \bar{a}.e + e.\bar{f} + \bar{a}.\bar{d}), (c.\bar{a}, e + \bar{d}), (\bar{a}.f, \bar{d}.\bar{h} + \bar{e}.d), (g, \bar{a} + h)\}$

4 - $\{(\bar{a}, c.e + c.\bar{d} + \bar{d}.f.\bar{h} + \bar{e}.d.f + g), (\bar{a}.\bar{d}, f.\bar{h} + c)\}$

5 - $\{(c.e, \bar{a} + \bar{f}), (\bar{a}.\bar{d}, f.\bar{h} + c), (g, \bar{a} + h)\}$

La figure 1.3 illustre l'ensemble des arbres de conoyaux/noyaux compatibles algébriquement.

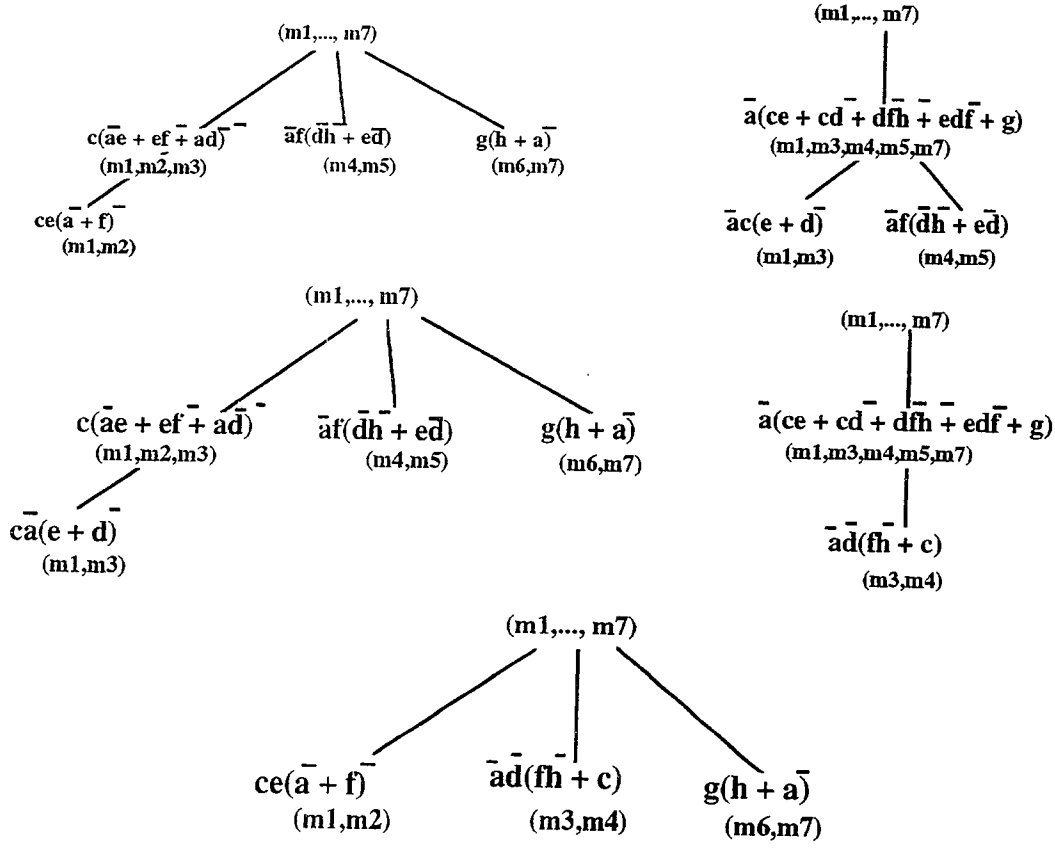


Figure 2.3: Arbres de conoyaux/noyaux algébriques.

Propriété 2.1:

Un nœud (C_i,K_i,ξ_i), d'un chemin dans un arbre de conoyaux/noyaux, est un fils d'un nœud (C_j,K_j,ξ_j) si et seulement si: $V(C_j) \subset V(C_i)$.

Propriété 2.2:

Pour tout chemin dans un arbre de conoyaux/noyaux pris dans le sens de la racine vers les feuilles, un nœud (C_i,K_i,ξ_i) est un fils d'un nœud (C_j,K_j,ξ_j) si et seulement si: $\xi_j \supset \xi_i$

La relation $V(C_j) \subset V(C_i)$ de la propriété précédente montre qu'un monôme contenant $V(C_i)$ contient aussi $V(C_j)$. Ceci implique (théorème 2.1) que $\xi_j \supset \xi_i$.

Propriété 2.3:

Dans un arbre de conoyaux/noyaux algébriquement compatibles, deux nœuds N_i, N_j appartenant à deux chemins distincts de l'arbre sont étiquetés par deux ensembles disjoints de monômes ($\xi_i \cap \xi_j = \emptyset$).

Supposons que cette propriété n'est pas vraie. Il existe alors deux nœuds N_i, N_j appartenant à deux chemins différents d'un arbre et étiquetés par (C_i,K_i,ξ_i), (C_j,K_j,ξ_j), et qui vérifient $\xi_i \cap \xi_j \neq \emptyset$.

$= \xi_j \supset \xi_i$ ou $\xi_i \supset \xi_j$ (théorème 2.1)

Alors N_i est un fils de N_j ou N_j est un fils de N_i. Ceci est en contradiction avec l'hypothèse initiale.

2.4.3 Arbre de conoyaux/noyaux compatibles lexicographiquement

Un arbre de conoyaux/noyaux compatibles lexicographiquement est un arbre tel que l'ensemble de ses conoyaux/noyaux respecte un ordre donné des variables d'entrée. Cet ensemble définit un arbre de conoyaux/noyaux unique et compatible avec l'ordre de référence.

Cette structure de base est fondamentale dans notre algorithme général de factorisation lexicographique. Elle est utilisée en fait pour effectuer une division lexicographique rapide. La première étape de cette division consiste à rechercher l'ensemble des factorisations élémentaires compatibles avec un ordre de référence et le théorème 2.1 nous garantie la compatibilité algébrique. Ce résultat est très important car il permet d'éviter le test, très coûteux en

temps de calcul, de la compatibilité algébrique de l'ensemble des factorisations élémentaires choisies.

La construction de l'arbre de conoyaux/noyaux est faite progressivement en examinant les différentes paires conoyaux/noyaux et en explorant les relations d'inclusion des conoyaux.

Exemple:

Si on considère la fonction suivante F = a.b.c.d + a.b.c.f + a.b.h.k + h.i.j + h.i.f + h.e, l'ensemble des factorisations élémentaires compatibles lexicographiquement est :

{(a.b.c, d + f), (a.b, c.d + c.f + h.k), (h, i.j + i.f+ e), (h.i, j + f)}

• La paire (a.b.c, d + f) est considérée en premier et l'arbre des conoyaux/noyaux est initialisé (figure 2.4a).

• Pour (a.b, c.d + c.f + h.k), le conoyau ab satisfait $1 \subset ab \subset abc$. Par conséquent le nœud correspondant est inséré dans l'arc initial.

• Pour (h, i.j + i.f + e), étant donné que h et abc ne sont pas liés par une relation d'inclusion un nouvel arc est créé en partant de la racine.

• Enfin le conoyau/noyau (h.i, j + f) crée un nouveau nœud dans l'arbre relatif à la relation d'inclusion $h \subset hi$.

L'arbre final est donné en figure 4d.

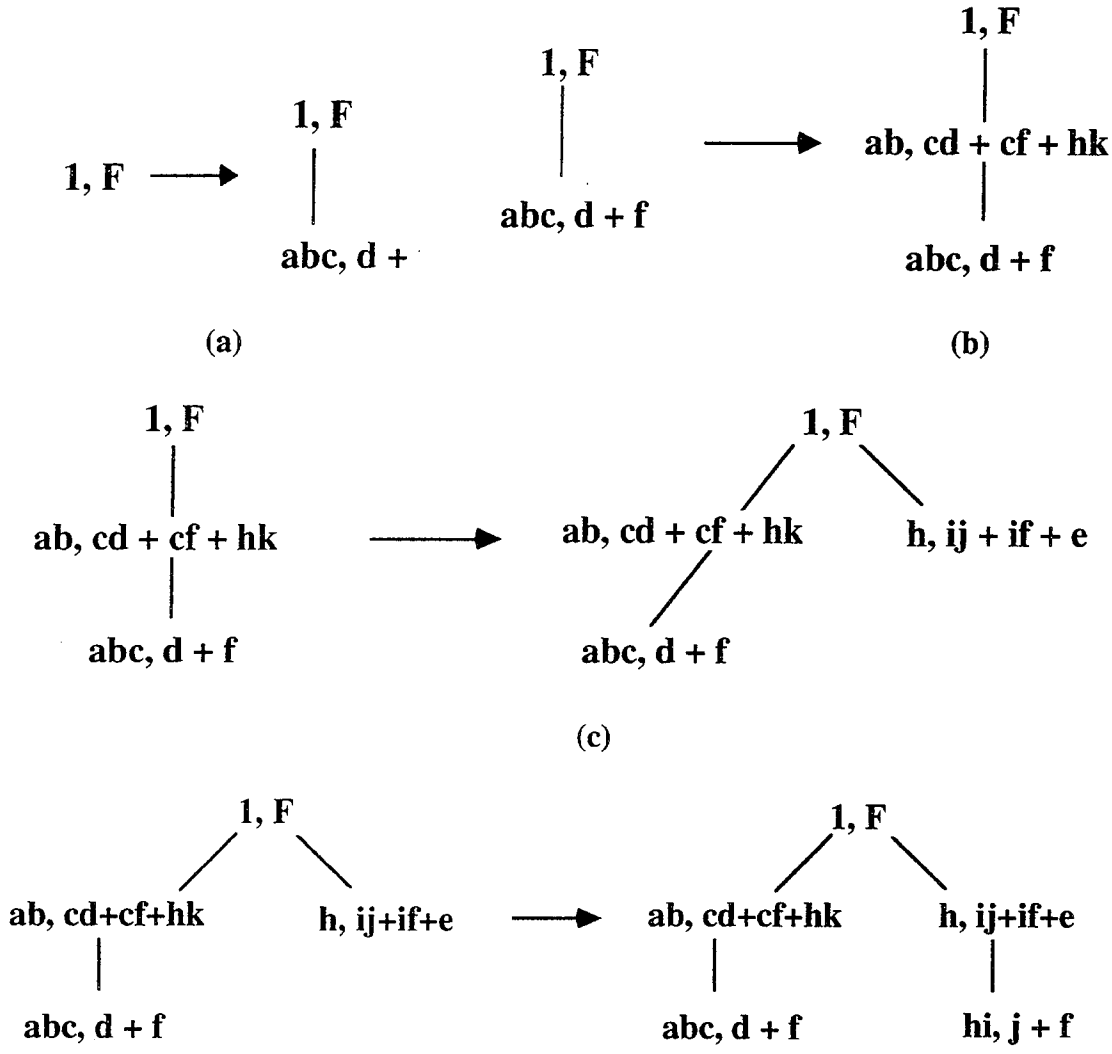


Figure 4. Construction de l'arbre de conoyaux/noyaux

2.5 Matrice de précédence

La factorisation lexicographique d'une expression Booléenne polynomiale consiste à choisir dans un premier temps un ensemble de factorisations élémentaires compatibles lexicographiquement. Cette exigence de compatibilité et les choix successifs des factorisations élémentaires compatibles définissent progressivement un ordre des variables. Pour gérer cette compatibilité, puis mettre à jour la relation de précédence entre les variables, un outil appelé matrice de précédence, a été élaboré.

2.5.1 Définitions et propriétés de la matrice de précédence

Définition 2.12: matrice de précédence

Soit $\rho = \{v_1,...,v_i,...,v_N\}$ l'ensemble des variables d'entrée d'une fonction Booléenne.

Soit B « B' une relation de précédence R sur l'ensemble V des variables d'entrée telle que: $B,B' \subset \rho$ et $B \cap B' = \emptyset$.

On définit la matrice de précédence associée à cette relation R de la façon suivante:

Pr(R) est une matrice de N x N telle que :

- Si v_i précède v_j, alors Pr(R)[i,j] = 1
- Si v_j précède v_i, alors Pr(R)[i,j] = 0

Tous les autres éléments de la matrice prennent la valeur indéterminée (-) "don't Cares". Cette notation indique qu'il n'existe pas de relation de précédence entre ces éléments.

Définition 2.13:

Deux matrices $Pr(R_1)$ et $Pr(R_2)$ sont dites incompatibles s'il existe v_i, v_j ($i \neq j$) tel que $Pr(R_1)[i,j] = 1$ et $Pr(R_2)[i,j] = 0$

Définition 2.14: matrice antisymétrique

Une matrice Pr définie sur l'espace {0, 1, (-)} est dite antisymétrique si et seulement si:

- Pr[i,j] = 0 = Pr[j,i] = 1
- Pr[i,j] = 1 = Pr[j,i] = 0
- Pr[i,j] = (-) = Pr[j,i] = (-)

Propriété 2.4:

La matrice de précédence associée à une relation de précédence R est antisymétrique.

Propriété 2.5:

Tout élément appartenant à la diagonale d'une matrice de précédence prend la valeur (-).

2.5.2 Mise à jour de la matrice de précédence

Considérons une matrice de précédence, contenant les informations sur les relations de précédence induites par les factorisations élémentaires déjà acceptées. Nous notons cette matrice Pr_c, et l'appelons matrice de précédence courante.

Soit une nouvelle factorisation élémentaire compatible avec l'ordre de référence courant et caractérisée par sa relation de précédence R: B « B'. La mise à jour de la matrice de précédence se fait en deux étapes:

• La première étape consiste à créer une matrice intermédiaire, notée ΔPr_c, qui nous permet de mettre à jour les relations de précédence induites par la réunion des relations de précédence correspondant aux matrice Pr_c et Pr(R). Cette matrice ΔPr_c est donnée par:

$$\Delta Pr_c = Pr_c \times Pr(R) + Pr(R) \times Pr_c \qquad (2.1)$$

où la multiplication et l'addition de matrices utilisent les règles de calcul suivantes:

$0 \times 0 = 0$ $\quad$ $1 \times 1 = 1$ $\quad$ $1 \times 0 = (-)$

$0 + 0 = 0$ $\quad$ $1 + 1 = 1$

Notons que (-) est un élément absorbant pour la multiplication et neutre pour l'addition. Le cas (1 + 0) montre une incompatibilité entre les deux matrices Pr_c et Pr(R).

Remarque:

En fait le calcul de ΔPr_c obtenu par la relation (2.1) peut être simplifié car ΔPr_c est une matrice antisymétrique. Il suffit donc d'effectuer le calcul suivant:

$$\Delta Pr_c = (Pr_c \times Pr(R))^* \qquad (2.2)$$

Où (*) indique l'opération qui consiste à compléter la matrice afin de la rendre antisymétrique. C'est à dire, pour tout élément Pr[i,j] de la matrice de précédence faire:

- Si Pr[i,j] = 0 => Pr[j,i] = 1
- Si Pr[i,j] = 1 => Pr[j,i] = 0
- Si Pr[i,j] = (-) => Pr[j,i] = (-)

• Dans la deuxième étape, il s'agit de calculer la nouvelle matrice de précédence $Pr_c(i+1)$ en considérant toutes les relations de précédence

caractérisées par les matrices $Pr_c(i)$, $Pr(R)$ et ΔPr_c. Nous obtenons la matrice $Pr_c(i+1)$ à l'aide de la formule suivante:

$$Pr_c(i+1) = Pr_c(i) + Pr(R) + \Delta Pr_c \qquad (2.3)$$

Remarque:

La relation (2.3) peut être modifiée si les matrices Pr_c et $Pr(R)$ sont étendues à $\mathbf{Pr'_c}$ et $Pr'(R)$, où Pr' est définie par $Pr' = Pr + Id$, avec Id la matrice identité. Nous obtenons alors la formule suivante:

$$\mathbf{Pr'_c}(i+1) = (\mathbf{Pr'_c}(i) + Pr'(R))^* \qquad (2.4)$$

Afin d'illustrer la technique de mise à jour de la matrice de précédence, nous allons l'appliquer sur un exemple simple:

Exemple:

Etant donnée une fonction Booléenne de huit variables (a,b,c,d,e,f,g,h). Soient trois factorisations élémentaires quelconques induisant trois relations de précédence sur les variables:

(R_1) ab « efg

(R_2) e « fh

(R_3) cd « ah

Supposons de plus que les factorisations élémentaires sont choisies dans l'ordre R_1, R_2, R_3.

• Etape 1: Relation (R_1)

Cette première relation donne la matrice de précédence initiale $Pr_c(1) = Pr(R_1)$ (figure 2.5).

	A	B	C	D	E	F	G	H
A					1	1	1	
B					1	1	1	
C								
D								
E	0	0						
F	0	0						
G	0	0						
H								

Figure 2.5: Matrice de précédence initiale $Pr_c(1)$.

• Etape 2: Relation (R_2)

Nous appliquons ensuite la relation (R_2), dont la matrice de précédence associée est $Pr(R_2)$. Nous calculons ΔPr_c à l'aide de la formule:

$$\Delta Pr_c = (Pr_c(1) \times Pr(R_2))^* \qquad \text{(figure 2.6)}$$

	A	B	C	D	E	F	G	H
A					1	1	1	
B					1	1	1	
C								
D								
E	0	0						
F	0	0						
G	0	0						
H								

	A	B	C	D	E	F	G	H
A								
B								
C								
D								
E						1		1
F					0			
G								
H					0			

	A	B	C	D	E	F	G	H
A						1		1
B						1		1
C								
D								
E								
F	0	0						
G								
H	0	0						

Figure 2.6: Matrice de mise à jour ΔPr_c après l'étape 2.

La formule (2.3) nous donne la matrice de précédence $Pr_c(2)$ (figure 2.7).

	A	B	C	D	E	F	G	H
A					1	1	1	1
B					1	1	1	1
C								
D								
E	0	0				1		1
F	0	0			0			
G	0	0						
H	0	0			0			

Figure 2.7: Matrice de précédence $Pr_c(2)$

• Etape 3: Relation (R_3)

La mise à jour de la matrice de précédence relativement à la relation (R_3) donne la matrice finale suivante:

	A	B	C	D	E	F	G	H
A			0	0	1	1	1	1
B					1	1	1	1
C	1				1	1	1	1
D	1				1	1	1	1
E	0	0	0	0		1		1
F	0	0	0	0	0			
G	0	0	0	0				
H	0	0	0	0	0			

Figure 2.8: Matrice finale

2.5.3 Extraction de l'ordre des variables à partir de la matrice de précédence

La matrice finale représente la relation d'ordre entre les variables d'entrée de la fonction. Cette relation d'ordre peut être partielle ou totale. L'extraction d'un ordre de référence compatible avec la matrice finale se fait de la manière suivante:

- On sélectionne dans la matrice de précédence finale les lignes ne contenant pas de zéros; Les entrées correspondantes constituent le premier sous-ensemble des variables d'entrées.
- On élimine les lignes et les colonnes correspondant à ces variables.
- On réitère le processus pour extraire les sous-ensembles suivants, et jusqu'à ce qu'on détermine le sous-ensemble de variables ne contenant que des zéros ou des (-). Les variables appartenant à ce dernier sous-ensemble apparaissent à la fin de l'ordre lexicographique.

De la matrice de la figure 2.8, on extrait une relation d'ordre partielle entre les variables. Cela donne:

bcd « a « eg « fh

2.6 Division Algébrique et lexicographique respectant un ordre initial des variables d'entrée

Dans certains cas, un ordre sur les variables d'entrée peut être imposé. C'est le cas, par exemple, lorsque les signaux d'entrée sont asynchrones. Dans ce cas, les variables d'entrée ayant des retards plus importants que celui d'autres variables du réseau doivent être considérées comme prioritaires dans le processus de synthèse. Donc, ces variables sont placées le plus près possible des sorties du réseau (figure 2.9). Par conséquent le délai à travers la logique entre ces entrées et les sorties est minimal.

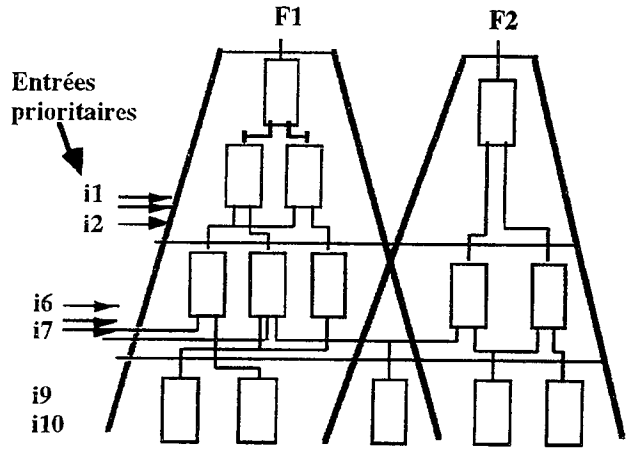


Figure 2.9: Illustration de la structure des fonctions Booléennes avec une priorité sur certaines variables d'entrée.

Nous supposons ici qu'un ordre total sur les variables d'entrée est imposé. Par conséquent une factorisation algébrique et lexicographique doit respecter cet ordre. Dans ce but, nous réalisons une division algébrique en imposant certains critères qui assurent le respect de l'ordre imposé.

Procédure de division

La procédure de division tient donc compte de l'ordre de référence des variables. On effectue la division en considérant successivement les variables en partant de la première dans l'ordre de référence. Les deux littéraux correspondant à la même variable sont pris en compte successivement. Les monômes contenant le littéral courant sont factorisés, tant que cela n'entraîne pas la violation de la relation de précédence induite par l'ordre imposé.

L'algorithme de division, décrit ci-dessous, utilise les notations suivantes:

- V(F) représente le support d'une expression Booléenne F.
- M_{cour} est l'ensemble courant des monômes.
- m_{cour} est le monôme formé par l'ensemble des littéraux qui ont déjà factorisé M_{cour}.
- L est le littéral courant candidat à la factorisation de M_{cour} ou d'un sous-ensemble de M_{cour}.
- $\mathbf{M_L^{init}}$ est l'ensemble des monômes de la fonction Booléenne initiale qui contiennent L.

L'algorithme récursif de division respectant un ordre initial des variables d'entrée est le suivant:

```
Division_Alg_Lex ([M_cour,m_cour], L)
début
    si (L = NULL) retourner;          /* cela signifie qu'il n'y a plus de
                                      littéraux dans la liste ordonnée */
    if ( |M_cour| = 1) retourner;     /* Cardinal de M_cour doit être >1*/
        M_inter = M_cour ∩ M_L^init;
    M_res = M_cour - M_inter;
    Pour chaque monôme m_i tel que m_i ∈ M_inter faire
                                      /* Test de la compatibilité lexicographique*/
    début
    si (la relation (V(m_cour.L) « V(\F(m_i;(m_cour.L)))) n'est pas
    compatible avec l'ordre de référence) alors
     début
        M_inter = M_inter - m_i;      /* mise à jour de M_inter et M_res suivant la
        M_res = M_res + m_i;          compatibilité lexicographique*/
      fin
    fin
    si ( |M_inter| > 1) alors
    début
    si ( |M_res| = 0) alors
     début
        Division_Alg_Lex ([M_cour,m_cour.L], LittéralSuivant(L));
        retourner;
      fin
    BlocGauche = [M_inter,m_cour.L];
    Division_Alg_Lex (BlocGauche, LittéralSuivant(L));
    BlocDroit = [M_res,m_cour.L];
    Division_Alg_Lex (BlocDroit, LittéralSuivant(L));
    FaireArbreDivision ([M_cour,m_cour], BlocGauche , BlocDroit);
    retourner;
    fin
    Division_Alg_Lex ([M_cour,m_cour], LittéralSuivant(L));
    retourner;
fin;
```

L'appel initial de la procédure est:

Division_Alg_Lex ([M_F,1], PremierLittéral)

où:

F est la fonction initiale minimisée.

$\mathbf{M_F}$ est l'ensemble des monômes de F.

1 est le cofacteur de F.

PremierLittéral est le premier littéral de la relation d'ordre totale.

Exemple:

Soit une fonction Booléenne F formée de 6 monômes:

$F = \sum_{i=1}^{6} m_i = a.b.l + a.c.l + a.c.d + a.b.c + b.k + b.c.e$

Les blocs initiaux associés aux littéraux a, b, et c sont:

$\mathbf{M}_{\mathbf{a}}^{\mathbf{init}} = \{m_1, m_2, m_3, m_4\}$

$\mathbf{M}_{\mathbf{b}}^{\mathbf{init}} = \{m_1, m_4, m_5, m_6\}$

$\mathbf{M}_{\mathbf{c}}^{\mathbf{init}} = \{m_2, m_3, m_4, m_6\}$

La procédure de division respectant l'ordre {a,b,c,...} est illustrée ci-dessous, et conduit à l'arbre donné en figure 2.10:

- littéral a

$M_a = M_F \cap \mathbf{M}_{\mathbf{a}}^{\mathbf{init}} = \{m_1, m_2, m_3, m_4\}$ $\quad \mathbf{M}_{\mathbf{res}}^{\mathbf{a}} = M_F - M_a = \{m_5, m_6\}$

- littéral b

$M_{ab} = M_a \cap \mathbf{M}_{\mathbf{b}}^{\mathbf{init}} = \{m_1, m_4\}$ $\quad \mathbf{M}_{\mathbf{res}}^{\mathbf{ab}} = M_a - M_{ab} = \{m_2, m_3\}$

$M_b = \mathbf{M}_{\mathbf{res}}^{\mathbf{a}} \cap \mathbf{M}_{\mathbf{b}}^{\mathbf{init}} = \{m_5, m_6\}$ $\quad \mathbf{M}_{\mathbf{res}}^{\mathbf{b}} = \emptyset$

- littéral c

$M_{abc} = M_{ab} \cap \mathbf{M}_{\mathbf{c}}^{\mathbf{init}} = \{m_4\}$

$M_{ac} = \mathbf{M}_{\mathbf{res}}^{\mathbf{ab}} \cap \mathbf{M}_{\mathbf{c}}^{\mathbf{init}} = \{m_2, m_3\}$ $\quad \mathbf{M}_{\mathbf{res}}^{\mathbf{ac}} = \emptyset$

$M_{bc} = M_b \cap \mathbf{M}_{\mathbf{c}}^{\mathbf{init}} = \{m_6\}$

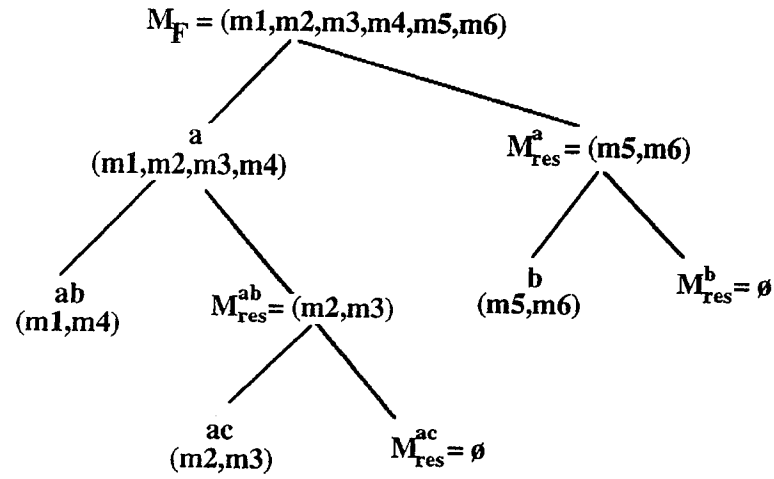


Figure 2.10. Division algébrique et lexicographique respectant l'ordre de référence {abc...}.

L'expression factorisée finale est: F = a[b(c + l) + c(d + l)] + b(c e + k).

2.7 Factorisation lexicographique d'un ensemble de fonctions Booléennes

Le but de cette factorisation est de trouver un ordre des variables d'un ensemble de fonctions Booléennes, ainsi que leur factorisation lexicographique, respectant cet ordre et donnant une solution de coût minimal. Ce coût est exprimé en nombre de littéraux des feuilles de l'ensemble des arbres factorisés, dans lequel un sous-arbre commun n'est compté qu'une seule fois.

Il est évident qu'une recherche exhaustive du meilleur ordre possible n'est pas envisageable. Un tel algorithme ne pourrait pas donner une solution dans un temps raisonnable (solution en N!, N étant le nombre de littéraux). Nous proposons donc, une heuristique, utilisant la notion de gain relatif à une factorisation élémentaire (cf. § 2.7.1).

Partant d'un ensemble de fonctions Booléennes minimisées localement, les différentes étapes de factorisation lexicographique sont:

- Sélection d'un ensemble de factorisations élémentaires compatibles lexicographiquement et construction de l'ordre de référence des variables d'entrée.
- Construction de l'arbre de conoyaux/noyaux et application de la première étape de division rapide.
- Division des restes.
- Recherche de sous-expressions communes.

2.7.1 Calcul du gain d'un candidat diviseur

2.7.1.1 Gain des noyaux algébriques

Soit NbMon(K) le nombre de monômes du noyau K, soit NbLit(C) le nombre de littéraux du conoyau C associé à K et NbLit(K) le nombre de littéraux du noyau K.

1- Calcul de gain pour les noyaux locaux:

Le noyau apparaît dans une seule fonction Booléenne, le gain défini dans le paragraphe 2.2 est donc:

$$Gain(K) = \sum_{i=1}^{m} \left((NbMon(K) - 1).NbLit(C_i)\right) + (m-1).NbLit(K)$$

Où $(C_i)_{i=1..m}$ sont les conoyaux associés au noyau K.

2- Calcul de gain pour les noyaux globaux:

Le noyau K est un noyau de plusieurs fonctions Booléennes. Si ce noyau apparaît p fois dans l'ensemble des fonctions, le gain associé en nombre de littéraux est le suivant:

$$Gain(K) = (p-1).(NbLit(K)) - p$$

La formule générale du gain associé à un noyau K devient:

$$Gain(K)=\sum_{i=1}^{m} \left((NbMon(K)-1).NbLit(C_i)\right)+(m-1).NbLit(K)+(p-1).(NbLit(K))-p$$

Exemple:

$F_1 = a.b.c + a.b.d + a.e.f + g$

$F_2 = a.c + a.d + g.h$

$K_0 = c + d$	noyau global	conoyaux: a.b (F_1) et a (F_2)
$K_1 = b.c + b.d + e.f$	noyau local	conoyau: a (F_1)

$Gain(K_0) = ((2 - 1).2) + (2 - 1).1)) + 1.(2) - 2 = 3$

$Gain(K_1) = (3 - 1).1 = 2$

2.7.1.2 Gain des monômes ou parties de monômes communs

On appelle:

- NbLit(m): le nombre de littéraux du monôme m,
- NbOcc(m): le nombre d'occurrences du monôme m.

La formule du gain associé à un monôme est la suivante:

$$Gain(m) = (NbOcc(m) - 1).(NbLit(m) - 1) - 1$$

Exemple:

$F_1 = a.b.c.d + d\,e + h$

$F_2 = a.b.c.e + d.e + h$

$m_1 = a.b.c$ est une partie commune de monôme dont le gain associé est 1.

$m_2 = d.e$ est un monôme commun, son gain est égal à 0.

2.7.2 Génération de l'ordre des variables et de l'ensemble des factorisations élémentaires compatibles.

En partant d'un ensemble de fonctions Booléennes minimisées localement, si l'ordre des variables d'entrée n'est pas imposé, il est nécessaire de trouver un ordre qui diminue le nombre de littéraux du réseau final.

On peut décrire l'algorithme de construction de l'ordre des variables par l'organigramme suivant. Il faut remarquer qu'à la fin de l'application de cet algorithme, nous possédons un ensemble de factorisations élémentaires compatibles qui serviront directement dans la phase de division.

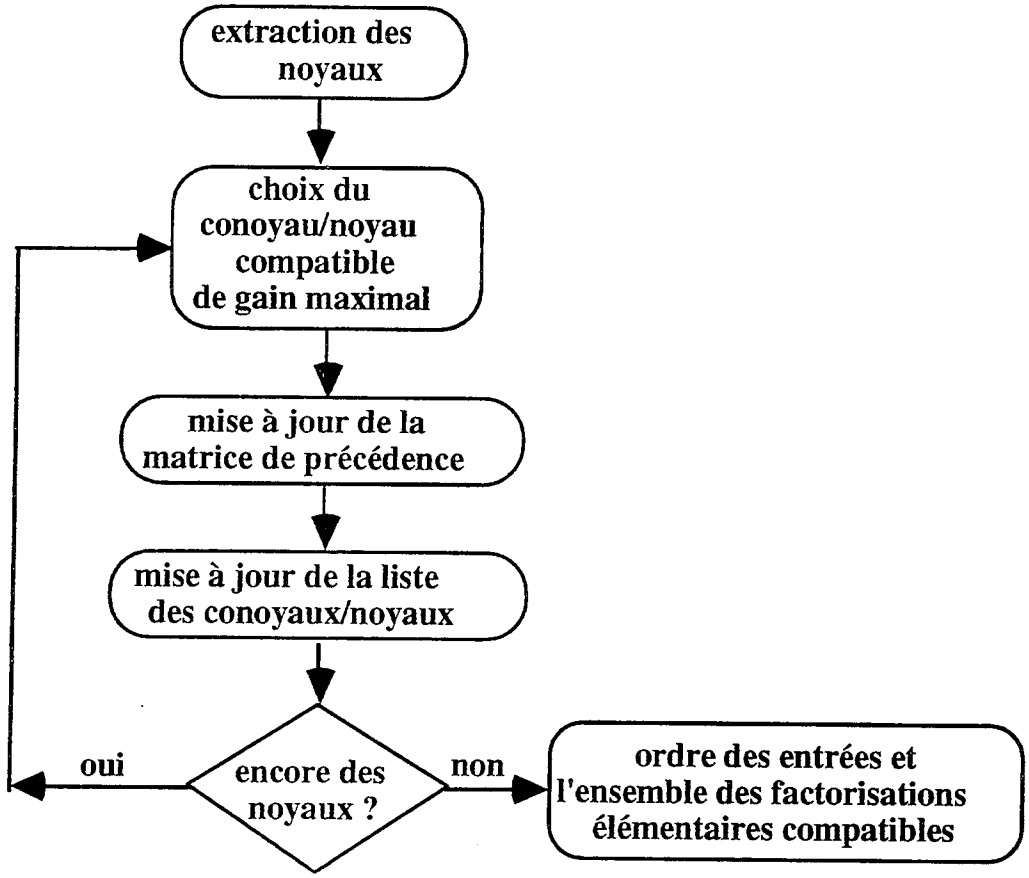


Figure 2.11. Algorithme de sélection des conoyaux/noyaux lexicographiquement compatible.

Cette approche commence donc par l'étape d'extraction des couples conoyaux/noyaux du réseau d'équations Booléennes. Pour déterminer cet ensemble de conoyaux/noyaux nous utilisons l'algorithme décrit dans [Abo94]. A cette étape nous déterminons aussi si un noyau est global ou non (c'est à dire s'il appartient à plusieurs fonctions du réseau).

La sélection des factorisations élémentaires se fait en utilisant un algorithme glouton. La mise à jour de la relation d'ordre utilise les propriétés des matrices de précédence énoncées précédemment.

Une fois la sélection de l'ensemble des factorisations élémentaires compatibles terminée, la matrice de précédence finale permet d'obtenir un ordre partiel ou total des variables d'entrée.

2.7.3 Construction de l'arbre de conoyaux/noyaux et première étape de division

L'étape précédente de sélection des factorisations élémentaires donne un ensemble de factorisations élémentaires (C_i,K_i) compatibles lexicographiquement. Le théorème 2.1 montre que tous les couples (C_i,K_i) sont compatibles algébriquement. Cette approche novatrice très importante nous permet, pour chaque fonction, de construire l'arbre de conoyaux/noyaux sans se préoccuper de la compatibilité algébrique nécessaire dans les approches classiques de factorisation. On a vu également que l'arbre de conoyaux/noyaux est unique pour un ensemble de factorisations élémentaires. Il constitue la structure de base pour l'algorithme de division.

Pour un nœud donné de l'arbre de conoyaux/noyaux, ces successeurs indiquent la division à faire à ce stade qui déterminera explicitement la forme factorisée de ce nœud.

Exemple:

soit la fonction

$$F = \sum_{i=1}^{7} m_i = \bar{a}.c.e + c.e.\bar{f} + \bar{a}.c.\bar{d} + \bar{a}.\bar{d}.f.\bar{h} + \bar{a}\,\bar{e}.d.f + g.h + \bar{a}.g$$

Soit {cagfedh} un ordre de référence donné, l'arbre lexicographique de conoyaux/noyaux est donné par la figure 2.12.

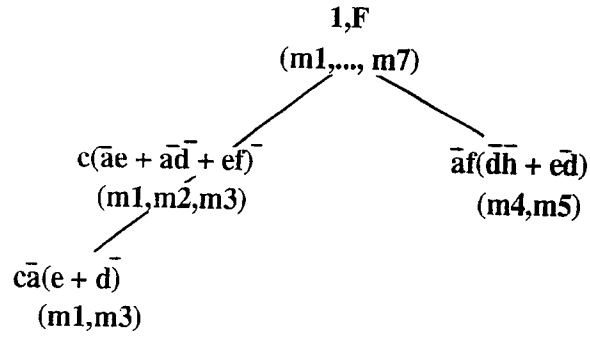


Figure 2.12. Arbre de conoyaux/noyaux lexicographique.

A l'étape 1, la fonction F s'écrit:

$$F = c.(\bar{a}.e + e.\bar{f} + \bar{a}.\bar{d}) + \bar{a}.f.(\bar{d}.\bar{h} + \bar{e}.d) + g.h + \bar{a}.g;$$

les deux monômes m_6 et m_7 non utilisés à cette étape constituent le reste.

A l'étape 2, l'expression finale factorisée de la fonction F est:

$$F = c.[\bar{a}.(e + \bar{d}) + e.\bar{f}] + \bar{a}.f.(\bar{d}.\bar{h} + \bar{e}.d) + g.h + \bar{a}.g;$$

Cette expression finale est représentée sous la forme d'un arbre lexicographique dont les nœuds sont maintenant des opérateurs Booléens classiques OU et ET (figure 2.13).

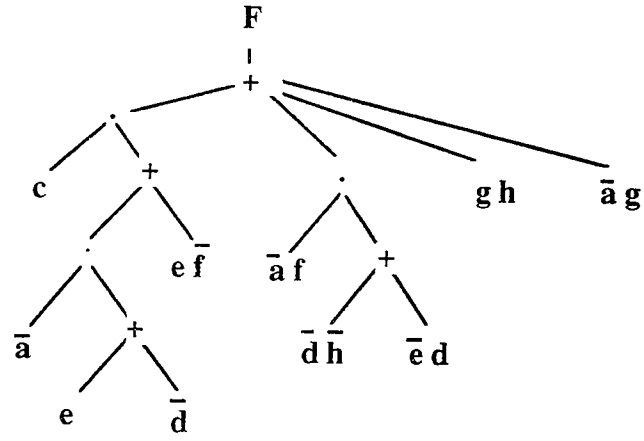


Figure 2.13. Arbre de factorisation lexicographique.

2.7.4 Division des restes

A l'issue de l'étape de construction de l'arbre de conoyaux/noyaux, certain nombre de factorisations reste possible, surtout celles que les fortes contraintes imposées par l'ordre lexicographique ont rejetées, notamment au niveau des restes de la division à chaque nœud dans l'arbre des conoyaux/noyaux. Dans l'exemple précédent les monômes g h, $\bar{a}$ g constituent le reste de la division de la fonction F par les conoyaux c et $\bar{a}$ f, et le monôme $e\bar{f}$ est le reste de la division du noyau de degré 1 ($\bar{a}$ e + $e\bar{f}$ + $\bar{a}$ $\bar{d}$) par le conoyau relatif $\bar{a}$.

Ces restes seront factorisés dans cette étape en respectant évidemment l'ordre des variables d'entrée obtenu à l'issue de l'étape précédente.

Cette étape est composée de deux phases:

• 1ère phase: Extraction de noyaux

Cette première phase utilise le même algorithme de recherche de noyaux décrit précédemment. Elle est appliquée recursivement à l'ensemble des restes de chaque nœud de l'arbre de conoyaux/noyaux. La matrice de précédence déterminée dans l'étape précédente sert de référence, et sa mise à jour est faite après chaque sélection de factorisation élémentaire.

Notons que les noyaux extraits pendant cette étape ne font pas partie de l'ensemble initial des noyaux de la fonction. Pour illustrer ce point, on

considère la fonction F = a.b + a.d + a.f + a.i + i.j.h + i.e. Il existe deux factorisations élémentaires possibles $\{(a, b + d + f + i), (i, a + g.h + e)\}$.

Le gain de la première factorisation élémentaire étant supérieur à celui de la deuxième, on choisit donc cette factorisation qui donne l'expression factorisée suivante: F = a.(b + d + f + i) + i.g.h + i.e; le reste (i.g.h + i.e) peut lui même être factorisé sous la forme i.(g.h + e), sans pour autant violer l'ordre des variables induit par la première factorisation. Le noyau utilisé pour la factorisation de ce reste est une partie du noyau initial (a + g.h + e), et ne peut donc pas être trouvé lors de la première étape de recherche de noyaux.

• 2ème phase: Division lexicographique et algébrique suivant un ordre de référence.

Cette phase utilise l'algorithme de division décrit dans la deuxième partie de ce chapitre. Elle est appliquée sur les sommes de monômes dans l'ensemble des fonctions Booléennes du réseau, et elle permet d'obtenir des factorisations partielles compatibles avec l'ordre des variables.

Exemple:

Soit la fonction

F = a.(b + d + f + i) + i.(g.h + e) + a.c.l.k + c.k.l.m;

La factorisation k.l.c.(a + m), ne peut être réalisée à cause de son incompatibilité avec la première factorisation, a.(c + d + f + i). Par contre la factorisation partielle k.l.(a.c + c.m) est compatible.

2.7.5 Recherche de sous-expressions communes

Cette dernière étape concerne la reconnaissance de sous-expressions communes (autres que les noyaux globaux, qui eux sont repérés et mis en commun pendant la phase de construction de l'arbre de conoyaux/noyaux). Elle peut être considérée comme optionnelle car elle perturbe la régularité du routage obtenue à partir des trois étapes précédentes. Néanmoins, les expériences faites dans ce domaine montrent que c'est une étape primordiale dans le cas où une optimisation en surface est requise. Certains paramètres peuvent être pris en compte pour améliorer la surface du routage dans le circuit final. C'est le cas, par exemple, du filtre appliqué sur certaines sous-fonctions, en fonction de leur taille (nombre de littéraux admis dans l'expression de la sous-fonction), qui permet d'éviter la création d'un très grand ensemble de

sous-fonctions de faibles tailles dans le réseau final. Les expériences montrent que la taille de ces sous-fonctions doit être limitée à trois littéraux et cela pour des circuits de moyenne et haute complexité. Les résultats présentés dans ce chapitre tiennent compte de cette dernière étape ainsi que du critère de filtrage précédent.

Cette étape, comme l'étape précédente, est composée de deux phases, la première est une phase de prétraitement de certains nœuds ayant comme successeurs, des feuilles étiquetées par un simple littéral. La deuxième est la phase d'extraction de monômes ou sous-monômes communs dans le réseau de fonctions.

- Phase 1: Prétraitement des feuilles de l'arbre étiquetées par un seul littéral

Pour chaque nœud N_i de l'arbre factorisé dont l'opérateur Booléen est un opérateur OU, nous partitionnons les successeurs du nœud N_i en deux blocs, le bloc A contient les feuilles étiquetées par un seul littéral (l_1^i , ..., l_k^i), le bloc B contient les feuilles étiquetées par un monôme (m_1^i , ..., m_n^i).

Pour chaque nœud N_i ayant k (k > 1) successeurs appartenant au bloc I, nous effectuons la transformation suivante: Nous transformons la somme de littéraux ($l_1^i + ... + l_k^i$) appartenant au bloc A en un pseudo-monôme $m_0^i = \overline{(\overline{l_1^i} \cdots \overline{l_k^i})}$. Nous rajoutons ce pseudo-monôme à l'ensemble des monômes et nous insérerons un inverseur dans l'arbre factorisé entre le nœud Ni et la nouvelle feuille étiquetée par ce nouveau pseudo-monôme.

- Phase 2: Extraction des monômes et sous-monômes communs.

La liste de monômes est maintenant établie, elle comprend les monômes initiaux et ceux créés lors de la phase 1. L'algorithme d'extraction de monômes et sous-monômes communs décrit dans le chapitre I est alors appliqué, avec un calcul classique du gain; ces monômes ou sous-monômes deviennent alors une sous-fonction.

A la fin de cette étape, les inverseurs créés pendant la phase 1 seront propagés jusqu'aux feuilles et les nouvelles sous-fonctions sont rajoutées à l'ensemble des fonctions du réseau.

Ces deux phases sont itérées successivement tant qu'il y a des monômes ou sous-monômes communs.

Pour illustrer cet algorithme, considérons les deux nœuds OU de la figure 2.14-a. La figure 2.14-b montre la transformation des feuilles étiquetées par un seul littéral en monômes. La figure 2.14-c est le résultat de l'algorithme de recherche de sous-expressions communes, et la figure 2.14-d donne le résultat final après propagation des inverseurs vers les feuilles.

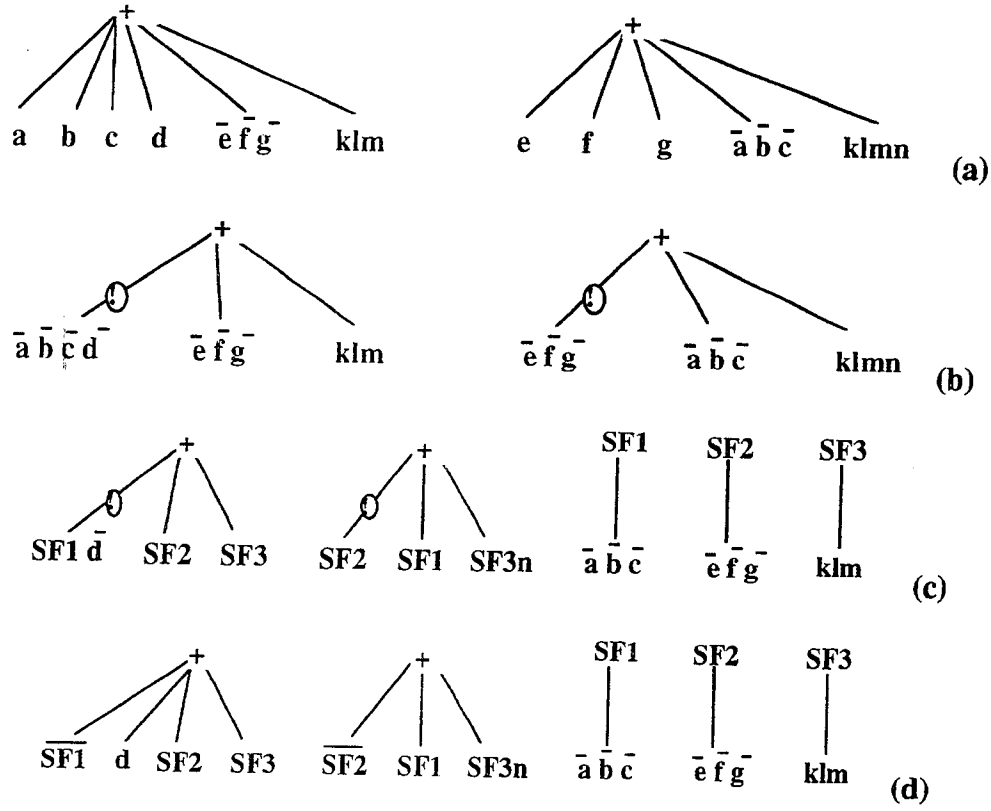


Figure 2.14. Illustration de la recherche de sous-expressions communes.

2.8 Conclusion

La méthode de factorisation lexicographique [Abo92] [Abo93] repose sur l'ordonnancement des variables était une idée extrêmement originale et on peut la considérer comme un précurseur des représentations structurées comme les OBDD (*Ordered Binary Decision Diagram*) qui ont été réutilisées en synthèse un peu plus tard. L'avantage est la réduction de complexité de la connectique. Ceci est non seulement précieux pour une cible telle que Xilinx mais est redevenue de première importance pour les technologies submicroniques pour lesquelles les connexions sont coûteuses en performance et en dissipation de puissance.

Chapitre III

Décomposition technologique

3.1 Introduction

Nous avons vu dans le premier chapitre que les FPGA à base de LUT étudiés ici sont organisés en cellules contenant plusieurs LUT et des points mémoires. La décomposition technologique sur ce type de FPGA peut être divisée en deux phases : la première phase consiste à décomposer le réseau Booléen en sous fonctions Booléennes qui peuvent être implantées sur un LUT (fonction réalisable). La deuxième phase consiste à déterminer l'ensemble de fonctions réalisables qui seront placées dans une même cellule (phase de regroupement). Pour certaines technologies, la phase de regroupement doit prendre en compte les points mémoire pour minimiser le nombre de cellules.

De façon théorique, dans la phase de décomposition, il s'agit de résoudre le problème de la décomposition d'une fonction Booléenne en sous fonctions d'un nombre limité de k variables. Pour certaines cibles, nous verrons que ce nombre k peut être variable. Pour faciliter la phase de regroupement, il est important de favoriser le partage des variables entre ces sous fonctions.

Après une présentation de l'état de l'art de la décomposition technologique sur FPGA à base de LUT, nous nous intéressons d'abord au problème de la décomposition d'une fonction en sous fonctions d'un nombre limité de variables en utilisant comme support la représentation lexicographique. On verra ensuite que le partage des entrées souhaité pour le regroupement des fonctions dans une cellule, sera facilité par la représentation lexicographique.

3.2 Les nouvelles approches spécifiques à la cible

La cellule de base des réseaux programmables de type LUT peut implanter n'importe quelle fonction de k variables soit 2^{2^k} fonctions. L'utilisation des approches classiques de synthèse, fondées sur une bibliothèque de portes, est impossible pour des valeurs de $k > 3$ [Mur90]. Des méthodes de décompositions spécifiques à cette technologie ont donc été développées.

3.2.1 Bin packing

L'approche utilisée dans *Chortle-crf* [Fra90] [Fra91a] [Fra91b] s'inspire de la décomposition technologique fondée sur une librairie introduite par DAGON [Keu87]. Le système initial est partitionné en une forêt d'arbres puis chaque arbre est décomposé en sous-circuits de k entrées. Le circuit final est obtenu par l'assemblage de ces sous circuits.

Pour la première phase de décomposition des arbres, Chortle-crf utilise un algorithme de « bin-packing » qui fait simultanément la décomposition et le « matching » en utilisant la programmation dynamique pour assurer la minimisation de la décomposition du noeud courant.

3.2.2 Décomposition de Roth-Karp

La décomposition technologique dans *Mis-pga* [Mur90] [Mur91a] [Mur91b] [Mur94] minimise le nombre de cellules en deux phases. La première phase consiste à décomposer le système initial pour que chaque noeud du système soit réalisable par une cellule. La seconde phase utilise une heuristique de couverture minimale pour la réduction du nombre de cellules dans le circuit final.

Dans la première phase, quatre approches sont utilisées pour décomposer les noeuds qui ne peuvent être implantés sur une cellule. La première approche est basée sur la décomposition de Roth-Karp [Rot62], la seconde est fondée sur l'extraction de noyaux [Bra92], la troisième est fondée sur le « bin-packing » comme dans Chortle-crf et la dernière est fondée sur la cofactorisation basée sur le théorème de Shannon.

Pour la solution optimisée chemin critique, Mis-pga utilise un arbre décomposé en réduisant la profondeur et dont tous les noeuds ont une entrance de deux. L'arbre est parcouru en partant des feuilles (entrées primaires) vers la racine (sortie primaire). Chaque noeud de profondeur p est réinjecté dans le noeud successeur de profondeur $p+1$, si ce noeud successeur est toujours réalisable sur une cellule après réinjection.

3.2.3 Décomposition de graphe ITE

Rappelons qu'un graphe ITE (« *if-then-else* ») d'une fonction Booléenne F est un graphe dirigé sans cycle où chaque noeud possède trois fils. Le premier fils représente la fonction de décision (partie *if*) alors que les deux autres correspondent à la fonction F suivant que la fonction de décision prend la valeur 1 (partie *then*) ou 0 (partie *else*).

Soit F une fonction Booléenne, celle-ci peut s'exprimer :

$F = \overline{X}.F(X=0)+X.F(X=1)$ où X est une fonction Booléenne arbitraire. = ITE (X, F(X=1), F(X=0))

La décomposition technologique dans *Xmap* [Kar91] utilise deux passes pour minimiser le nombre de cellules nécessaires pour implanter un réseau Booléen. La première passe décompose les noeuds du réseau Booléen initial représenté sous forme de graphe ITE.

Dans un graphe ITE, chaque noeud représente une fonction d'un multiplexeur 2 vers 1. Chaque noeud peut donc être réalisé par un seul k-LUT pour k supérieur ou égal à 3.

Une seconde passe parcourt le réseau ainsi décomposé en partant des entrées primaires et marque les noeud qui seront réalisés par une cellule.

3.2.4 Décomposition disjonctive

L'approche de *Hydra* [Fil91] peut être décrite par les deux tâches suivantes : premièrement un réseau Booléen réalisable est obtenu en utilisant deux techniques de décomposition : une décomposition disjonctive et une décomposition ET-OU.

Rappelons qu'une décomposition disjonctive d'une fonction f(S) consiste à écrire f sous la forme :

$f = g\,(h(S^A),S^B)$ ou $S^A,S^B \subseteq S$ et $S^A \cup S^B = S$ et $S^A \cap S^B = \varnothing$.

On dit que g est une décomposition disjonctive de f

L'application de la décomposition disjonctive au DAG n'aboutit pas à des fonctions réalisables. La décomposition ET-OU est appliquée aux fonctions de plus de k variables pour aboutir à un réseau réalisable. Pour faciliter le choix des partitions (S^A,S^B), une première décomposition ET-OU est appliquée aux fonctions qui ont un large support puis la décomposition disjonctive est appliquée au réseau résultant.

La seconde tâche optimise le nombre de CLBs en appliquant une heuristique de regroupement pour déterminer le placement des noeuds réalisables dans les CLBs. Pour déterminer le meilleur regroupement, les auteurs utilisent une fonction coût basée sur le nombre des entrées partagées par les fonctions.

3.2.5 Autres méthodes

D'autres approches ont été proposées dans la littérature mais elles présentent beaucoup de points communs avec les méthodes présentées ci-dessus. Nous citerons :

Aloe-CLB [Dre91] : utilise la décomposition d'Ashenhurt et une décomposition fondée sur les multiplexeurs pour aboutir à un réseau de noeuds réalisables. Les noeuds obtenus sont regroupés par un algorithme « glouton » fondé sur le nombre des variables partagées par les sous fonctions.

FGmap [Yun93] : cette approche est fondée sur la décomposition disjonctive. Les auteurs utilisent la représentation des fonctions Booléennes sous forme de BDD. La phase de regroupement est basée sur la recherche de motifs. Pour Xilinx série 4000, les auteurs ont défini neuf motifs sans considérer les éléments de mémorisation.

NODECA [He93] : les auteurs présentent une approche fondée sur une décomposition ET/OU suivie par une décomposition simple. Il donnent la priorité à la décomposition disjonctive par rapport à la décomposition non disjonctive en utilisant une fonction coût de mesure de « dispersion » pour détecter la possibilité d'une décomposition disjonctive. L'algorithme présenté est exponentiel.

Rmap [Mar94] : Les auteurs s'intéressent essentiellement aux problèmes de routage. Ils présentent une méthode assez proche de celle de *Hydra*, fondée sur la décomposition et le regroupement en une même phase. le graphe initial est un graphe ET/OU dont tous les noeuds ont une entrance < k (k nombre des entrées du LUT). Le choix des noeuds à inclure dans un CLB est déterminé par une fonction coût basée sur le nombre des arcs couverts par le CLB (interne au CLB) et le nombre des entrées et des sorties de ce dernier ; ce qui assure la minimisation du nombre des signaux et des connexions à router.

3.3. Décomposition d'une fonction en sous fonctions dépendant d'un nombre fixe de variables

Dans un premier temps, on considère un problème simplifié qui est la recherche d'une sous fonction d'une fonction dépendant d'un nombre limité de variables.

Définition 3.3.1 : Chaîne lexicographique

On appelle chaîne lexicographique une séquence d'entrées I=(I1,I2, ...,In) correspondant à l'ordre de référence lexicographique dans l'ordre de précedence inverse.

Exemple : Soit $F = a(b(c + \bar{f}) + c(d + \bar{g})) + b(c\ e + k)$ une fonction factorisée. L'ordre de référence est (a b c d e f g k). La chaîne lexicographique est alors (k,g,f,e,d,c,b,a)

Définition 3.3.2 : sous-chaîne lexicographique

Une sous-chaîne lexicographique est une sous chaîne d'entrées successives dans la chaîne lexicographique.

Définition 3.3.3 : sous-chaîne lexicographique terminale

Une sous-chaîne lexicographique terminale est une sous-chaîne qui contient la dernière variable d'entrée (In).

Exemple : Soit la chaîne lexicographique I=(k,g,f,e,d,c,b,a). La chaîne S=(c,b,a) est une sous-chaîne lexicographique terminale de trois variables.

Définition 3.3.4 : Insertion de point de coupure

Soit une sous-chaîne terminale, soit SI le sous-ensemble d'entrées correspondant. L'insertion de points de coupure dans l'arbre lexicographique relative à cette sous-chaîne lexicographique terminale consiste à diviser l'arbre factorisée en deux parties, une partie dépendante des variables de SI noté A(SI) et l'autre dépendante des variables de I-SI noté A(I-SI).

La partie dépendante de SI est appelée le sous-arbre associé à SI. le sous ensemble d'entrées SI est appelé support de coupure.

Si un point de coupure est inséré sur un arc, le noeud originel de l'arc est appelé noeud de coupure.

L'insertion de points de coupure a pour objet de transformer l'arbre initial en un nouvel arbre en introduisant des noeuds fictifs sur les arcs de l'arbre initiale. Le nouvel arbre est caractérisé par la propriété suivante :

- Les noeuds successeurs de chaque noeud appartiennent soit à A(SI) soit à A(I-SI).

Exemple :

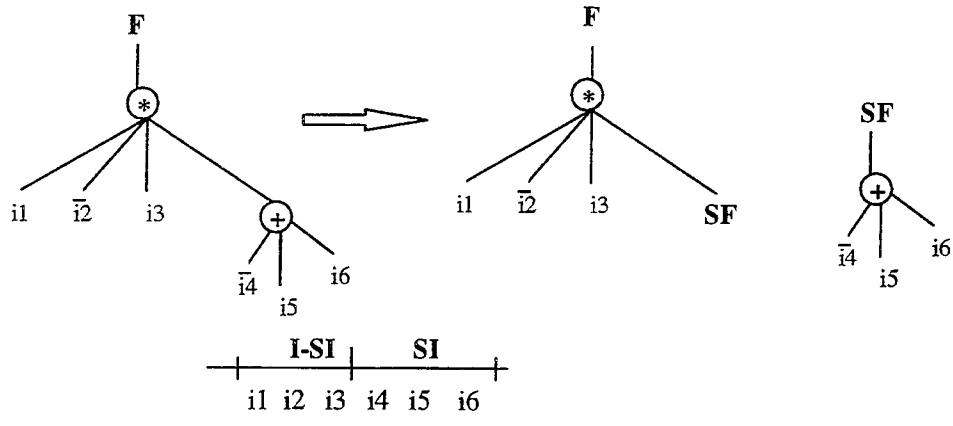


Figure 3.1 : Insertion de point de coupure.

Définition 3.3.5 : Expansion d'un noeud

Pour que l'arbre obtenu par l'insertion des points de coupure vérifie la propriété précédente, il est nécessaire de réaliser l'expansion de certains noeuds.

L'expansion d'un noeud étiqueté par OU (ET) consiste à remplacer le noeud par deux noeuds étiquetés par le même opérateur et liés par un arc. Cet arc est appelé arc d'expansion.

Exemple :

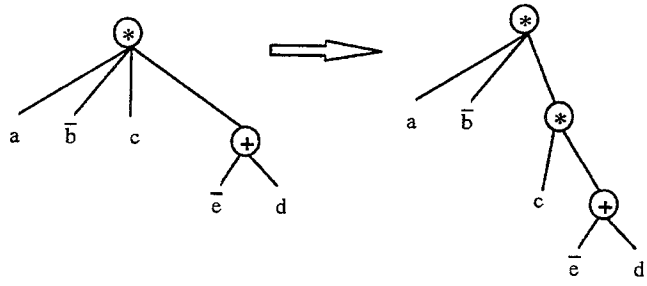


Figure 3.1 : Expansion d'un noeud.

- **Procédure d'insertion de points de coupure**

L'algorithme d'insertion de points de coupure commence par les feuilles, en appliquant recursivement les règles suivantes :

- Si un noeud a tous ses successeurs dans SI (resp I-SI) il est associé à Si (resp. I-SI).
- Si un noeud a un seul successeur associé à SI et a un ou plusieurs noeuds associés à I-SI, un point de coupure est alors inséré sur l'arc qui relie ce noeud avec son successeur associé à SI (Figure 3.2-a).
- Si un noeud a plusieurs successeurs associés à SI et a au moins un successeur associé à (I-SI). Il est remplacé par deux noeuds par le processus d'expansion décrit ci-dessus. Un point de coupure est inséré sur l'arc d'expansion (Figure 3.2-b).

Exemple :

Considérons l'exemple de la figure 3.2, si nous considérons que I={I2, I4, I1, I3}, Si={I1, I3} alors I-SI={I2, I4} ; deux différents cas de point de coupure sont illustrés par les figures 3.2-a et 3.2-b.

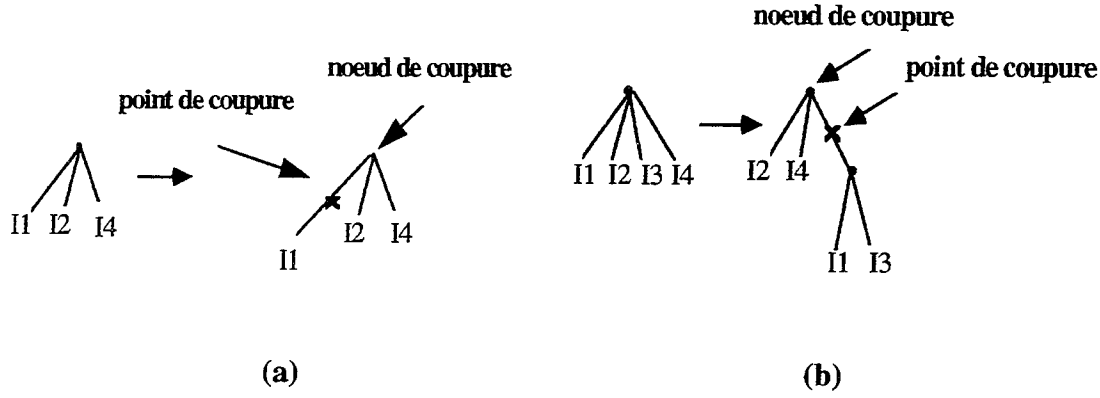


Figure 3.2 : procédure d'insertion de point de coupure

Théorème 3.3.1 :

Soit l'arbre lexicographique correspondant à la factorisation algébrique $F=P.Q+R=P.(Q1+Q2+...+Qn)+R$. F est représentée par un arbre de quatre niveaux. Le noeud du niveau supérieur est un noeud ET correspondant aux monômes (Figure 3.3).

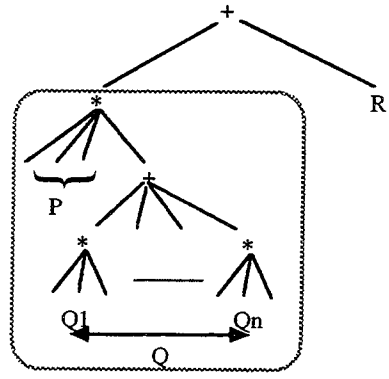


Figure 3.3 : arbre lexicographique.

La procédure d'insertion de point de coupure est recursivement appliquée sur les fils du noeud OU du niveau supérieur.

L'insertion ascendante de point de coupure relative à une chaîne terminale SI peut générer deux types de noeuds de coupure. les deux cas sont exclusifs :

Type 1 : Le noeud ET supérieur devient un noeud de coupure.

Type 2 : Quelques noeuds ou tous les noeuds Et du niveau inférieur et le noeud OU, deviennent des noeuds de coupure.

Preuve :

La séparation des variables de I-SI et SI peut être divisée en quatre cas illustrés par la figure 3. :

Cas 1 : Aucun point de coupure n'est inséré.

Cas 2 : Les littéraux de P appartiennent à I-SI et les littéraux de Q appartiennent à SI ; le noeud ET supérieur est assigné à I-SI ; le noeud OU est assigné à SI. Un point de coupure est inséré entre le noeud OU et le noeud supérieur ET (Figure 3.5-a).

Cas 3 : Les littéraux de P appartiennent à SI et I-SI et les littéraux de Q appartiennent à SI ; dans ce cas le noeud supérieur ET est un noeud de coupure ; il est développé en deux noeuds ET. Le noeud supérieur ET est assigné à I-SI et le noeud ET du bas est assigné à SI. On insère un point de coupure sur l'arc qui les relie (Figure 3.5-b).

Cas 4 : Tous les littéraux de P appartiennent à I-SI et les littéraux de Q appartiennent à SI et à I-SI. Le monôme de Q dont les littéraux qui appartiennent à I-SI et à SI, sont appelés monômes critiques. Les autres monômes ont leurs littéraux dans SI ou dans I-SI. Les noeuds ET qui génèrent des monômes critiques sont des noeuds de coupure. On réalise l'expansion du noeud et un point de coupure est inséré entre les deux noeuds ET résultants. Le noeud OU est aussi un noeud de coupure. On réalise l'expansion du noeud et un point de coupure est inséré sur l'arc qui lie les deux noeuds OU résultants. Ce cas est illustré par la figure 3.5-c. Le nombre de points de coupure est égal au nombre de monômes critiques plus un.

Le cas qui ne peut se produire est celui où les littéraux de P et Q appartiennent à SI et I-SI. Il est impossible puisque les littéraux de P précèdent ceux de Q. Ceci est très important car ce cas, qui ne peut pas se produire dans un arbre lexicographique, génère un nombre de points de coupure plus important que les quatre cas cités.

Le cas 1 est considéré le meilleur car il ne nécessite pas de développement.

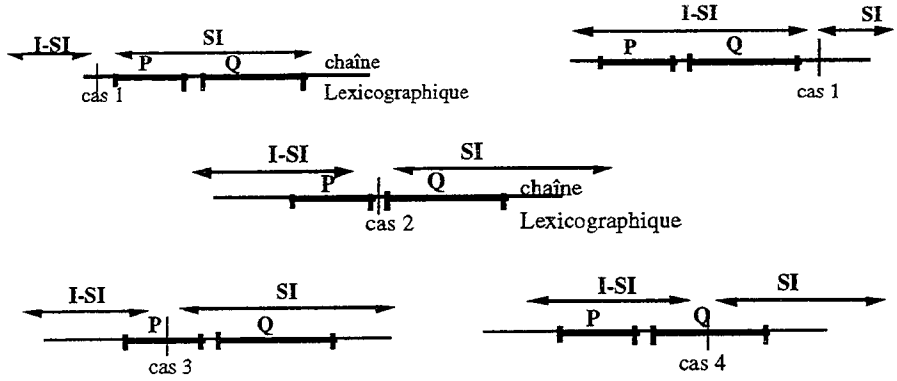
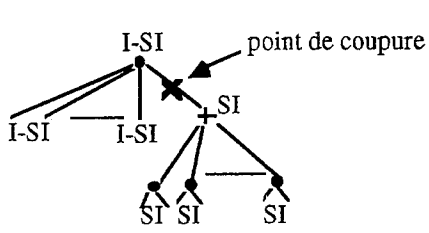
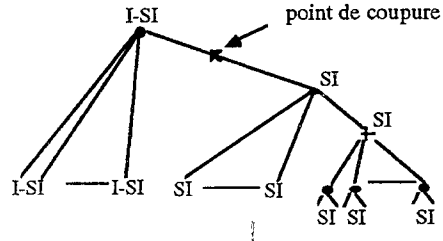


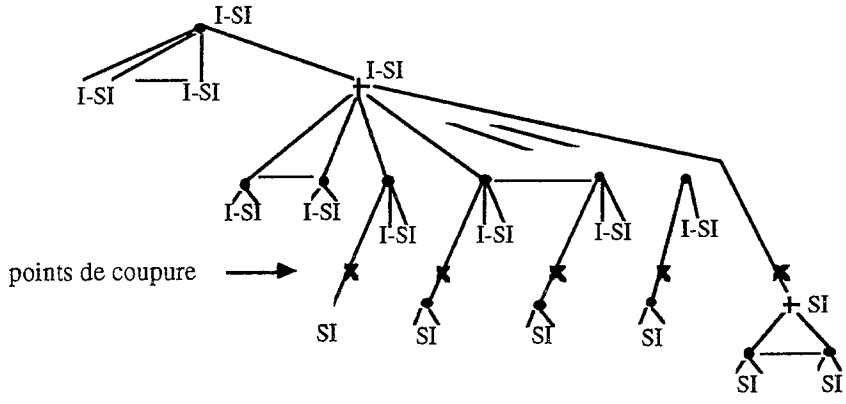
Figure 3.3. Point de coupure dans la chaîne lexicographique



(a) cas 2



(b) cas 3



(c) cas 4

Figure 3.5 : Insertion de points de coupure

Définition 3.3.6 : Chemin complet

Un chemin complet est un chemin qui relie la racine à une feuille de l'arbre lexicographique.

Remarque :

D'après les caractéristiques de la factorisation lexicographique, l'ordre des variables rencontrées sur un chemin complet de l'arbre factorisé correspond à l'ordre de référence.

Théorème 3.3.2 :

Dans un arbre lexicographique, l'insertion ascendante de points de coupure relative à une chaîne terminale introduit au plus un point de coupure sur chaque chemin complet.

Preuve : Un chemin complet commence par la racine, passe successivement par les noeuds ET des conoyaux feuilles et des noeuds OU correspondants à la somme des monômes. Nous avons mentionné qu'une séquence d'un sous-ensemble de variables d'entrées respectant l'ordre lexicographique, est associée à un tel chemin complet. La définition d'une chaîne terminale implique la coupure de cette séquence en un seul point, dans un sous-ensemble ou entre deux sous-ensembles successifs. Dans le premier cas, le noeud correspondant au sous-ensemble sera un noeud de coupure et il nécessitera un développement et l'insertion d'un point de coupure. Dans le second cas, un point de coupure sera inséré sur l'arc qui relie les deux noeuds correspondants aux deux sous-ensembles successifs.

Dans l'exemple de la figure 3.6 qui correspond à la fonction F avec l'ordre lexicographique (j f d h a c b i e g), la chaîne terminale SI=(c b i e g) introduit la création de six points de coupure (C1, ..., C6). Les points de coupure C1, C4, C5 et C6 nécessitent un développement.

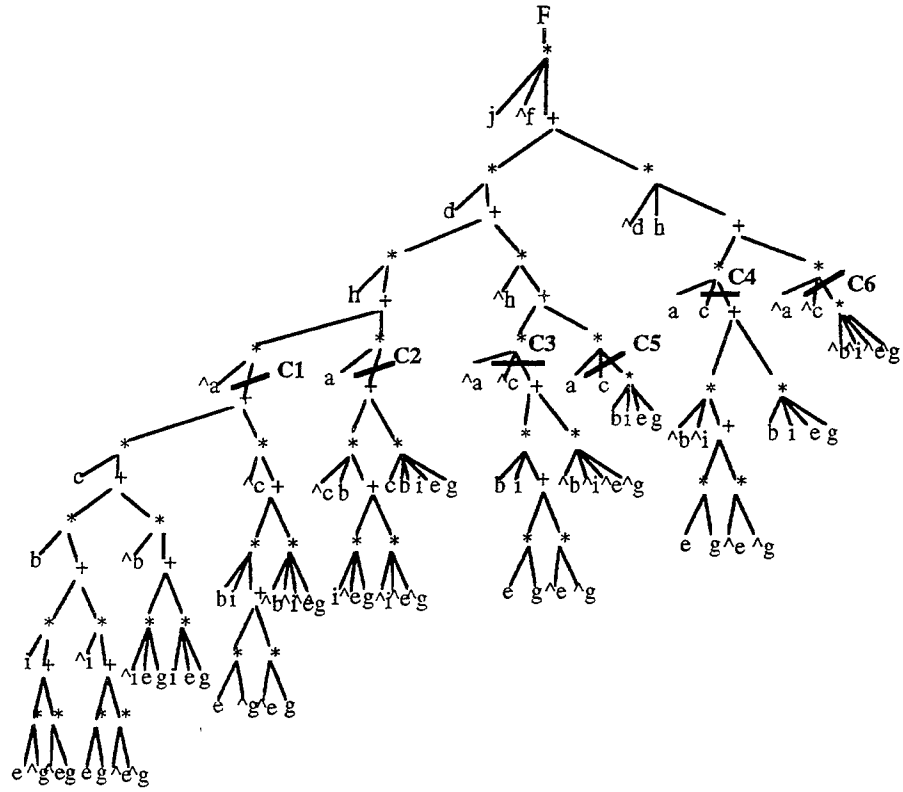


Figure 3.6 : Décomposition d'une fonction Booléenne en sous fonctions dépendants de cinq variables

3.4 Méthodes de décompositions technologiques

3.4.1 Méthode I : Méthode fondée sur les sous-chaînes lexicographiques

Communément la décomposition de fonctions Booléennes, en sous fonctions dépendantes de k variables au maximum, est faite en optimisant le remplissage des cellules sans tenir compte de l'ordre des entrées. Cette technique présentée dans [Fra90] est connue sous le nom de « bin packing ».

On propose ici une méthode de décomposition technologique en partant des feuilles de l'arbre lexicographique. L'algorithme consiste à remplacer l'insertion de point de coupure dépendant strictement d'une chaîne terminale SI par une procédure qui sature la cellule de base. Le noeud initial est le noeud le plus profond des feuilles restantes dans l'arbre.

Définitions préalables

- 1- Un prédécesseur immédiat d'un noeud dans un arbre orienté est appelé noeud père.
- 2- Un frère d'un noeud N est un noeud qui a le même noeud père que N.
- 3- Les variables associées à un noeud N sont l'ensemble des variables qui apparaissent sur les feuilles successeurs de N. Cet ensemble est noté var(N).
- 4- L'ensemble des variables associées à un ensemble de noeuds, est l'union des ensembles des variables associées à chaque noeud ;

  var (N1, ..., Nn)= U var(Ni).

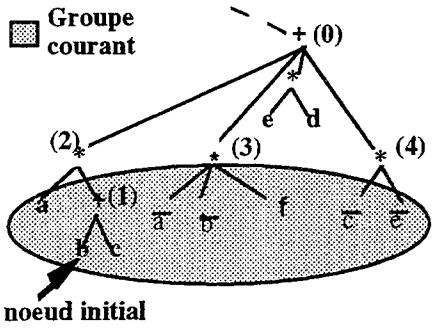
L'algorithme du partitionnement optimisant le remplissage des cellules sur un arbre lexicographique est le suivant :

Pas a) Le noeud-courant est la feuille de profondeur maximale.
Pas b) tant que (|var(noeud-courant)| ≤ k)
{ noeud-courant = père (noeud-courant) ;
fils-1 = noeud-courant ;
Si (noeud-courant == racine) fin ;}
Pas c) ensemble-des-fils = {fils-1} ;
Ajouter à ensemble-des-fils les frères de fils-1 tant que |var(ensemble-des-fils)| ≤ k. Choisir en priorité les frères de fils-1 qui ont le maximum de variables en commun avec var(ensemble-des-fils).
Pas d) insérer un point de coupure sous le noeud-courant, en appliquant l'algorithme d'insertion de points de coupure avec SI=var(ensembles-des-fils).
Pas e) le sous-arbre sous le point de coupure est remplacé par une feuille étiquetée par une nouvelle variable SFi, SFi est insérée dans la chaîne lexicographique avec une priorité minimale, aller au pas a.

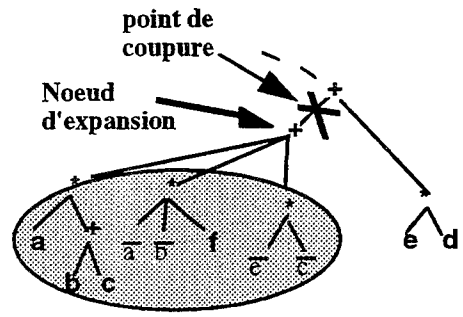
Exemple :

La figure 3.7 illustre le déroulement de cet algorithme sur le noeud (0). L'équation pour ce noeud est: SF= a.(b+c)+($\bar{a}.\bar{b}$.f)+e.d+$\bar{e}.\bar{c}$.

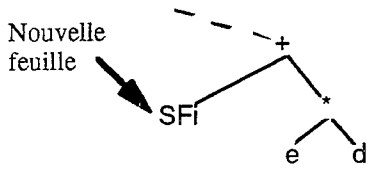
Le noeud le plus profond est b (ou c). Au pas c: *fils-1* est le noeud (0). On collecte alors les noeud 2, 3 et 4 (figure 3-7-a). Une expansion du noeud (0) est crée (figure 3-7-a) et le pas d crée la nouvelle feuille Sfi (figure 3-7-c) et l'arbre correspondant (figure 3-7-d).



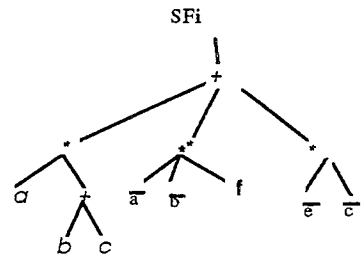
(a) ensemble-des-fils (Pas c).



(b). Insertion de point de coupure



© Arbre après création de la sous fonction



(d) arbre de la sous fonction

Figure 3.7 : Méthode I

3.4.2 Méthode II : amélioration de la méthode I par saturation des cellules

Le partitionnement guidé par les sous-chaînes lexicographique d'entrées est restrictif puisque la localisation des points de coupure est strictement définie par la sous-chaîne lexicographique des entrées. Avec le partitionnement guidé par la tranche d'entrées, quelques cellules ne sont pas saturées ou non remplies efficacement. De l'autre côté, le partitionnement classique guidé par la cardinalite ignore l'ordre des entrées en choisissant les noeuds de coupure. C'est pourquoi, nous proposons une stratégie mixte.

Cette stratégie est liée à la définition des tranches comme le partitionnement guidé par les entrées mais elle tient compte des variables du premier noeud

choisi. A chaque itération, la plus profonde des feuilles étiquetées par une variable dans la tranche terminale est choisie. Ceci est justifié par le fait que les feuilles étiquetées par des variables de la tranche terminale courante sont traitées avant de considérer une autre tranche. Pour saturer les cellules créées, on choisit les noeuds frères qui minimisent le nombre des variables qui n'appartiennent pas à la tranche terminale courante. Avec ce choix, on arrive à un compromis entre l'optimisation des cellules et le respect de la structure lexicographique.

L'algorithme de cette stratégie est le suivant :

Pas a) la chaîne lexicographique est partitionnée en tranches de k variables SL1, .., SLn dans l'ordre décroissant de priorité. la tranche courante est celle de priorité maximale.

Pas a) S'il n'y a plus de variables de la tranche courante dans l'arbre, la tranche courante devient la tranche suivante dans l'ordre décroissant de priorité.

Le noeud-courant est la feuille de profondeur maximale qui appartient à la tranche courante.

Pas b) tant que $(|var(noeud\text{-}courant)| \leq k)$

{ noeud-courant = père (noeud-courant) ;

fils-1 = noeud-courant ;

Si (noeud-courant == racine) fin ;}

Pas c) ensemble-des-fils = {fils-1} ;

Ajouter à ensemble-des-fils les frères de fils-1 tant que $|var(ensemble\text{-}des\text{-}fils)| \leq k$. *Choisir en priorité les frères de fils-1 qui minimisent le nombre des variables qui n'appartiennent pas à la tranche courante.*

Pas d) insérer un point de coupure sous le noeud-courant, en appliquant l'algorithme d'insertion de points de coupure de la section 2-1 avec SI=var(ensembles-des-fils).

Pas e) le sous-arbre sous le point de coupure est remplacé par une feuille étiquetée par une nouvelle variable SFi, SFi est insérée dans la chaîne lexicographique avec une priorité minimale, aller au pas a.

Exemple :

Soit F=f.(c.b+d.c.a+e.c.a+e.d.c) une expression lexicographique avec l'ordre de référence (f e d c b a). Si nous supposons k=4, la première tranche est (d c b a). La figure 3.8 montre les variables contenues dans la première cellule en utilisant le partitionnement saturant les cellules (bloc de droite) et celle crée par l'algorithme mixte proposé ici (bloc de gauche).

Les deux algorithmes dans cet exemple donnent une cellule saturée mais avec la stratégie mixte, la cellule ne contient que des variables de la tranche courante.

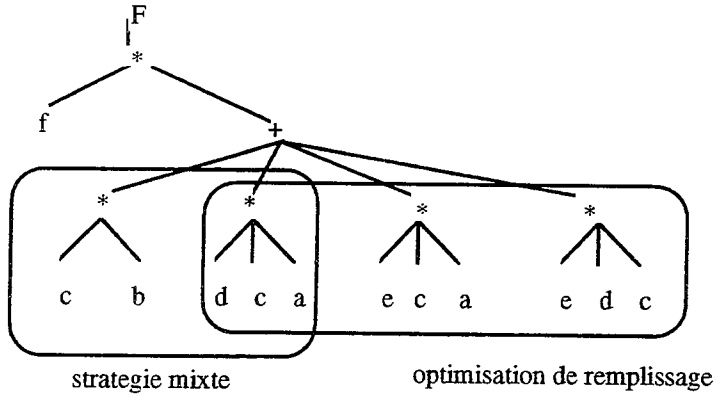


Figure 3.8 : Comparaison entre les deux stratégies.

L'intérêt de l'arbre lexicographique est que la plupart des cellules ainsi créées auront des variables ou appartenant à la même tranche ou à deux tranches successives. Comme conséquence, le regroupement des cellules sera plus probable. Ce regroupement est spécialement intéressant dans les cas de Xilinx série 3000.

3.5 Décomposition technologique adaptée aux réseaux Xilinx

Nous présenterons successivement, les méthodes de décomposition adaptées aux séries 3000 et 4000 avec comme critères d'optimisation, respectivement la surface et le chemin critique.

3.5.1 Décomposition orientée surface pour la série 3000

Cet algorithme a pour objet de minimiser le nombre de CLBs. Nous appliquons la méthode II en prenant en compte de façon plus précise les caractéristiques des CLBs de la série 3000. Ceci est obtenu en contrôlant le choix entre des sous fonctions de quatre variables et des sous fonctions de cinq variables. Une sous fonction de quatre variables est intéressante seulement si elle partagera un CLB avec une autre sous fonction. Une sous fonctions de quatre variables est créée seulement si elle possède au moins trois variables dans la tranche courante, sinon une sous fonction de cinq variables est créée. Ceci favorise la création de sous fonctions qui dépendent du même sous-ensemble de variables à chaque itération.

L'algorithme de partitionnement orienté surface est le suivant :

Pas a) la chaîne lexicographique est partitionnée en tranches de 4 variables SL1, .., SLn dans l'ordre décroissant de priorité. la tranche courante est celle de priorité maximale.

Pas a) S'il n'y a plus de variables de la tranche courante dans l'arbre, la tranche courante devient la tranche suivante dans l'ordre décroissant de priorité.

Le noeud-courant est la feuille de profondeur maximale qui appartient à la tranche courante.

k =4 ;

Pas b) tant que (|var(noeud-courant)| ≤ k)

{ noeud-courant = père (noeud-courant) ;

fils-1 = noeud-courant ;

Si (noeud-courant == racine) fin ;}

Pas c) ensemble-des-fils = {fils-1} ;

Ajouter à ensemble-des-fils les frères de fils-1 tant que |var(ensemble-des-fils)| ≤ k. Choisir en priorité les frères de fils-1 qui minimisent le nombre des variables qui n'appartiennent pas à la tranche courante.

Si (k == 4)

{ Si (au moins trois variables de ensemble-des-fils sont dans la tranche courante)

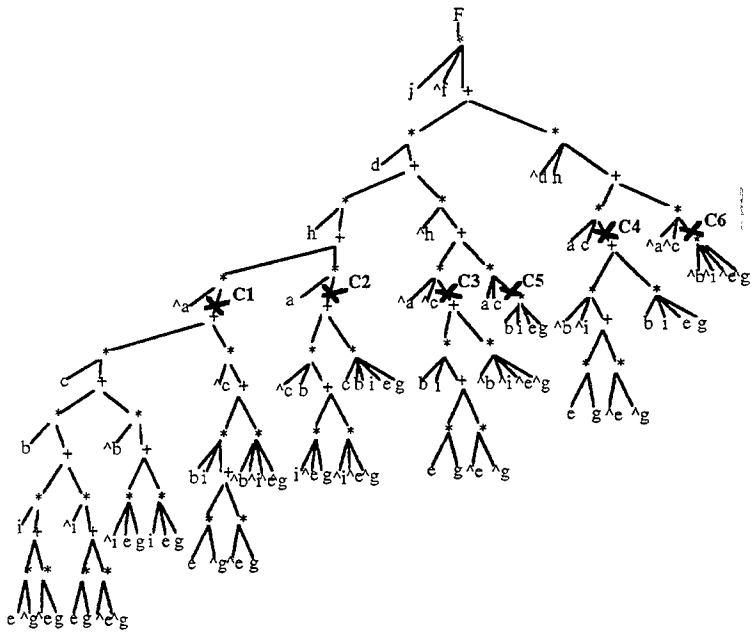
{ k = 5 ; aller au pas b ;}

}

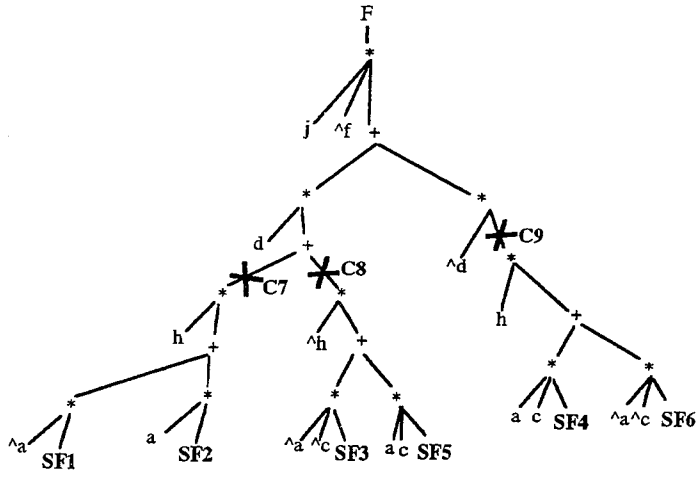
Pas d) insérer un point de coupure sous le noeud-courant, en appliquant l'algorithme d'insertion de points de coupure avec SI=var(ensemble-des-fils).

Pas e) le sous-arbre sous le point de coupure est remplacé par une feuille étiquetée par une nouvelle variable SFi, SFi est insérée dans la chaîne lexicographique avec une priorité minimale, aller au pas a.

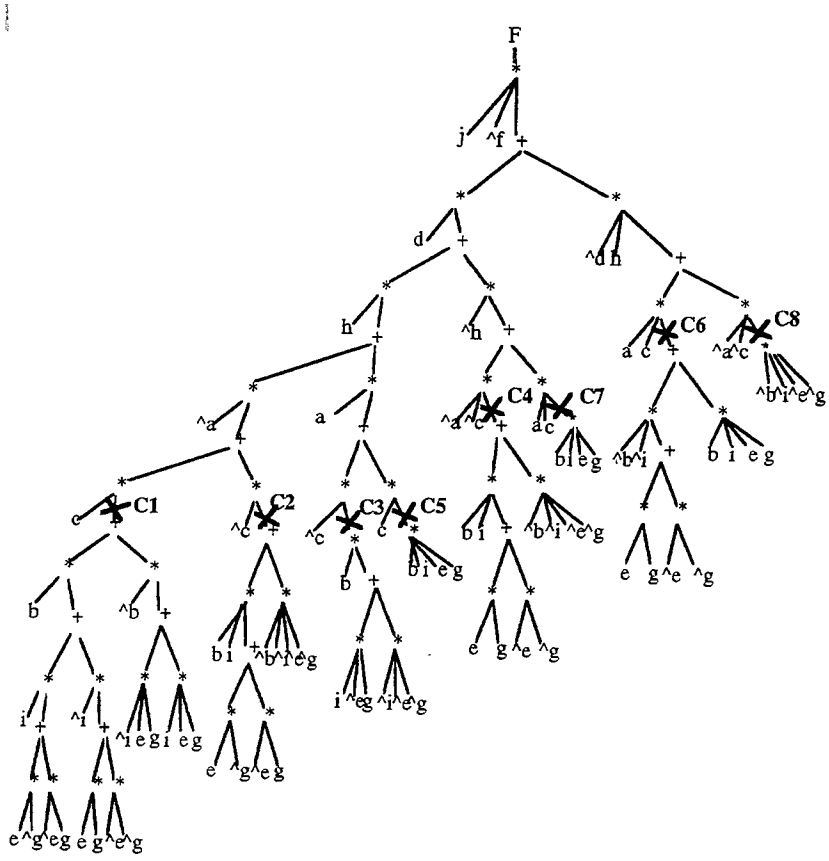
Utilisons le même exemple de la figure . La première tranche est définie par (b i e g). Les points de coupure C1, C2, C3, C4, C5 et C6 sont créés successivement (fig. 3.9-a) et introduisent les six sous fonctions Sf1 à SF6. Les points de coupure C1 et C2 correspondent au pas c avec l'extension à cinq variables. Les points de coupures C5 et C6 nécessitent l'expansion du noeud (dans la figure 3.9-a le noeud ET correspondant est déjà développé). l'arbre obtenu après la création des sous fonctions associées aux points de coupures est donné dans la figure 3.9-b.



(a) algorithme de décomposition (première tranche)



(b) algorithme de décomposition (seconde tranche)



(c) Stratégie mixte

Figure 3.9 : algorithme de décomposition

Les points de coupure obtenus par la stratégie mixte (avec k=4) sont donnés dans la figure 3.9-c pour le même exemple. Avec la stratégie mixte, huit points

de coupures C1 à C8 sont créés. L'algorithme modifié avec le choix entre quatre et cinq variables a fait gagner deux CLBs.

3.5.2 Décomposition orientée surface pour la série 4000

Pour la série 4000, la différence principale de l'algorithme de partitionnement est le critère du choix entre des sous fonctions de quatre et des sous fonctions de trois variables. D'après la structure de la cellule de la série 4000, une sous fonction de n variables (n < 4) sera créée si elle a au moins n-1 entrées qui sont des fonctions dépendant de quatre variables, c'est-à-dire si elle peut être réalisée par le LUT « H » du CLB.

Nous utilisons pour cela un marquage des points de coupure. Un point de coupure N est marqué « H », s'il a au plus 3 variables et si au moins n-1 de ses successeurs sont marqués « F » (ou n est le nombre de variables de N). Un point de coupure est marqué « F » s'il ne vérifie pas cette condition. Un noeud étiqueté par une variable d'entrée primaire est marqué par « I ».

Dans cette algorithme, Nous commençons par une sous fonction de trois variables ; si le point de coupure obtenu vérifie la condition nous marquons le point de coupure par « H », sinon on passe à un point de coupure de taille quatre et il est marqué par « F ». L'algorithme pour la série 4000 est :

Pas a) la chaîne lexicographique est partitionnée en tranches de 4 variables SL1, .., SLn dans l'ordre décroissant de priorité. la tranche courante est celle de priorité maximale.

Pas a) S'il n'y a plus de variables de la tranche courante dans l'arbre, la tranche courante devient la tranche suivante dans l'ordre décroissant de priorité.

Le noeud-courant est la feuille de profondeur maximale qui appartient à la tranche courante.

k =3 ;

Pas b) tant que (|var(noeud-courant)| ≤ k)

{ noeud-courant = père (noeud-courant) ;

fils-1 = noeud-courant ;

Si (noeud-courant == racine) fin ;}

Pas c) ensemble-des-fils = {fils-1} ;

Ajouter à ensemble-des-fils les frères de fils-1 tant que |var(ensemble-des-fils)| ≤ k. Choisir en priorité les frères de fils-1 qui minimisent le nombre des variables qui n'appartiennent pas à la tranche courante.

Si (k == 3)

{ Si (au moins deux variables de ensemble-des-fils sont des points de coupures / sous-fonctions */)*

{ k = 4 ; aller au pas b ;}

}

Pas d) insérer un point de coupure sous le noeud-courant, en appliquant l'algorithme d'insertion de points de coupure avec SI= var(ensemble-des-fils).

Pas e) le sous-arbre sous le point de coupure est remplacé par une feuille étiquetée par une nouvelle variable SFi, SFi est insérée dans la chaîne lexicographique avec une priorité minimale, aller au pas a.

3.5.3 Décomposition orientée chemin critique pour la série 3000

Dans le but de contrôler le chemin critique, nous utiliserons une estimation du temps d'arrivée des sorties. Pour cela, nous définissons recursivement le temps d'arrivée (qu'on notera par TD) d'un noeud dans l'arbre lexicographique, en appliquant les règles suivantes :

- Le temps d'arrivée d'un noeud étiqueté par une entrée primaire est égal à zéro.
- le temps d'arrivée d'un noeud « OU » ou un noeud « ET », est le temps d'arrivée maximal de tous ses successeurs.
- Le temps d'arrivée d'un noeud étiqueté par une sous fonction (point de coupure) est le temps d'arrivée maximal de tous ses successeurs plus un. Ceci reflète le fait que chaque sous fonction sera assignée à un CLB. Le temps d'arrivée de la sortie du CLB sera le temps d'arrivée maximal des entrées du CLB plus le temps de traversée du CLB.

L'idée essentielle de cette approche est de ne pas augmenter la profondeur du noeud courant de plus d'un niveau à chaque fois qu'on crée un point de coupure même si la sous fonction ainsi créée n'est pas saturée. La phase de réinjection locale, qu'on discutera plus loin, essayera de maximiser le remplissage sans augmenter la profondeur. L'algorithme se fait en deux phases. La première phase est celle de marquage des noeuds par le couple (|var (n)|, DT(n)), puis le parcours de l'arbre en profondeur à partir de la racine. A chaque noeud on prend le noeud fils qui a le plus grand nombre de variables (|var (n)|). On s'arrête sur le noeud n tel que :

- $|var(n)| \geq k$ et $\forall$ *p* prédécesseur de n alors $|var(p)| \leq k$.

L'algorithme orienté chemin critique est :

Pas a) la chaîne lexicographique est partitionnée en tranches de 4 variables SL1, .., SLn dans l'ordre décroissant de priorité. la tranche courante est celle de priorité maximale.

Pas a) S'il n'y a plus de variables de la tranche courante dans l'arbre, la tranche courante devient la tranche suivante dans l'ordre décroissant de priorité.

k =4 ; noeud-courant = racine de l'arbre;

Pas b) Parcourir l'arbre en profondeur

tant que (|var(noeud-courant)| > k)

{ noeud-courant = fils (noeud-courant) ;

fils-1 = noeud-courant ;

Si (noeud-courant == racine) fin ;}

Pas c) ensemble-des-fils = {fils-1} ;

Ajoute à ensemble-des-fils le frère B de fils-1 tant que (|var(ensemble-des-fils)| ≤ k) et (TD(B) < TD(noeud-courant)). Choisir en priorité les frères du fils-1 qui ont le temps d'arrivée minimal. S'il existe plusieurs frères de même temps d'arrivée. Il faut choisir celui qui minimise le nombre des variables qui n'appartiennent pas à la tranche courante.

Si (k == 4)

{ Si (tous les variables de ensemble-des-fils sont dans la tranche courante) ou (|var(ensemble-des-fils)| < 4)

{ k = 5 ; aller au pas b ;}

}

Pas d) insérer un point de coupure sous le noeud-courant, en appliquant l'algorithme d'insertion de points de coupure avec SI=var(ensemble-des-fils).

Pas e) le sous-arbre sous le point de coupure est remplacé par une feuille étiqueté par une nouvelle variable SFi, SFi est insérée dans la chaîne lexicographique avec une priorité minimale, aller au pas a.

Dans cet algorithme, le chemin critique est optimisé en réduisant le nombre de CLBs traversés dans le chemin critique. La minimisation des délais introduits par le routage, comme on l'a mentionné dans le chapitre II, est garantie par la factorisation lexicographique.

La figure 3.10 montre la différence entre l'algorithme orienté surface et l'algorithme orienté chemin critique. La configuration initiale est un réseau de 4 LUT et de profondeur 2. Sur la figure 3-10-a, on remarque que l'option surface, en cherchant à saturer le LUT, crée la sous fonction $SF_6 = \bar{b}.d.(c+SF_2).SF_3$ qui a une profondeur de 3.

Sur la figure 3-10-a', l'option chemin critique en parcourant l'arbre à partir de la racine, s'arrête au noeud étiqueté (7,2) et crée la sous-fonction $SF_6' = \bar{b}.d.(c+SF_2)$ ce qui ne sature pas le LUT mais n'augmente pas la profondeur du noeud courant (2). Une seconde itération (figure 3-10-b') de l'approche chemin critique s'arrête au noeud (6.2) et crée la sous fonction $SF_7 = SF_2.b$ de profondeur 2, ceci ne change pas la profondeur du noeud racine. Les figures 3-10-b et 3-10-c' représentent des fonctions de 5 variables qui sont les points des sorties des algorithmes.

Nous remarquons que l'approche surface donne 2 LUT avec une profondeur égale à 4, alors que l'approche chemin critique donne 3 LUT avec une profondeur égale à 3.

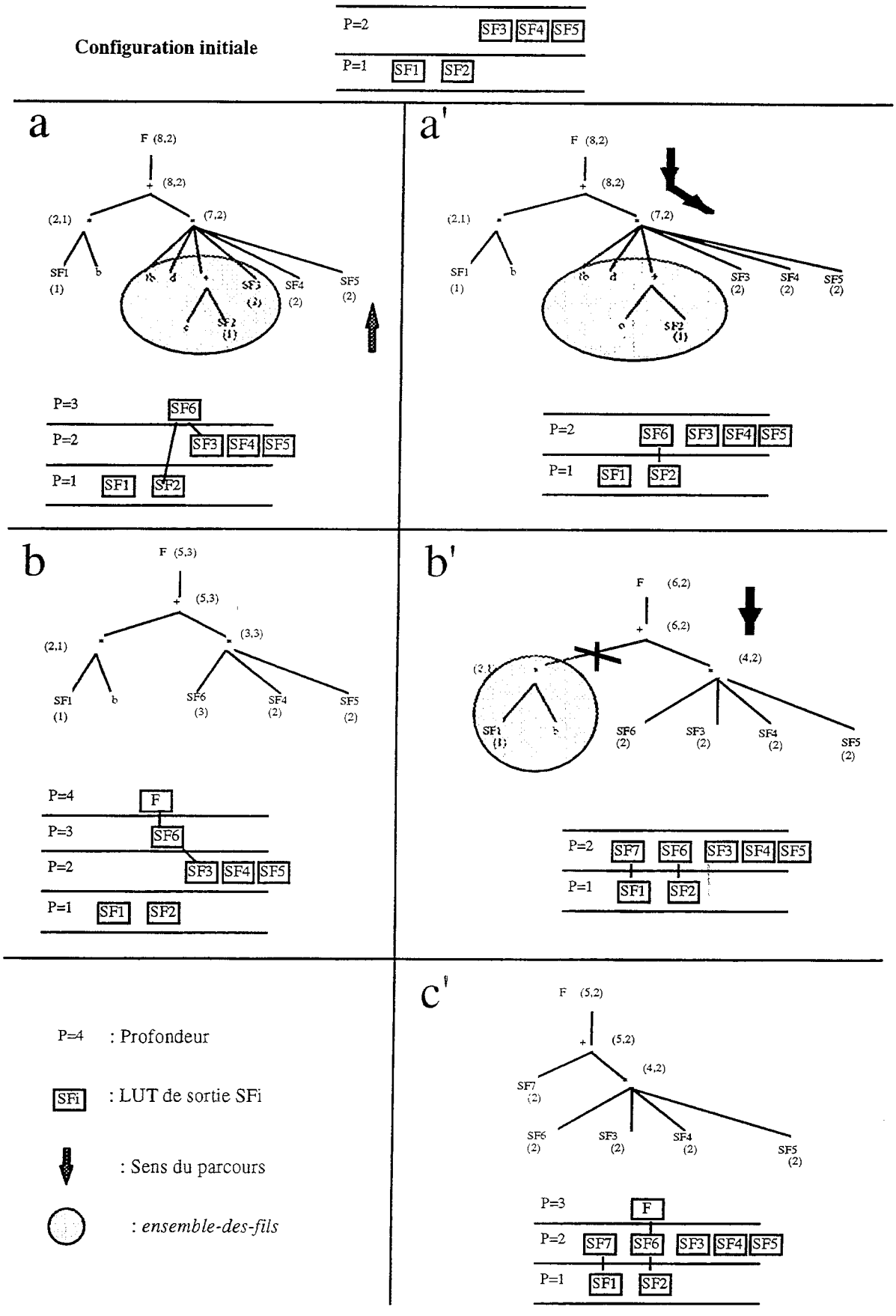


Figure 3.10 : Comparaison des décompositions orientées surface et chemin critiques

3.6 Réinjection

La réinjection d'une sous fonction partagée est le processus inverse de la sélection d'un candidat diviseur partagé. Elle consiste à ne plus exprimer un sous-ensemble de l'ensemble des fonctions initiales à l'aide de cette sous fonction partagée, mais à l'exprimer en terme des entrées de cette sous fonction partagée. Au niveau des arbres factorisés, elle consiste à supprimer un sous-arbre partagé en le réinjectant dans les arbres de départ.

Exemple :

$F = a.b.SF_1$

$G = \bar{c}.\bar{d}.SF_1$

$H = SF_2$

$SF_1 = e+f.SF_2$

$SF_2 = g.h$

après réinjection de SF_1

$F = a.b.(e+f.SF_2)$

$G = \bar{c}.\bar{d}.(e+f.SF_2)$

$H = SF_2$

$SF_2 = g.h$

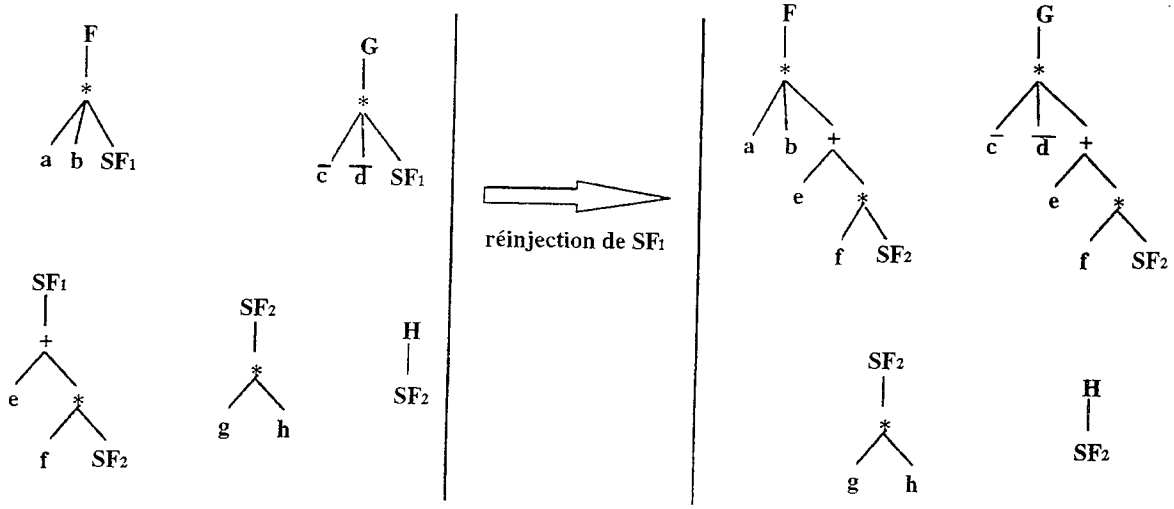


Figure 3.11 : Réinjection

La réinjection locale d'une sous fonction partagée par plusieurs fonction, consiste à la réinjecter dans certaines fonctions.

La factorisation peut créer des sous fonctions partagées avec un support de cardinalite très inférieur à k. De plus, après le partitionnement des sous

fonctions partagées qui ont initialement un support important supérieur à k, on peut obtenir des sous fonctions ayant un support de cardinalite très réduite. Dans les deux cas, on a intérêt à envisager la réinjection de ces sous fonctions. La réinjection dans tout le système peut être très coûteuse, même dans le cas de sous fonctions de deux variables seulement. Pour cela, nous ferons seulement une réinjection locale des sous fonctions partagées en tenant compte du point de coupure au quel elle sera associée.

Dans ce but, la décomposition se fera en deux phases. La première phase consiste à décomposer toutes les sous fonctions partagées puisque ces sous fonctions peuvent être elles-mêmes exprimées en fonctions de sous fonctions et on trie la liste des sous fonctions partagées dans l'ordre croissant de profondeur. La deuxième phase consiste à décomposer les fonctions de sorties primaires. A chaque fois qu'on crée un point de coupure, nous réinjectons localement à la sous fonction associée au point de coupure, toutes les sous fonctions qu'elle utilise si le support de celle-ci reste inférieur à k variables. Pour le chemin critique, on réinjecte en priorité les sous fonctions qui ont un temps d'arrivée maximal. Pour l'option surface, on réinjecte en priorité les sous fonctions qui introduisent le minimum de nouvelles variables à la sous fonction associée au point de coupure.

3.7 Conclusion

La méthode proposée, pour la décomposition d'une fonction booléenne en sous fonctions d'un nombre limité de k variables, est adaptée à la structure de l'arbre lexicographique. Sa principale caractéristique est la grande cohérence avec l'étape de factorisation. En effet, elle conserve l'ordre des variables et les cônes logiques créés par la factorisation lexicographique. Cette approche est valable pour tous les réseaux programmables à base de LUT. Elle est également applicable à tous les réseaux de type CPLD et a été appliquée avec succèes à ces cibles. Cette application n'est pas discutée dans cette thèse pour ne pas disperser l'attention du lecteur.

Chapitre IV

Autres méthodes de factorisation

4.1 Introduction

La factorisation lexicographique est développée dans le but de résoudre les problèmes de routage que rencontre les utilisateurs de FPGA (essentiellement pour la série 3000 de Xilinx).

Cette méthode restreint l'étape de factorisation en imposant un ordre sur les variables d'entrées pour l'ensemble des fonctions. Cette restriction impose un choix limité des noyaux qui doivent être lexicographiquement compatibles. Les fonctions factorisées ne sont pas nécessairement minimales en nombre de littéraux et par conséquent les résultats obtenus par cette approche ne sont pas toujours satisfaisants en terme de nombre de cellules.

Pour présenter une méthode de synthèse logique sur FPGA, la plus complète possible et donc offrir des résultats optimisés suivant plusieurs critères; nous présentons deux autres méthodes de factorisations:

La première factorisation est basée sur le théorème de Shannon et la construction des ROBDD. Comme nous le verrons cette méthode utilise aussi un ordonnancement des variables d'entrées comme la factorisation lexicographique. On profitera de la même notion de cônes logiques comme pour la lexicographie, et des avantages de partage des variables entre sous fonctions en vue de préparer le routage.

La deuxième méthode est la factorisation algébrique restreinte à une division par les conoyaux pour laquelle nous présentons des filtres pour le choix des noyaux. Ces filtres sont spécifiques à notre cible technologique et aux critères d'optimisation, ce qui nous permettra d'offrir des résultats optimisés surface ou chemin critique, mais moins sensibles aux problèmes de routage.

4.2 Factorisation basée sur le théorème de Shannon

Devant l'accroissement considérable de la taille des fonctions Booléennes à traiter (grand nombre de fonctions, de monômes et de variables), de nouvelles techniques de calculs Booléens sont apparues ces dernières années [Bry86].

Elles sont fondées sur la récursivité du théorème de Shannon et non sur les formules habituelles du calcul Booléen.

Théorème 4.1: Théorème de Shannon

Soit une fonction f à n variables. On peut écrire:

- Première forme du théorème de Shannon:

$f(x_1,x_2...x_n) = x_i.f(x_1,x_2...1...x_n)+ x_i.f(x_1,x_2...0...x_n)$

- Deuxième forme du théorème de Shannon:

$f(x_1,x_2...x_n) = (x_i+f(x_1,x_2...1...x_n)).(x_i+f(x_1,x_2...0...x_n))$

$f(x_1,x_2...1...x_n)$ (resp. $x_i.f(x_1,x_2...0...x_n)$) est appelé cofacteur de f par rapport à x_i (resp. $\overline{x_i}$). Il est noté f_{x_i} (resp. $f_{\overline{x_i}}$).

Définition 4.1: Arbre de décision binaire

Un arbre de décision binaire (*Binary Decision Diagram:* BDD) est une arborescence associée à la décomposition d'une fonction Booléenne par la formule de Shannon selon un ordre fixé des variables d'entrées [MOR82].

On peut construire un BDD en appliquant récursivement la première forme du théorème de Shannon sur les variables de f. On applique à chaque niveau les règles de calcul suivantes:

- 1+f=1
- 0+f=f
- 1.f=f
- 0.f=0

On s'arrête dès que l'on se trouve en présence d'une constante (1 ou 0).

Dans un BDD, chaque noeud non terminal est associé à une variable d'entrée x_i. Il correspond à une certaine fonction F_i et possède deux fils correspondant aux cofacteurs de F_i par rapport à x_i (fils gauche) et au cofacteur de F_i par rapport à $\overline{x_i}$ (fils droit) (figure 4.1). Les feuilles du BDD sont donc toutes des constantes Booléennes.

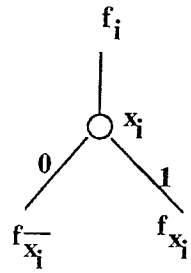


Figure 4.1: Représentation graphique d'un BDD

Exemple :

Soit f(a,b,c)=a.b+c, sa représentation sous forme de BDD est celle de la figure 4.2.

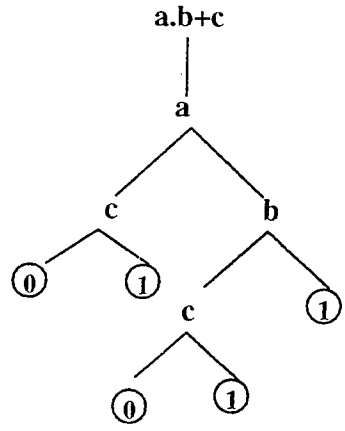


Figure 4.2: Représentation de f sous forme de BDD avec l'ordre a,b,c

4.2.1 Applications basées sur les BDD

Comme chaque décomposition de Shannon est unique pour une fonction et une variable donnée, un BDD est une représentation canonique, modulo l'ordre des variables.

Les applications basées sur les BDD, que nous développons, découlent de cette propriété.

4.2.1.1 Preuve de tautologie

Une fonction est une tautologie si et seulement si elle est toujours égale à 1. On en déduit donc que:

$f(x_1,x_2...x_n)$ est une tautologie si et seulement si f_{x_i} et $f_{\overline{x_i}}$ sont des tautologies.

On peut donc élaborer une procédure récursive sur BDD, qui reconnaît si une expression Booléenne est une tautologie. On remarque que dès qu'une branche de l'arbre de décision binaire aboutit à un échec, on peut immédiatement réfuter le test de tautologie pour la fonction de départ.

Exemple :

La fonction $f = \overline{a}.b.c + a.b.c + \overline{b}.c + \overline{b}$ est une tautologie. Pour s'en convaincre, on construit un BDD de la fonction et on remarque que toutes ses feuilles sont égales à 1.

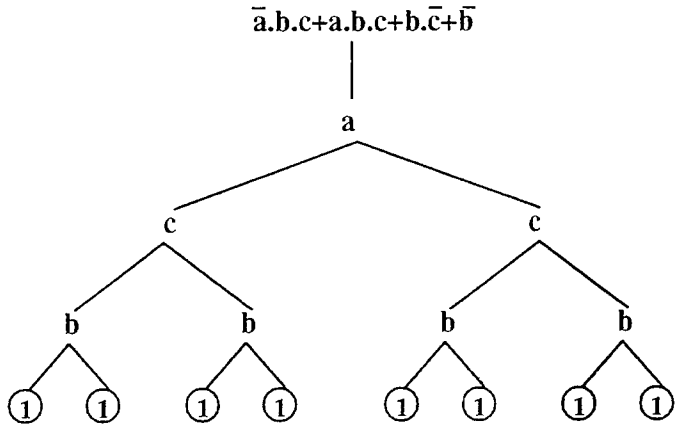


Figure 4.3: Preuve de tautologie par BDD

Accélération de la méthode

On utilise le théorème de Shannon pour accélérer la méthode.

Accélération 1:

Si un monôme composé d'une seule variable x apparaît dans l'expression Booléenne de f, on sait que la preuve de tautologie se réduit sur $f_{\overline{x}}$ car:

Si $f = x + g'$ alors $f = x + f_{\overline{x}}$ donc $f=1$ $f_{\overline{x}} = 1$

Exemple :

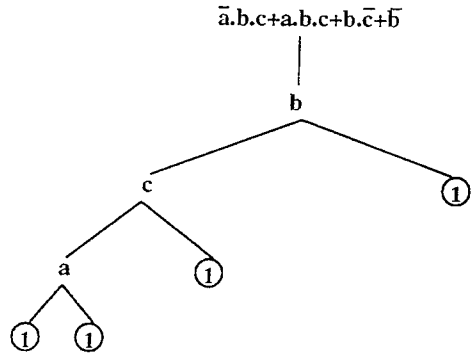


Figure 4.4: Accélération de la preuve de tautologie.

Accélération 2:

Si une variable x apparaît dans l'expression de f toujours:

- sous la forme normale, alors la recherche se réduit à la recherche sur $f_{\bar{x}}$,

car : $f = 1 \Leftrightarrow f_{\bar{x}} = 1$.

- sous la forme complémentée, alors la recherche se réduit à la recherche sur f_x, car : $f = 1 \Leftrightarrow f_x = 1$.

Réduction du BDD:

Quand aucun des deux cas précédents n'est détecté, on choisit la variable qui apparaît le plus sous ses deux formes dans l'expression de f. On minimise ainsi le nombre de monômes du niveau suivant et donc la taille de l'arbre de décision binaire.

Réfutation:

Si l'un des cas suivant est détecté, il est possible d'arrêter le traitement; la fonction n'est pas une tautologie:

- Toutes les variables de f sont monoformes.
- Une variable monoforme est présente dans tous les monômes.
- La somme des tailles des monômes est inférieure à 2^n.

4.2.1.2 Calcul du complément

En utilisant le théorème de Shannon, on obtient:

$$\overline{f} = x_i . \overline{f_{x_i}} + \overline{x_i} \; \overline{f_{\overline{x_i}}}$$

Donc, en itérant sur les variables successives de la fonction, on obtient une méthode de calcul récursif du complément.

Exemple:

Pour f= a. $\overline{b}$ + b.c + $\overline{a}$ $\overline{c}$, la recherche du complément sur BDD est constituée de deux phases: la descente, telle que l'illustre la figure 4.5 $\overline{f}$ et la montée, qui définit la fonction suivante:

$$\overline{f} = \overline{a} . \overline{(b.c + \overline{c})} + a. \overline{(\overline{b} + b.c)}$$

$$\overline{f} = \overline{a} .(\overline{b} .c + b.0) + a.(\overline{b} .0 + b. \overline{c})$$

$$\overline{f} = \overline{a} . \overline{b} .c + a.b. \overline{c}$$

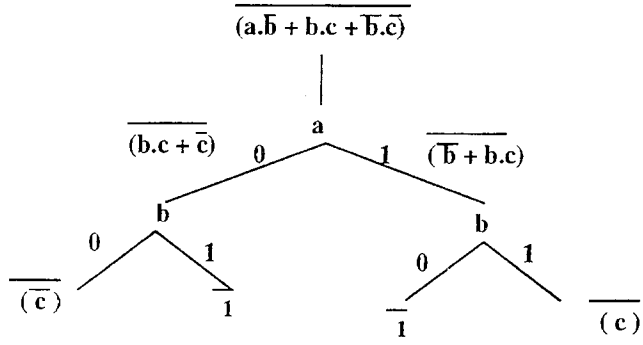


Figure 4.5: Recherche du complément sur BDD (descente).

Accélération de la méthode

Accélération lors de la descente:

A chaque niveau de la descente s'il existe dans f un monôme composé d'une seule variable, on choisit d'abord cette variable pour la décomposition.

Soit x_i cette variable, on sait alors que $\overline{f}$ se réduit à: $\overline{f} = \overline{x_i} \; \overline{f_{\overline{x_i}}}$

Réduction du BDD:

On choisit, comme pour la preuve de tautologie, la variable qui minimise le nombre de monômes du niveau suivant, c'est-à-dire la variable qui apparaît le plus sous ses deux formes.

Accélération lors de la remontée :

• S'il existe un monôme m; élément de $\overline{f_{x_i}}$ et de $\overline{f_{\overline{x_i}}}$ alors:

$$\overline{f} = x_i.(\overline{f_{x_i}} - \{m\}) + \overline{x_i}\ (\overline{f_{\overline{x_i}}} - \{m\}) + m$$

La détection d'un monôme identique dans les deux expressions permet ainsi une simplification rapide et significative de la fonction résultante à chaque niveau de la montée (gain d'un monôme et d'une variable).

• Si f est composée d'un seul monôme, on applique la loi de De Morgan.

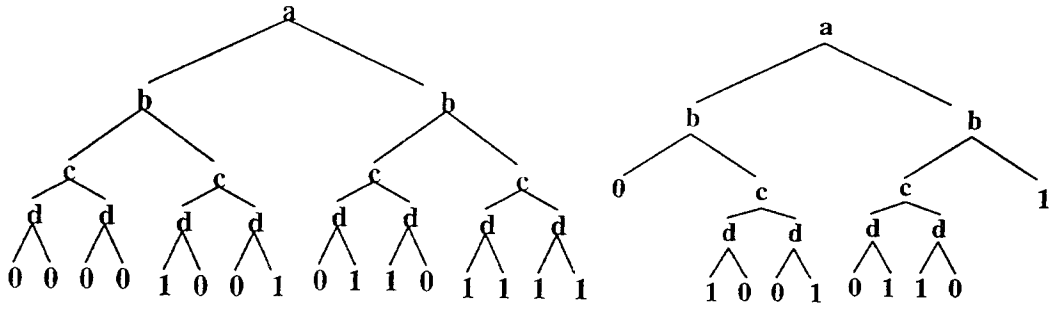
4.2.2 Minimisation du graphe de décision binaire

Une application directe de la preuve de tautologie et du calcul de complément est la réduction du BDD.

Dans un BDD, si toutes les feuilles accessibles à partir d'un même noeud sont égales à une constante *C* (0 ou 1), ce noeud est alors redondant et peut être remplacé par la constante *C*. En supprimant ainsi tous les noeuds redondants, nous obtenons l'*arbre réduit de décision binaire* qui est également une représentation canonique pour un ordre fixé de variables.

Exemple :

La figure 4.6 représente le BDD et le BDD réduit de f = a.(c d) + b ().



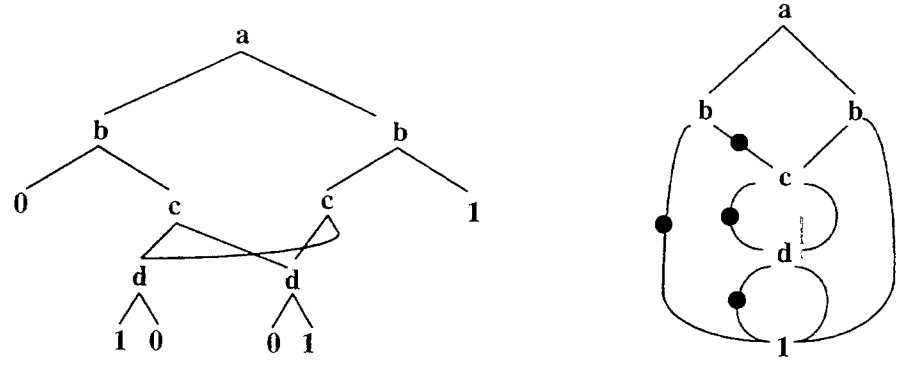
(a) BDD **(b) BDD réduit**

Figure 4.6 : Exemple d'arbre de décision binaire réduit.

Nous pouvons encore minimiser la taille de la représentation en partageant les sous-arbres identiques. On obtient alors un graphe acyclique orienté. Le graphe ainsi obtenu est appelé ROBDD (*Reduced Ordered Binary Decision Diagram*). La figure 4.7-a représente le ROBDD de la fonction de l'exemple précédent.

La dernière amélioration est celle de réduire en même temps la taille des graphes et les opération de négation en introduisant des arcs *typés*: un arc pourra pointer de façon directe ou inverse sur un noeud. Ainsi les sous arbres qui existent de façon inverse et directe dans un ROBDD n'existent plus que sous une seule forme dans un *ROBDD typé*. La figure 4.7-b représente le ROBDD typé de l'exemple précédent. Un arc négatif est marqué par un point noir.

Sans contraintes au moment de la construction, un ROBDD typé n'est pas canonique mais on peut le rendre canonique en imposant par exemple que l'arc correspondant au fils gauche d'un noeud doit toujours être directe.



(a) ROBDD **(b) ROBDD typé**

Figure 4.7: Exemple de ROBDD et ROBDD typé.

Définition 4.2: Multi-graphe de décision binaire

Dans la synthèse logique, nous manipulons des systèmes de fonctions Booléennes. Il est nécessaire d'avoir une définition analogue à celle des

graphes de décision binaire pour représenter un ensemble de fonctions Booléennes.

Un multi-graphe de décision binaire est un graphe acyclique dirigé, possédant plusieurs noeuds sources correspondant aux racines des différents ROBDD des fonctions du système Booléen.

Exemples :

La figure 4.8 représente le multi-graphe de décision binaires des fonctions:

$F1 = a.\bar{b} + \overline{a.b}$

$F2 = \bar{a}.\bar{b} + a.(b+c)$

$F3 = b + c$

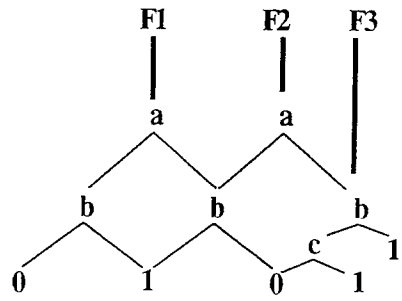


Figure 4.8: Multi-graphe de décision binaire.

4.2.3 Importance de l'ordre des variables

La taille du graphe de décision binaire est étroitement liée à l'ordre des variables de décomposition. Pour illustrer l'importance de l'ordre, prenons la fonction $F = x_1.x_2 + ... + x_{n-1}.x_n$. Si l'on choisie l'ordre x_1,x_2 , ... , x_{n-1},x_n on a un ROBDD de taille $O(2n)$ alors qu'avec l'ordre x_1,x_3,x_5, ... , x_2,x_4,x_6,... le graphe obtenu est de taille $O(2^n)$.

Il est démontré que:

• Il existe des fonctions Booléennes qui n'admettent pas d'ordre permettant d'obtenir un graphe de taille polynomiale.

• étant une fonction Booléenne F, le problème de trouver un ordre qui minimise la taille du graphe de F est NP-complet [Bol94]: Il existe un algorithme de complexité $O(n^2 3^n)$ permettant de déterminer cet ordre.

4.2.4 Heuristiques de recherche de l'ordre des variables

Plusieurs heuristiques sont présentées dans la littérature pour la recherche de l'ordre minimal pour la construction des ROBDD:

- l'heuristique la plus connue est celle basée sur le nombre d'occurrences des variables.

- heuristiques basées sur l'information acquise par l'analyse des circuits combinatoires réalisant la fonction Booléenne [Min90].

- heuristiques fondées sur l'amélioration de la taille du ROBDD par échange de variables [Fuj91] [Iish91] [Rud93].

Dans le but d'avoir un ordre efficace avec le moindre coût, nous avons mené des expériences en utilisant l'ordre lexicographique. Ces expériences ont été concluantes, l'ordre lexicographique élimine les risques d'explosion mémoire. En effet, cette ordre est très similaire à l'ordre induit par les noyaux [Bes93] [Bou93] puisque l'ordre lexicographique est par définition induit par la relation noyaux/conoyaux.

4.3 Décomposition technologique sur arbre de Shannon

A partir du ROBDD, on construit l'arbre de factorisation de l'expression Booléenne (figure 4.9). Cette construction est récursive et s'appuie directement sur le théorème de Shannon: à chaque noeud N du ROBDD, on applique la formule récursive:

$$f(N) = v_N f(D_N) + \overline{v_N} f(G_N)$$

où:

- $f(x)$ est l'équation Booléenne correspondant au sous graphe de racine le noeud x.

- v_N: est la variable de décomposition du noeud N.

- D_N (resp. G_N) fils droit (resp. fils gauche) du noeud N.

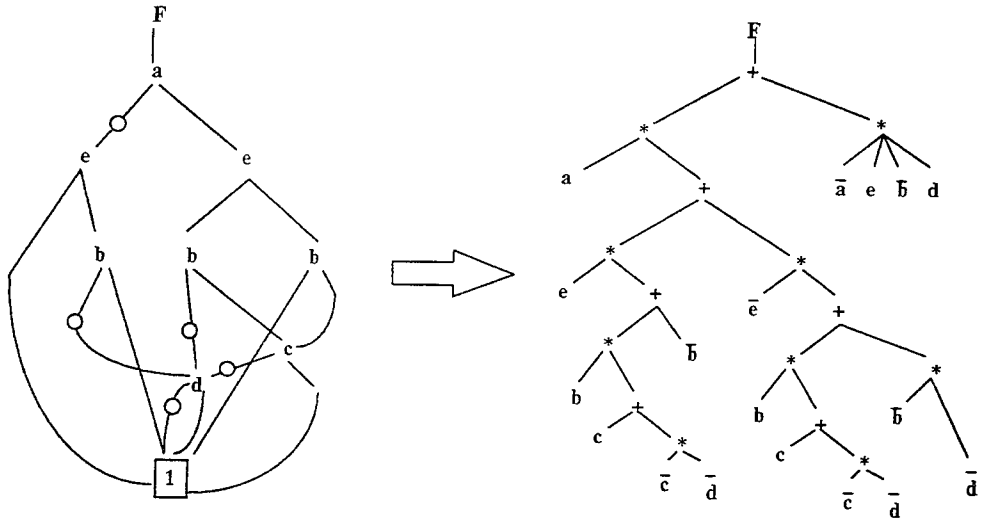


Figure 4.9: Arbre factorisé déduit de la représentation en ROBDD typé.

L'utilisation des ROBDD comme représentation initiale amène plusieurs avantages [Sau93]:

- C'est une représentation compacte; le partage de sous-fonctions et sous-fonctions complémentaires est traité efficacement.

- L'ordonnancement des variables qui est celui de l'approche lexicographique, permet d'appliquer les mêmes principes de décomposition que précédemment.

Du fait de l'ordonnancement des variables d'entrées, l'arbre ainsi obtenu a la propriété la plus importante de l'arbre lexicographique:

L'organigramme de cette approche est donné par la figure 4.10 .

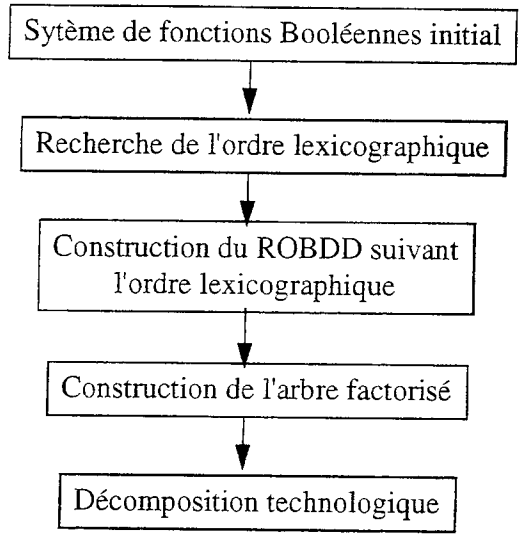


Figure 4.10 : Organigramme de la synthèse fondée sur la décomposition de Shannon

4.4 Factorisation algébrique restreinte

4.4.1 Introduction

La division algébrique classique associée à la substitution algébrique permet de diviser par des noyaux, parties de noyaux ainsi que par le complément de ces diviseurs. Plusieurs parenthèses sont créées indiquant ainsi la création de sous-fonctions éventuellement partagées. On peut avoir trois inconvénients:

-1- un nombre important de couches logiques, ce qui rendra difficile la minimisation ultérieure de chemin critique.

-2- un mauvais contrôle de la connectique car la mise en place de facteurs communs entraîne une connectique plus riche.

-3- une mauvaise maîtrise du choix des candidats diviseurs dans l'approche heuristique. En effet, le choix d'un candidat diviseur entraîne dans le cas classique le choix d'autres candidats (partie de noyau, complément d'un candidat). Ces choix masquent ou empêchent le choix d'autres candidats "ultérieurs" dont le gain peut être supérieur.

Nous présentons une approche alternative plus efficace pour les blocs combinatoires complexes. D'une part, la division algébrique sera restreinte à

une division par conoyau et non à une division par noyau, ce qui empêche la division par les parties de noyaux et l'extension aux facteurs complémentaires. D'autre part, une attention particulière sera portée à la sélection et au filtrage poussé des candidats diviseurs. Ce choix étant guidé par la simplification ultérieure de la connectique.

4.4.2 Principe général

La division algébrique est une méthode puissante qui permet d'effectuer la factorisation d'un ensemble de fonctions Booléennes quelques soient les candidats diviseurs choisis (noyaux et leurs intersections, monômes et plus généralement toutes expressions non libres). La méthode de division restreinte proposée ici est fondée sur la notion de la portée d'un noyau (C,K,ξ) (définition 4.6). Cette portée est localisée sur l'ensemble des monômes donnant naissance au noyau.

4.4.3 Calcul du gain d'un candidat diviseur

- **Gain des noyaux algébriques**

Soit NbMon(K) le nombre de monômes du noyau K, soit NbLit(C) le nombre de littéraux du conoyau C associé à K et NbLit(K) le nombre de littéraux du noyau K.

1- Calcul de gain pour les noyaux locaux:

Le noyau apparaît dans une seule fonction Booléenne, le gain défini dans le paragraphe I.2.2 est donc:

$$\text{Gain(K)} = \sum_{i=1}^{m} \left((\text{NbMon(K)} - 1).\text{NbLit}(C^i)\right) + (m-1).\text{NbLit(K)}$$

Où $(C_i)_{i=1..m}$ sont les conoyaux associés au noyau K.

2- Calcul de gain pour les noyaux globaux:

Le noyau K est un noyau de plusieurs fonctions Booléennes. Si ce noyau apparaît *p* fois dans l'ensemble des fonctions, le gain en nombre de littéraux associé est le suivant:

$$Gain(K) = (p-1).(NbLit(K)) - p$$

La formule générale du gain associé à un noyau K devient:

$$Gain(K)=\sum_{i=1}^{m} ((NbMon(K) - 1).NbLit(C^i))+(m-1).NbLit(K)+ (p-1).(NbLit(K)) - p$$

Exemple:

$F_1 = a.b.c + a.b.d + a.e.f + g$

$F_2 = a.c + a.d + g.h$

$K_0 = c + d$ noyau global conoyaux: a.b (F_1) et a (F_2)

$K_1 = b.c + b.d + e.f$ noyau local conoyau: a (F_1)

Gain (K_0) = ((2 - 1).2) + (2 - 1).1)) + 1.(2) - 2 = 3

Gain (K_1) = (3 - 1).1 = 2

- **Gain des monômes ou parties de monômes communs**

On appelle:

- NbLit(m): le nombre de littéraux du monôme m,
- NbOcc(m): le nombre d'occurrences du monôme m.

La formule du gain associé à un monôme est la suivante:

$$Gain(m) = (NbOcc(m) - 1).(NbLit(m) -1) - 1$$

Exemple:

$F_1 = a.b.c.d + d e + h$

$F_2 = a.b.c.e + d.e + h$

m_1 = a.b.c est une partie commune de monôme dont le gain associé est 1.

m_2 = d.e est un monôme commun, son gain est égal à 0.

- **Principe de la division restreinte par les conoyaux:**

Supposons qu'un noyau candidat K_i soit sélectionné suivant la fonction gain (définie dans 4.4.3). Soient les triplets (C_i^j, K_i, ξ_i^j) associés à ce noyau. Le résultat de la division restreinte s'écrit sous la forme suivante:

$$F = \left[\sum_j C_I^j \right] . K_i + R$$

Où le reste R est la somme des monômes restants de F ($M_R = M_F - \xi_i^j$).

Pour illustrer la différence entre la méthode de division et substitution algébrique et la méthode de division restreinte, nous considérons l'exemple suivant:

Soit la fonction F: $F = a.b.c + a.\overline{b}.\overline{c} + a.d + b.c.e.f.g + \overline{b}.\overline{c}.e.f.g$

La liste des noyaux, conoyaux et gains associés est:

noyaux	conoyaux	gain
$K_1 = a.b.c + a.\overline{b}.\overline{c} + a.d + b.c.e.f.g + \overline{b}.\overline{c}.e.f.g$	1	0
$K_2 = b.c + \overline{b}.\overline{c} + d$	a	2
$K_3 = a + e.f.g$	b.c	2
$K_4 = a + e.f.g$	$\overline{b}.\overline{c}$	2
$K_5 = b.c + \overline{b}.\overline{c}$	e.f.g	3

L'application de la méthode de division algébrique sur le noyau de plus grand gain (K_5) donne le résultat suivant:

$$F = (b.c + \overline{b}.\overline{c}).(e.f.g + a) + a.d$$

Pour la méthode de division restreinte par les conoyaux, après le premier choix de (C_5,K_5) la fonction F s'écrit:

$$F = e.f.g.(b.c + \overline{b}.\overline{c}) + a.b.c + a.\overline{b}.\overline{c} + a.d$$

On choisit ensuite le couple (C_2,K_2) et la fonction F devient:

$$F = e.f.g.(b.c + \overline{b}.\overline{c}) + a.(b.c + \overline{b}.\overline{c} + d)$$

Cette méthode de division par les conoyaux ne permet pas d'obtenir une mise en commun de la somme de monômes ($b.c + \overline{b}.\overline{c}$) sans passer par l'étape de recherche de monômes et de parties de monômes communs. Nous verrons dans la suite de ce chapitre, qu'à partir d'une certaine complexité des fonctions Booléennes, cette méthode s'avère très intéressante car si elle ne perturbe que localement l'ensemble des fonctions, elle permet une optimisation plus globale de ces fonctions

L'algorithme de division restreinte par les conoyaux est le suivant:

division_par_conoyaux (F, C^j)

- *pour tous les conoyaux C^j associés à un noyau K faire*

  ξ_j = *Ensemble des monômes de F multiples de* C^j

fin pour

- $M_E = \cup M\xi_j$
- $M_R = M_F - M_E$ *(R est le reste de la division de F par les conoyaux* C^j *)*
- $F = \left[\sum_j C^j\right].K + R$ *(mise en commun de K)*

Cette approche peut être illustrée par l'arbre factorisé (figure 4.1) construit à partir des triplets (C_i^j, K_i, x_i^j) déterminés par l'algorithme de recherche de noyaux. L'arbre factorisé est obtenu itérativement à chaque étape.

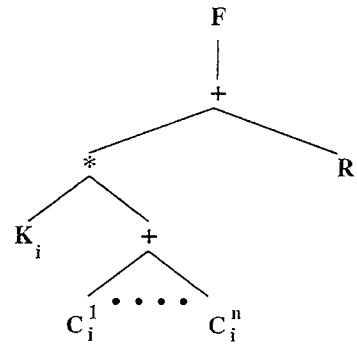


Figure 4.11: Arbre factorisé après la mise en facteur de K_i

Exemple:

La fonction $F = e.f.g.(b.c + \overline{b}.\overline{c}) + a.(b.c + \overline{b}.\overline{c} + d)$, factorisée dans l'exemple précédent est représentée sous la forme d'un arbre (figure 4.1).

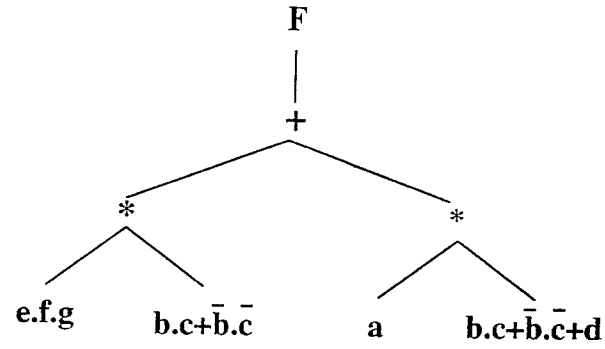


Figure 4.12: Arbre factorisé de F.

4.4.3 Comparaison du gain de la division restreinte à celui de la division algébrique classique

Soit F une fonction Booléenne et supposons qu'à une étape donnée, un noyau K a été choisi. Soient C_i les conoyaux associés à K. Si on applique la méthode de division par les conoyaux que l'on note (m1), le gain est le suivant:

$$Gain_{m1}(K) = \sum_{i=1}^{I} (NbMon(K)-1).NbLit(C_i) + (I-1).NbLit(K)$$

Si on applique la méthode de division et de substitution algébrique (m2), le gain devient:

$$Gain_{m2}(K) = Gain_{DivNo}(K) + Gain_{DivNoComp}(K) + Gain_{global}(K)$$

- $Gain_{DivNo}(K)$ est le gain apporté par la division algébrique par le noyau K.

$$Gain_{DivNo}(K) = \sum_{j=1}^{J} (NbMon(K)-1).NbLit(C_j) + (J-1).NbLit(K)$$

Les (C_j) tels que $(J \geq I)$ sont les cofacteurs du noyau K, les (C_i) sont les conoyaux de K et les (C_k) tels que $(I < k \leq J)$ sont les conoyaux associés aux noyaux K_l où $K \subset K_l$, d'où

$$Gain_{DivNo}(K) \geq Gain_{m1}(K)$$

- $Gain_{DivNoComp}(K)$ est le gain apporté par la division algébrique par le complément du noyau.

$$Gain_{DivNoComp}(K) = \sum_{i} (NbMon(\overline{K})-1).NbLit(C_i(\overline{K}))$$

$C_i(\overline{K})$ sont les cofacteurs du complément du noyau K, $Gain_{DivNoComp}(K) \geq 0$. Ce gain peut être nul si $\overline{K}$ est un monôme, c'est à dire que le noyau K est une somme de monômes de degré 1.

- $Gain_{global}(K)$ est le gain apporté par la sous-fonction correspondant au noyau K si la fonction F est divisible par $\overline{K}$.

$$Gain_{global}(K) = NbLitExp(\overline{K}) - 2$$

Comme $NbLitExp(\overline{K}) \geq 2$ et $NbLitExp(K) \geq 2$; le gain $Gain_{global}(K) \geq 0$.

La méthode fondée sur la division algébrique et la substitution permet donc d'augmenter le gain en littéraux à chaque étape pour obtenir une minimisation locale du nombre de littéraux et ne pas impliquer forcément une réduction globale. En effet, comme signalé dans §4.4.1, la division plus forte par le noyau et son complément, restreint d'autant plus le choix des diviseurs restants ce qui risque d'éliminer des candidats ultérieurs intéressants.

L'estimation de la fonction gain est exacte (celle calculée au moment de la génération des noyaux) dans la méthode de division par les conoyaux. C'est celle qui a été calculée dans $Gain_{m1}(K)$. Dans la division et la substitution algébrique, le $Gain_{m2}(K)$ ne peut être estimé au stade de la génération des noyaux, la valeur exacte ne peut être calculée que si on fait effectivement la division algébrique et la substitution, ce qui ne peut être envisageable pour tout noyau candidat. On se contente donc de faire la sélection du noyau K selon $Gain_{m1}(K)$ qui constitue une limite inférieure de la fonction gain.

4.4.4 Sélection et filtrage des candidats diviseurs

4.4.4.1 Sélection des candidats diviseurs

Cette étape permet de choisir dans l'ensemble des candidats diviseurs, déterminés lors de la phase de génération, celui qui impliquera un gain maximal de logique, exprimé en nombre de littéraux. Ce choix doit tenir

compte de l'incompatibilité algébrique éventuelle entre les différents candidats, ce qui rend complexe le problème de sélection des diviseurs.

Une première méthode considère globalement l'ensemble des candidats pour tenir compte de l'impact créé par le choix de l'un d'entre eux sur l'ensemble des candidats restants. Cette méthode aboutit à l'extraction d'un ensemble de candidats mutuellement compatibles et de gain maximal. Ce problème peut être modélisé de la façon suivante :

On construit le graphe non orienté G qui a pour sommets l'ensemble D des diviseurs. Deux sommets d_1 et d_2 sont liés par une arête si les candidats diviseurs correspondants sont incompatibles. On donne un poids à chaque sommet qui correspond au gain du diviseur associé.

Trouver l'ensemble de tous les diviseurs de gain maximum revient exactement à résoudre le problème de la recherche du stable de poids maximum dans G. Ce problème est NP-complet, et donc la recherche d'un ensemble de candidats diviseurs compatibles deux à deux et de gain maximal ne peut pas être résolu par un algorithme polynomial.

Une heuristique, appelée principe de la couverture rectangulaire, fondée sur la couverture disjointe des cellules d'une matrice, est proposée dans [Bra87c]. Cette méthode consiste à construire une matrice M où chaque ligne correspond à un unique conoyau d'un noyau et où chaque colonne est associée à un unique monôme d'un noyau de la fonction. Ainsi un monôme de la fonction initiale est une case de cette matrice.

Bien que la formulation de cette méthode semble intéressante, il s'avère que la complexité actuelle des circuits ne permet pas toujours de l'appliquer de manière optimale. La méthode de factorisation étudiée est de type glouton. Elle sélectionne le candidat permettant d'obtenir un gain local maximal et élimine de la liste des candidats ceux qui ne sont plus compatibles après le choix de ce dernier. Cette mise à jour permet de garder uniquement les candidats diviseurs compatibles algébriquement avec ceux déjà choisis.

Exemple de sélection de conoyaux/noyaux:

F = a.b.c.n.o.p + a.b.c.d.e + d.e.h.i.j + h.i.j.k.l.m

(C_1,K_1) = (a.b.c, n.o.p + d.e) gain = 3

(C_2,K_2) = (d.e, a.b.c + h.i.j) gain = 2

(C_3,K_3) = (h.i.j, d.e + k.l.m) gain = 3

La méthode gloutonne se déroule de la façon suivante:

- Choix de (C_1,K_1)
- La mise à jour élimine le couple (C_2,K_2)
- Choix de (C_3,K_3)

L'expression factorisée de F est:

F = a.b.c (n.o.p + d.e)+ h.i.j.(d.e + k.l.m)

L'algorithme de sélection des candidats diviseurs est le suivant:

Tant qu'il existe des candidats diviseurs faire

- *{candidats} = {noyaux globaux ou locaux}∪ {monômes et parties de monômes communs}*
- *tri des candidats par gain décroissant.*
- *choix du meilleur candidat.*
- *division par le candidat sélectionné.*
- *mise à jour de la liste des candidats diviseurs.*

fin tant que

4.4.4.2 Filtrage des candidats diviseurs

Le filtrage des candidats diviseurs est fait selon deux critères:

- l'optimisation de la structure de la connectique en limitant la taille des sous-fonctions choisies.

- l'optimisation de la profondeur des fonctions Booléennes factorisées.

Limitation de la taille des sous-fonctions

Ce premier critère permet de choisir dans l'ensemble des candidats diviseurs lors de la phase de génération ceux dont la taille, en terme de nombre de littéraux, est supérieure à une valeur donnée k. Nos expériences montrent que la taille critique des candidats diviseurs est de 3 pour une implantation sur

LUT de 4 entrées. Les sous-fonctions dont la taille est inférieure à cette valeur sont rejetées lors de la phase de génération de candidats diviseurs. En effet, ces sous-fonctions sont en général partagées par un grand nombre de fonctions. Les cellules correspondantes risquent d'avoir une sortance très élevée et amènent une augmentation importante de la surface de routage. Le filtrage des sous-fonctions acceptées permet un bon contrôle de la connectique et une bonne préparation du chemin critique.

Optimisation de la profondeur

En partant d'un ensemble de fonctions Booléennes représentées sous forme polynomiale, les candidats diviseurs sont choisis de manière à ne pas augmenter la profondeur initiale du réseau. On peut appliquer la méthode de filtrage des candidats fondée sur la profondeur prédite avec la formule de calcul de profondeur introduite dans [Gol76]:

$$P_N = \left\lfloor \log_k \sum_{i=1}^{n} k^{P_i} \right\rfloor$$

- n: nombre d'entrées du noeud N.
- P_i: profondeur prédite de l'entrée i.
- k: nombre maximal d'entrées de la cellule (LUT).

Malheureusement, la profondeur prédite est toujours calculée après la division algébrique et ne peut l'être avant cette division. En conséquence, pour accepter ou rejeter un candidat diviseur, on est obligé d'effectuer les divisions algébriques par ce candidat pour calculer les profondeur prédites des arbres résultant de ces divisions. Cette méthode s'applique seulement sur un ensemble d'expressions de faible complexité [Sak93].

Pour limiter la profondeur, il nous faut une formule qui reflète la profondeur et qui peut être calculée sans faire la division algébrique. Pour cela nous exploitons la borne supérieure de la complexité d'un arbre factorisé proposée dans [Mur92]:

- Pour un LUT avec k=5 on a: $C_5(f) \leq \left\lfloor \frac{2^l - 1}{5} \right\rfloor$ (avec $l \geq 4$)

$C_5(f)$: nombre de LUT 5 pour implanter la fonction Booléennes f (complexité de l'expression de f).

• l: nombre de littéraux de f.

- Pour un LUT avec k=4 on a: $C_4(f) \leq \left\lfloor \frac{l-1}{2} \right\rfloor$ (avec $l \geq 3$).

L'idée est de rejeter un candidat si l'un des monômes du noyau a une complexité égale à la complexité du monôme correspondant avant la division. Cette heuristique élimine le risque d'augmenter la profondeur en créant des noeuds profonds avec une complexité égale aux noeuds les moins profonds dans l'arbre.

Exemple:

F=a.b.c.d+a.e.f+b.x+x.h ;

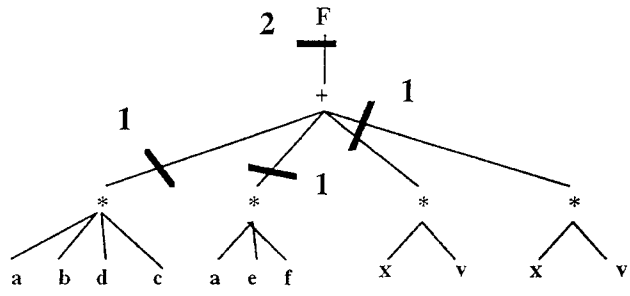
L'expression résultant de la division de F par le noyau N_1=(b.c.d+e.f) est:

F=a. (b.c.d+e.f)+b.x+x.h ;

nous avons les monômes:

- m_1 = a.b.c.d $C_4(m)$: 1
- m'_1 = b.c.d $C_4(m)$: 1

donc la division par N_1 est rejetée, la figure 4.13 montre l'augmentation de la profondeur par la division par N_1; F auras une profondeur de 3 au lieu de 2 (LUT de 4 entrées).



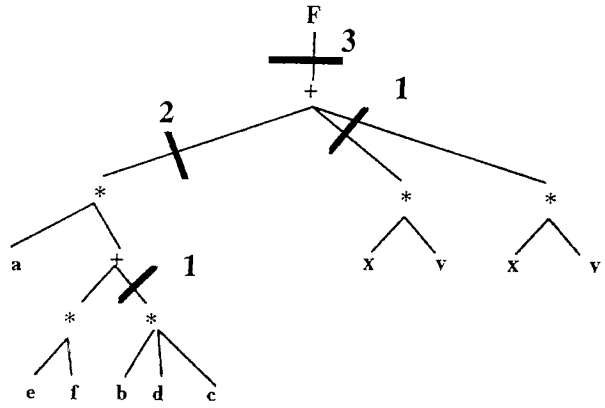


Figure 4.13: Effet de la factorisation sur la profondeur finale.

4.4.5 Discrétisation des étapes de factorisation

La méthode de factorisation présentée considère deux types de candidats:

- les noyaux.

- les monômes et les parties de monômes communs.

Une méthode de discrétisation, traitant alternativement ces deux types de candidats, utilise l'algorithme suivant:

Soit la suite décroissante d'entiers $S = \{s_0, s_1, \ldots, s_n = 0\}$ représentant une suite de valeurs du gain.

$S = s_0$
Tant qu'il existe des candidats faire
• *Division par les conoyaux ou les noyaux et leurs compléments:*
Division par des candidats (des noyaux de gain≥ S)
• *Recherche des monômes ou parties de monôme communs:*
Recherche de parties communes de gain ≥ S
• *S= Successeur de S*
fin tant que

Etant donné que les deux types de candidats choisis sont confrontés à une éventuelle incompatibilité algébrique, cette méthode de discrétisation ne favorise aucun type de candidats et donne une bonne vision globale. Dans l'algorithme de factorisation, nous avons adopté une suite de valeurs du gain obtenue d'une façon empirique. Cette suite correspond aux valeurs suivantes:

$$S = \{100, 90, 80, 70, 60, 50, 40, 30, 20, 14, 8, 1, 0\}$$

L'exemple suivant illustre l'incompatibilité qui peut avoir lieu entre les deux types de candidats diviseurs.

Exemple:

$F_1 = a.b.c.d + a.b.e.f + h$

$F_2 = a.b.c.d + c.d.k + j$

Les noyaux des deux fonctions Booléennes sont:

$K_1 = c.d + e.f$ gain $(K_1) = 2$

$K_2 = a.b + k$ gain $(K_2) = 2$

Si on divise par K_1 et K_2, l'ensemble des fonctions devient:

$F1 = a.b.(c.d + e.f) + h$

$F2 = c.d.(a.b + k) + j$

Cette division est incompatible avec la recherche des monômes ou parties de monômes communs qui aurait donné le monôme commun a.b.c.d:

$SF = a.b.c.d$

$F_1 = SF + a.b.e.f + h$

$F_2 = SF + c.d.k + j$

4.5 Résultats expérimentaux et évaluation de l'effet des méthodes de factorisation

Dans ce paragraphe nous allons comparer les différentes méthodes de factorisations proposées. Toutes ces techniques de factorisation et de décomposition ont été implantées dans le système de synthèse logique ASYL+ [Abo94][Asy95]. Pour la décomposition de l'arbre de la factorisation restreinte, nous utilisons l'algorithme optimisant le remplissage des cellules présenté dans 3.4.1. Les comparaisons ont été faites dans le cadre de l'outil ASYL+. Les exemples utilisés sont les benchmarks internationaux [MCN89] et des circuits industriels développés spécifiquement pour FPGA.

4.5.1 comparaison des résultats sur l'optimisation de la surface

Le tableau 4.1 présente les résultats de l'optimisation du nombre de CLB pour la série 3000 obtenus par les trois méthodes de factorisation. Ces résultat

montrent que la factorisation algébrique restreinte est plus efficace que les deux autres méthodes pour l'optimisation en surface. La factorisation Algébrique restreinte donne le meilleur résultat dans 50% des cas, contre 41% pour la factorisation lexicographique et 23% pour la factorisation à base de ROBDD. On remarque que le gain moyen en nombre de CLB de la factorisation algébrique restreinte est de 8.1% par rapport à la factorisation lexicographique et de 40.5% par rapport à la factorisation à base de ROBDD. Pour les exemples de grande complexité la factorisation à base de ROBDD ne donne pas de bons résultats par contre pour les exemples symétriques, elle est la plus appropriée. Pour les exemples industriels de grande complexité, la factorisation lexicographique s'avère aussi efficace que la factorisation algébrique restreinte pour l'optimisation de la surface. Pour certains exemples on ne donne pas le résultat de la factorisation à base de ROBDD ceci est dû au script implanté dans le système ASYL, qui arrête le traitement si le résultat ne peut être meilleur que les résultats déjà obtenus ou que la taille du ROBDD dépasse une certaine limite de taille mémoire. L'écart entre la factorisation algébrique restreinte et la factorisation lexicographique est relativement faible. L'histogramme 4.1 montre l'évolution des résultats des trois méthodes de factorisation suivant la complexité des exemples.

		ALG		LEX		ROBDD	
	RO	CLB	Cp	CLB	Cp	CLB	Cp
xrand2o	A	11	2	**10**	1	**10**	1
hexdec	A	**11**	2	**11**	2	13	3
dk15	M	**12**	3	13	3	**12**	3
kwwcnto	A	18	5	18	5	**17**	5
addpar	X	22	5	27	5	**21**	5
misex2	M	**23**	4	28	4	32	3
bestl	X	29	3	25	4	**24**	3
adrcnt	A	25	8	**25**	6	56	9
sao2	M	**26**	6	36	6	37	7
vertical	A	32	4	**31**	5	52	4
tbk	M	**73**	7	95	9	90	8
jay	M	**74**	5	77	6	79	7
platcos	M	**84**	3	87	3	134	8
planet	M	**115**	7	123	6	155	9
cache_ctrl	M	**168**	6	211	7	313	15
scf	M	**181**	9	194	7	223	9
biw	X	223	21	**207**	17	g	
corn	X	239	2	**223**	1	m	
art_top	X	287	13	**280**	12	m	
mike	X	320	22	**309**	21	m	
mystack	X	**329**	21	331	21	m	
top	X	346	10	**335**	6	m	
Meilleur resultat	**%**	**50**		**41**		**23**	

Tableau 4.1: Comparaison en surface (série 3000).

CLB: Nombre de CLB.

CP: Profondeur en terme d'imbrication de sous fonctions.

OR: Origine de l'exemples:

M: Benchmark MCNC.

A: Exemple industriel provenant d'Actel.

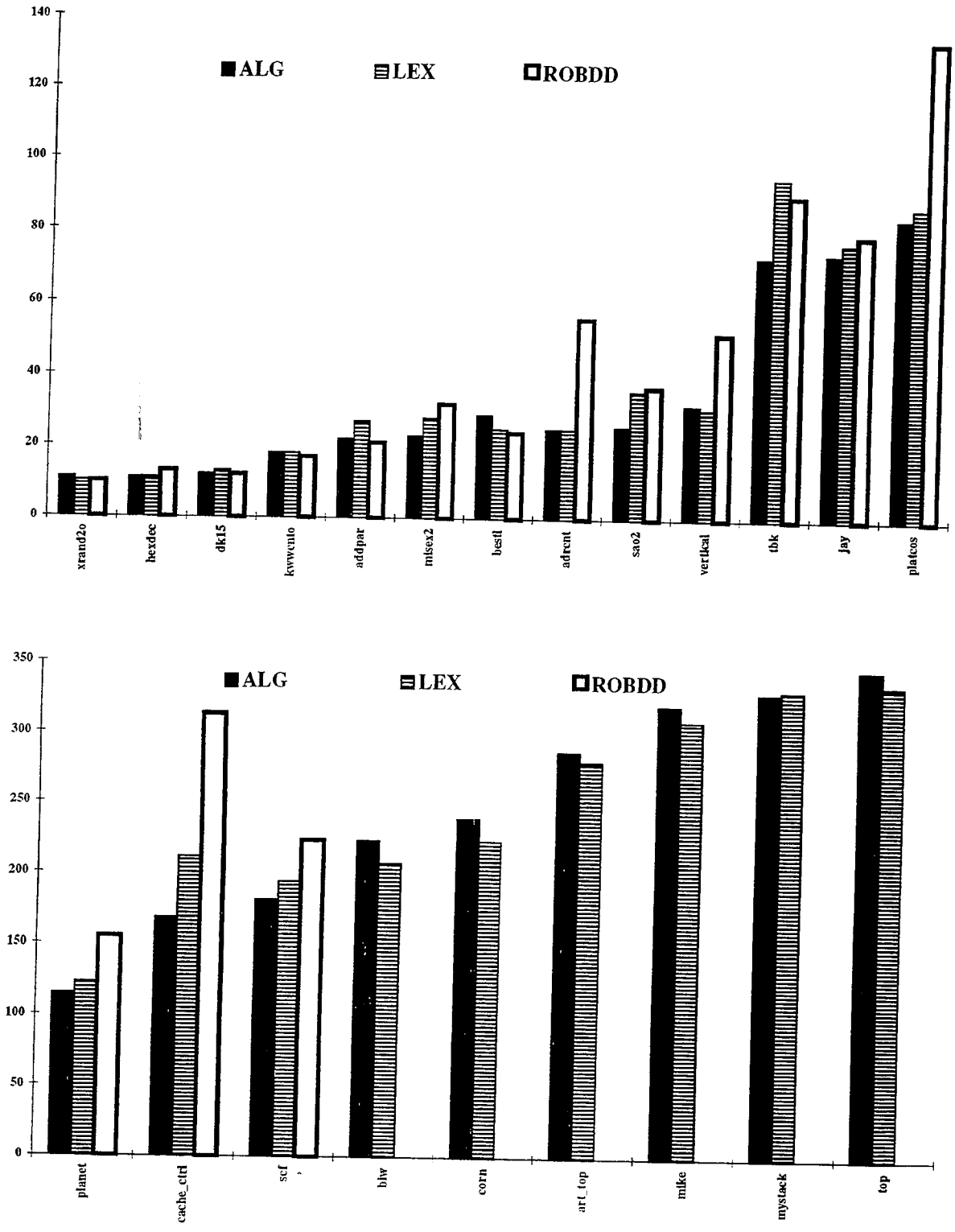
X: Exemple Industriel conçu directement sur FPGA Xilinx.

Construction ROBDD:

m: l'execution est arrêtée à cause de la taille mémoire du ROBDD.

g: la décomposition est arrêtée à cause du mauvais résultat.

Ces notations sont adoptées dans les tableaux suivants.



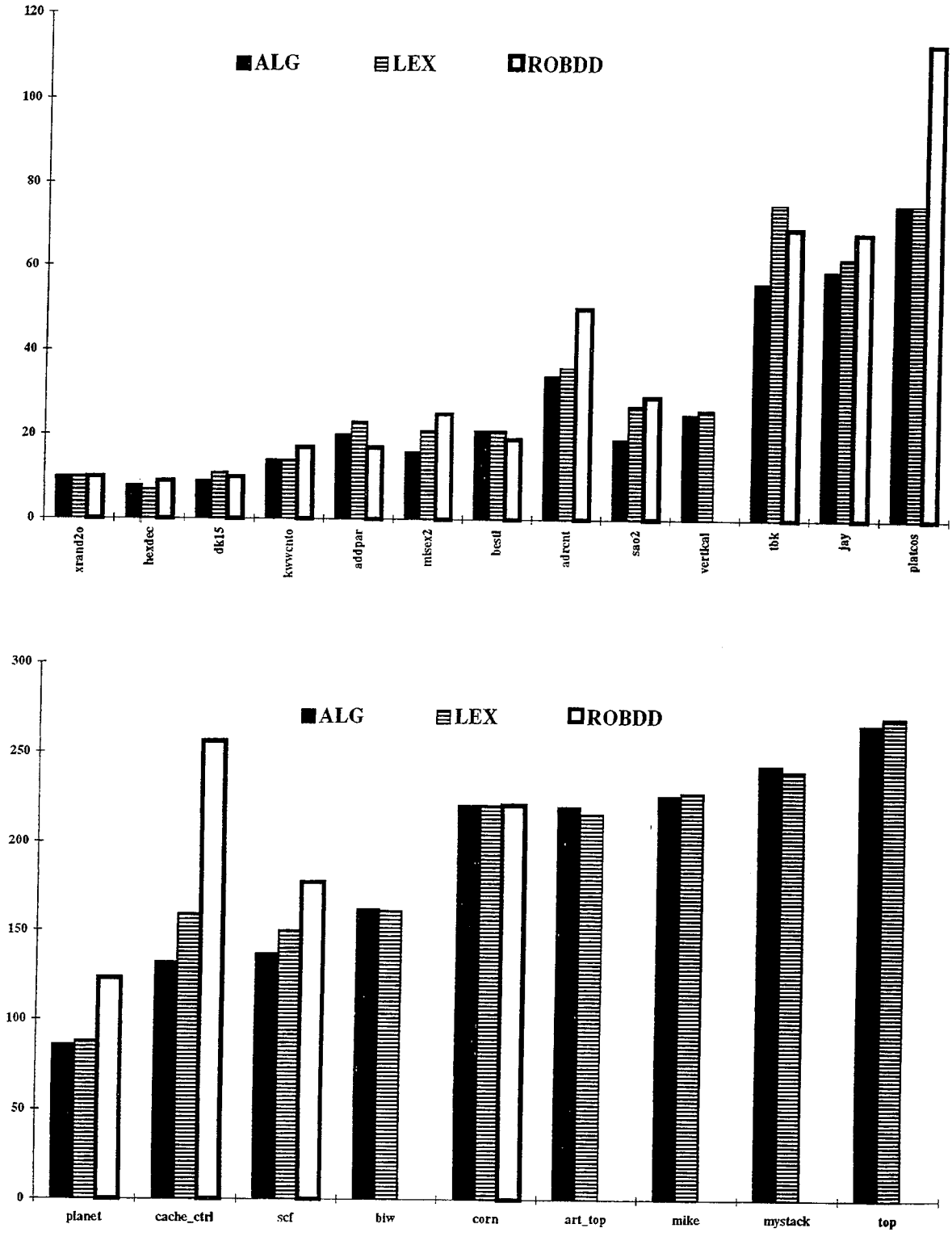
Graphe 4.1: Optimisation en surface série 3000

Dans le tableau 4.2 et le graphe 4.2, nous présentons les résultats de l'évaluation de l'optimisation orientée surface pour les FPGA xilinx série 4000.

On remarque que les pourcentages varient très peu, ainsi les conclusions de la série 3000 sont valables pour la série 4000.

		ALG	LEX	ROBDD
	Or.	CLB	CLB	CLB
xrand2o	A	**10**	**10**	**10**
hexdec	A	8	**7**	9
dk15	M	**9**	11	10
kwwcnto	A	**14**	**14**	17
addpar	X	20	23	**17**
misex2	M	**16**	21	25
bestl	X	21	21	**19**
adrcnt	A	**34**	36	50
sao2	M	**19**	27	29
vertical	A	**25**	26	g
tbk	M	**56**	75	69
jay	M	**59**	62	68
platcos	M	**75**	**75**	113
planet	M	**86**	88	123
cache_ctrl	M	**132**	159	256
scf	M	**137**	150	177
biw	X	162	**161**	g
corn	X	**221**	**221**	**221**
art_top	X	220	**216**	m
mike	X	**226**	228	m
mystack	X	243	240	m
top	X	**266**	270	m
Meilleur resultat	**%**	**76**	**33**	**19**

Tableau 4.2: Optimisation en surface (série 4000).



Graphe 4.2: Optimisation en surface (série 4000).

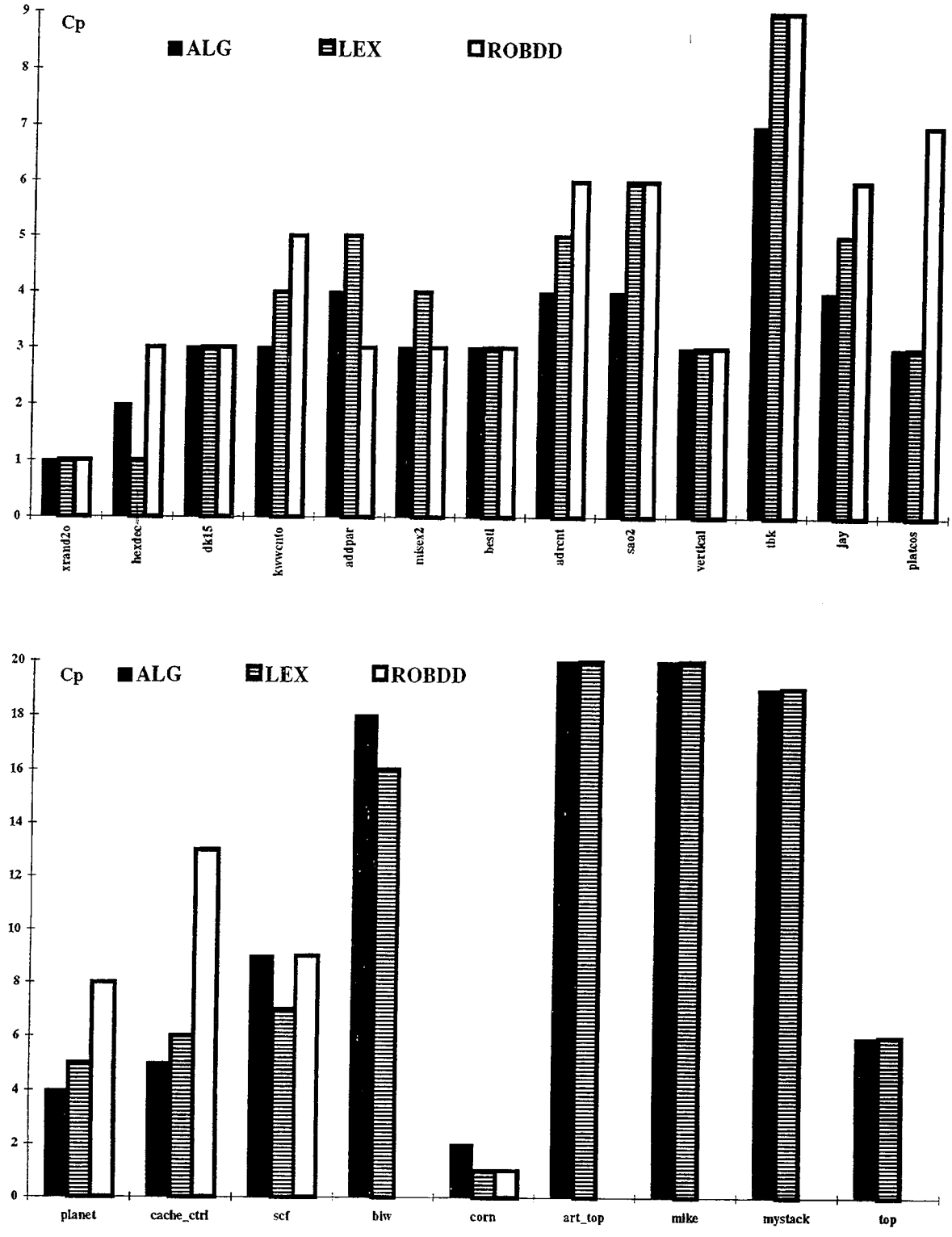
4.5.2 comparaison des résultats sur l'optimisation du chemin critique

Les résultats de l'optimisation du chemin critique obtenus par les trois méthodes de factorisation sont résumés dans le tableau 4.3 et le graphe 4.4. Le

meilleur résultat en profondeur est obtenu dans 57% des cas par la factorisation algébrique restreinte, dans 33% des cas par la factorisation lexicographique et dans 24% des cas par la factorisation à base de ROBDD. Le meilleur résultat en profondeur (tableaux 4.1) de l'optimisation surface est obtenu dans 41% des cas par la factorisation algébrique, dans 45% des cas par la factorisation lexicographique et dans 27% des cas par la factorisation à base de ROBDD. On précise ici que la factorisation lexicographique et la factorisation à base de ROBDD sont strictement les mêmes pour les deux options d'optimisation. Ces pourcentages qui se sont inversés au profit de la factorisation algébrique démontrent l'efficacité du filtre orienté chemin critique pour le choix des candidats diviseurs dans la factorisation algébrique restreinte. Un choix de l'ordre des variables par l'utilisateur suivant la structure de l'exemple peut modifier les résultats de la factorisation lexicographique et de la factorisation à base de ROBDD. On remarque sur l'histogramme 4.3 l'efficacité de la factorisation lexicographique sur les exemples industriels complexes.

		ALG		LEX		ROBDD	
	Or	CLB	Cp	CLB	Cp	CLB	Cp
xrand2o	A	**10**	**1**	**10**	**1**	**10**	**1**
hexdec	A	11	2	**8**	**1**	10	3
dk15	M	**12**	3	13	3	**12**	**3**
kwwcnto	A	**27**	3	19	4	15	5
addpar	X	44	4	28	5	**21**	**3**
misex2	M	**27**	**3**	28	4	33	3
bestl	X	32	3	27	3	**18**	**3**
adrcnt	A	**29**	**4**	27	5	58	6
sao2	M	**45**	**4**	36	6	39	6
vertical	A	36	3	**34**	**3**	57	3
tbk	M	**73**	**7**	95	9	90	9
jay	M	**94**	**4**	85	5	76	6
platcos	M	**86**	**3**	87	3	138	7
planet	M	**141**	**4**	132	5	157	8
cache_ctrl	M	**286**	**5**	231	6	351	13
scf	M	181	9	**194**	**7**	223	9
biw	X	247	18	**217**	**16**	-	-
corn	X	243	2	**223**	1	**223**	**1**
art_top	X	**322**	**20**	325	20	-	-
mike	X	365	20	**325**	**20**	-	-
mystack	X	364	19	**339**	**19**	-	-
top	X	395	6	**343**	**6**	-	-
Meilleur resultat	**%**		**55**		**41**		**23**

Tableau 4.3: Optimisation orientée chemin critique (série 3000).



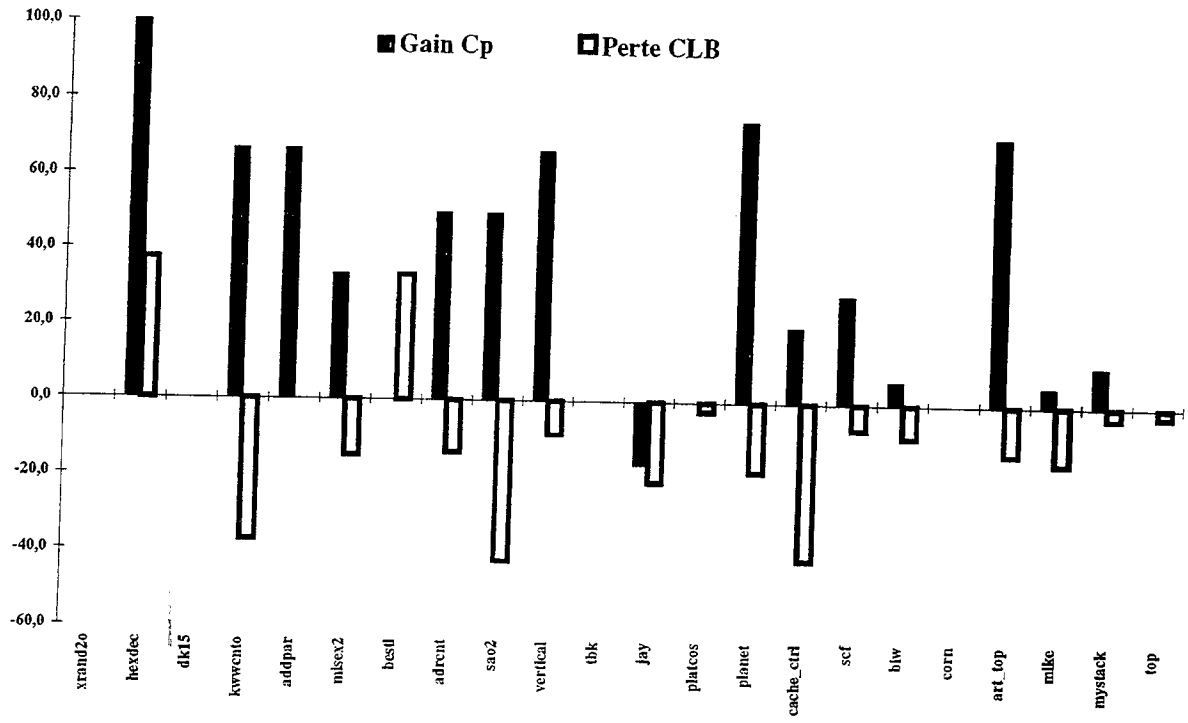
Graphe 4.3: Evaluation des factorisations pour l'optimisation orientée chemin critique (série 3000).

Dans le tableau 4.4 on rapporte le meilleur résultat de l'optimisation orientée surface et celui de l'optimisation orientée chemin critique. L'histogramme du graphe 4.4 représente les pourcentages des gains en

profondeur et les pertes correspondantes en nombre de CLB. Nous remarquons que le gain en profondeur entraîne une augmentation modérée du nombre des CLB ce qui est très important sachant que pour les FPGA Xilinx, comme nous le constaterons plus loin, que la profondeur en terme d'imbrication des sous fonctions n'est qu'une évaluation du chemin critique dans des conditions très strictes et que le chemin critique peut augmenter considérablement avec la complexité du placement routage et donc en fonction du nombre de CLB et de connexions.

		Opt. C.C.		Opt.	Surf.	Gain Cp	Perte CLB
	Or	CLB	Cp	CLB	Cp	Cp	CLB
xrand2o	A	10	1	10	1		
hexdec	A	8	1	11	2	**100,0**	**37,5**
dk15	M	12	3	12	3		
kwwcnto	A	27	3	17	5	**66,7**	**-37,0**
addpar	X	21	3	21	5	**66,7**	
misex2	M	27	3	23	4	**33,3**	**-14,8**
bestl	X	18	3	24	3		**33,3**
adrcnt	A	29	4	25	6	**50,0**	**-13,8**
sao2	M	45	4	26	6	**50,0**	**-42,2**
vertical	A	34	3	31	5	**66,7**	**-8,8**
tbk	M	73	7	73	7		
jay	M	94	6	74	5	**-16,7**	**-21,3**
platcos	M	86	3	84	3		**-2,3**
planet	M	141	4	115	7	**75,0**	**-18,4**
cache_ctrl	M	286	5	168	6	**20,0**	**-41,3**
scf	M	194	7	181	9	**28,6**	**-6,7**
biw	X	227	16	207	17	**6,3**	**-8,8**
corn	X	223	1	223	1		
art_top	X	322	7	280	12	**71,4**	**-13,0**
mike	X	365	20	309	21	**5,0**	**-15,3**
mystack	X	339	19	329	21	**10,5**	**-2,9**
top	X	343	6	335	6		**-2,3**
Moyenne						**28,8**	**-8,1**

Tableau 4.4: Comparaison de l'optimisation surface et de l'optimisation chemin critique (série 3000).



Graphe 4.4: Comparaison de l'optimisation surface et de l'optimisation chemin critique (série 3000).

4.5.3 Comparaison après placement routage

Le tableau 4.5 représente les résultats après placement routage pour des exemples qui posent des problèmes réels de routage. Le placement routage est fait par le programme APR V5.0.0 des outils Xilinx. Le routage est fait par trois itérations (option par défaut de APR V5.0.0).

Le graphe 4.5 montre le gain en temps CPU de routage et le gain en nombre de connexions non routées après trois itérations du programme de routage. Nous remarquons que pour les exemples qui sont entièrement routés pour les deux factorisations, le gain en temps CPU pour la factorisation lexicographique est important ce qui confirme la contribution de la factorisation lexicographique à la facilité du routage.

Pour l'ensemble des exemples, nous avons, dans le cas de la factorisation lexicographique un gain moyen de 51.4% des connexions non routées par rapport à la factorisation algébrique restreinte. Le gain moyen en temps CPU de routage est de 38.2%. Ces résultats confirment l'efficacité de la factorisation

lexicographique pour résoudre les problèmes de routage. Dans certains cas, si la factorisation lexicographique donne des résultats avec un nombre de CLB beaucoup plus grand que celui de la factorisation algébrique, le résultat de placement routage n'est plus au profit de la factorisation lexicographique mais si on impose un ordre des variables pour la factorisation lexicographique, on peut améliorer le résultat en nombre de CLB et par conséquent le routage.

	Algebrique				Lexico					
	CLB	% Ut	CPU	N.R	CLB	% Ut	CPU	N.R	Gain CPU	Gain NR
Jay	74	74	2339		77	77	1199		49	
planet	115	79,8611	3329	572	123	85,417	2465	46	26	92
dk16	50	78,125	649	1	57	89,063	753		-16	100
2051-4	170	75,8929	1889	195	173	77,232	2017	173	-7	11
s1	73	73	2417	7	78	78	2124		12	100
scf	182	81,25	15522	204	195	87,054	14853	176	4	14
								moyenne	11	63

Tableau 4.5: Effet de la factorisation sur le routage.

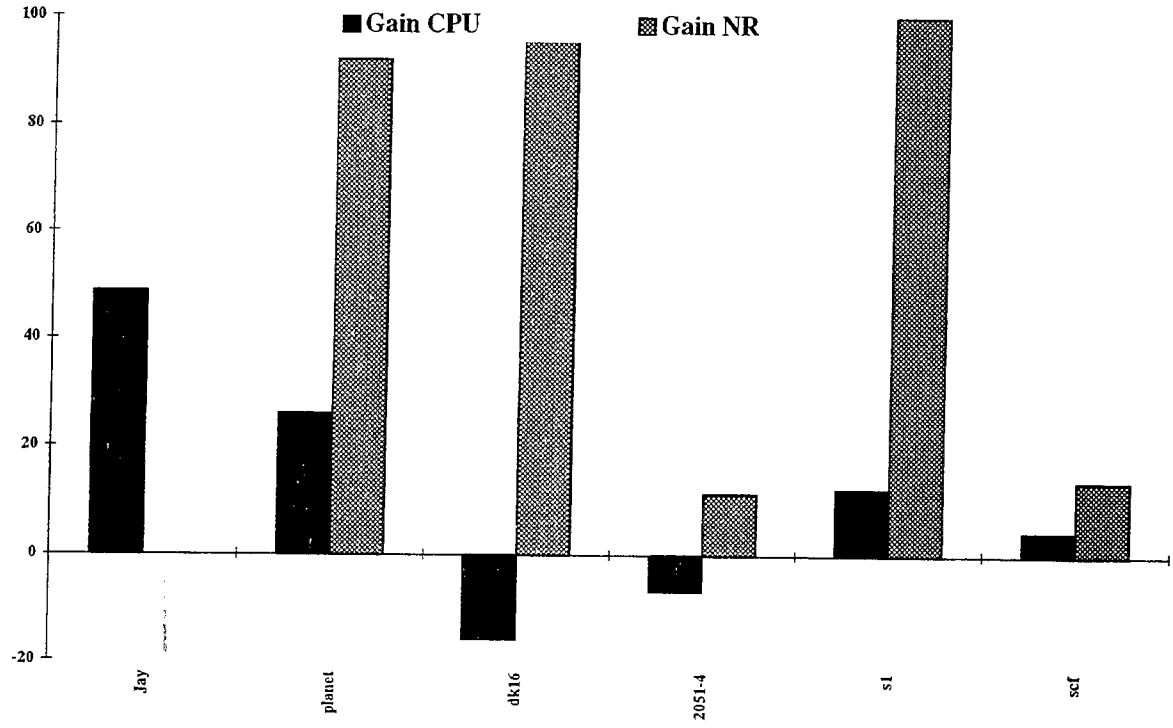
IOB: Nombre d'entrées/sorties.

CLB: Nombre de CLB utilisés.

%Ut: Pourcentage de remplissage; pourcentage des CLB utilisés dans le circuit choisi.

NR: Nombre des connexions non routées.

CPU: Temps CPU pour le routage seulement (Sparc 10).



Graphe 4.5: Comparaison CPU de routage et connexions non routées entre la factorisation Algébrique et la factorisation Lexicographie.

4.6 Conclusions

Une première alternative à la factorisation lexicographique est la factorisation fondée sur le diagramme de décision binaire (ROBDD). Cette factorisation présente les mêmes caractéristiques que celles de la factorisation lexicographique, à savoir l'ordonnancement des variables et la structure en cônes logiques. L'originalité de cette approche est d'utiliser l'ordre des variables construit par la factorisation lexicographique permettant ainsi une construction rapide du ROBDD. On peut donc remarquer que l'approche lexicographique est non seulement un précurseur de l'approche ROBDD mais de plus, elle permet à cette dernière de devenir applicable.

La deuxième alternative est la factorisation algébrique restreinte, pour laquelle nous avons développé des filtres spécifiques à la cible technologique et aux critères d'optimisation (filtre orienté chemin critique et filtre orienté surface).

Chapitre V

Regroupement et optimisation des Ressources

5.1 Introduction

Après avoir réalisé la décomposition technologique des fonctions Booléennes, il convient de traiter également les problèmes d'insertion de buffer, des signaux globaux de contrôle et des entrées sorties. Enfin, pour réussir une synthèse d'un réseau Booléen, il convient, au cours de la décomposition technologique de préparer l'étape de placement routage. Il s'agit de "passer" les résultats de la synthèse au placeur routeur.

Les approches proposées précédemment dans cette étude sont applicables à tout FPGA à base de LUT. Les étapes considérées ici sont intimement liées à la cible technologique. Nous devons donc considérer un réseau spécifique sur lequel nous développons les étapes finales d'optimisation de réseaux Booléens: regroupement, insertion des buffer globaux, optimisation des blocs d'entrées/sorties, etc.

5.2 Regroupement

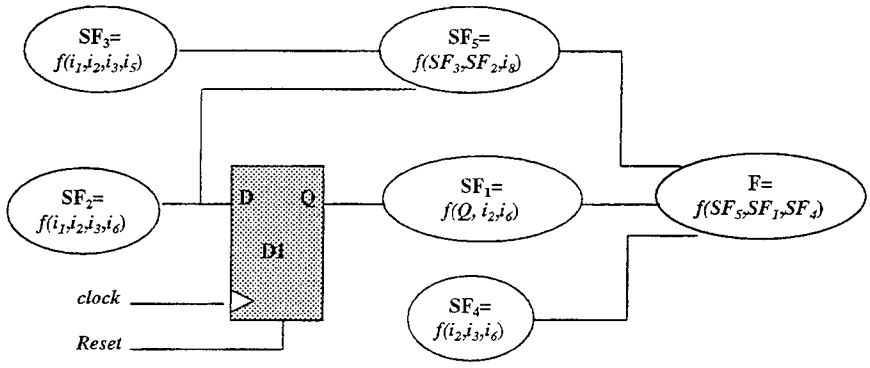
Dans les chapitres précédents, nous avons considéré que la cellule de base est constituée seulement par un LUT alors que pour la plupart des FPGA, la cellule renferme un ou plusieurs LUT, des points mémoires et d'autres ressources spécifiques. Le regroupement a pour but de décrire la configuration complète de la cellule de base.

Définition 5.1: graphe de décomposition

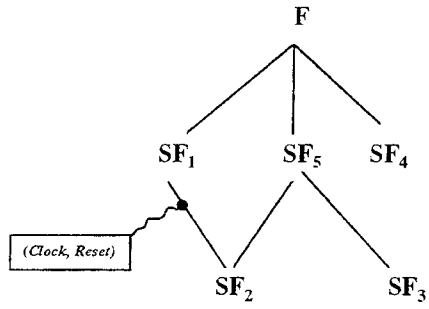
On appel graphe de décomposition, un graphe où les noeuds correspondent aux sous-fonctions définies au cours de la phase de décomposition technologique et où un arc relie SF_i et SF_j si SF_i est une variable de SF_j. si SF_i est mémorisée, l'arc (SF_i, SF_j) est étiqueté par le couple (<*nom du signal d'horloge*>, <nom du signal de *reset*>) pour la série 3000 et par le triplet (<*nom du signal d'horloge*>, <nom du signal de *reset*>, <nom du signal de *set*>) pour la série 4000.

Exemple :

Pour le réseau représenté par la figure 5.1-a, le graphe de décomposition correspondant est représenté par la figure 5.1-b. On remarque que l'arc reliant SF_2 à SF_1 est étiqueté par le couple (*Clock, Reset*) qui représente le point mémoire D1 de la figure 5.2-a.



(a) Exemple d'un réseau séquentiel



(b) Graphe de décomposition

Figure 5.1: Graphe de décomposition d'un réseau séquentiel

5.2.1 Regroupement pour la série 3000

Dans le cas du CLB de la série 3000 (figure 1.3), on remarque qu'il existe deux points mémoire dont les entrées peuvent être une entrée directe supplémentaire "di" ou une des sorties du bloc combinatoire. Les deux bascules ont la même horloge et le même "preset". L'objectif du regroupement est d'affecter chaque noeud du graphe de décomposition à un CLB ou à une partie du CLB. Ce problème est traité dans [Mur92] par la recherche de motifs prédéfinis dans le graphe de décomposition (*tree-matching*). Pour cela, on définit les motifs qui représentent les différentes configurations possibles du

CLB. Dans [Mur93], les auteurs ont revu cette méthode et ont défini un ensemble complet de 19 motifs pour la série 3000 (Figure 5.2).

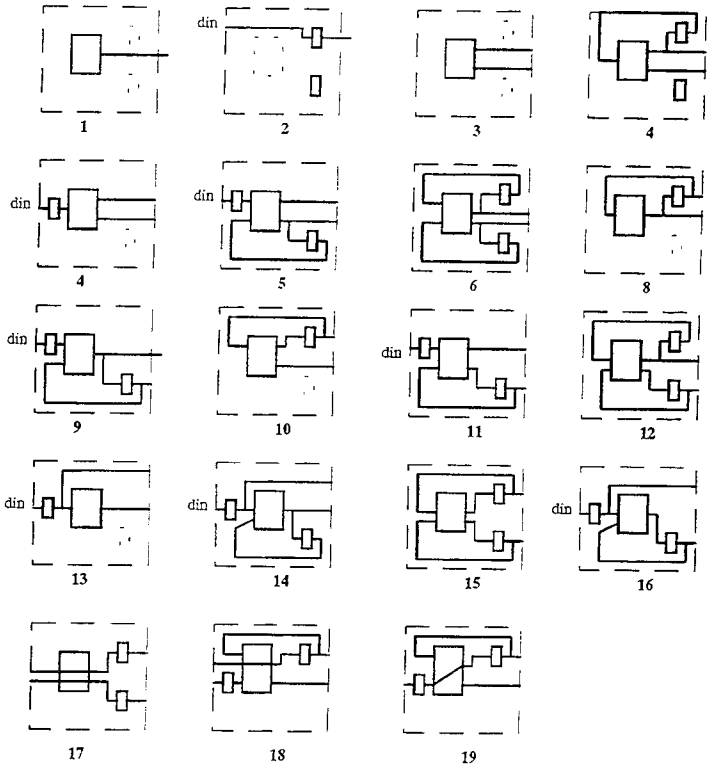


Figure 5.2: Motifs de regroupement pour la série 3000.

Dans cette section, nous allons donner une autre approche du problème et proposer une heuristique en O(n) (n nombre des noeud du graphe de décomposition). Pour ceci, nous allons définir un graphe appelé graphe de compatibilité.

Définition 5.2: Graphe de compatibilité

Dans le graphe de compatibilité un noeud représente une sous fonction ou une sortie primaire, un arc entre deux noeuds signifie que les deux sous fonctions associées aux extrémités, peuvent être placées dans le même CLB. Nous dirons alors que ces noeuds sont compatibles. Deux noeuds sont donc compatibles si:

-1- Les deux sous fonctions correspondantes dépendent au plus de quatre variables chacune et la cardinalité de l'union de leurs variables d'entrées est cinq.

-2- Si les deux sous fonctions correspondantes sont sur des points mémoire, ceux ci ont alors la même horloge et le même "preset" s'il existe.

Les sous fonctions dépendantes de cinq variables ne sont pas représentées dans ce graphe puisque toute fonction de cinq variables occupe la totalité du CLB et ne peut être compatible avec une autre sous fonction.

Exemple :

Le graphe de compatibilité du réseau de l'exemple précédent, est représenté par la figure 5.3.

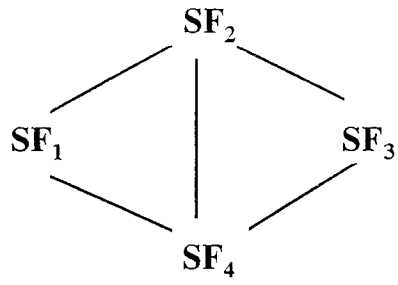


Figure 5.3: Graphe de compatibilité.

La minimisation du nombre de CLBs correspond alors à la recherche du couplage maximal dans le graphe de compatibilité. Cette étape de décomposition technologique peut être résolue avec une complexité en O(n2.5). Sachant que le plus grand circuit de la série 3000 actuel compte 418 CLB, il est nécessaire de chercher une heuristique pour le problème de regroupement.

Nous proposons une heuristique fondée sur l'idée de choisir les noeuds qui ont initialement un petit nombre de noeuds compatibles. l'algorithme de cette heuristique est en O(n):

Pas a) Initialisation de la liste des noeuds, triée par ordre croissant du nombres des noeuds compatibles.

Pas b) Placer dans CLB le premier noeud de la liste avec le premier noeud compatible dans cette liste. S'il n y a pas un noeud compatible alors placer ce noeud seul dans un CLB. aller au pas a.

Pour l'optimisation du routage, d'autres critères sont pris en considération dans le graphe de compatibilité. On remarque la possibilité d'utiliser les sorties des bascules comme entrées des fonctions combinatoires dans la cellule de la série 3000 sans utilisation des ressources de routage (Boucle interne à la cellule). Pour tenir compte de cette facilité, nous allons définir la notion de graphe de compatibilité pondéré.

Définition 5.3: Graphe de compatibilité pondéré

Un graphe de compatibilité pondéré est un graphe de compatibilité dont les arcs sont pondérés:

- Une arête aura un poids égal à 2 si elle relie deux fonctions compatibles et dont la première utilise la sortie du point mémoire sortie de la deuxième fonction [arête étiquetée par un couple (clk,reset) dans le graphe de décomposition].
- Une arête aura un poids égal à 1 si elle relie deux sous fonctions SF_i et SF_j compatibles et il existe une sous fonction SF_k dont l'expression utilise les deux fonctions SF_i et SF_j ($SF_k=f(SF_i,SF_j,..)$).
- Sinon l'arête aura un poids nul.

L'heuristique précédente est utilisée en modifiant le critère de trie qui doit tenir compte des poids des arêtes.

Exemple :

Pour le réseau de l'exemple précédent (figure 5.1), le graphe de compatibilité pondéré est représenté par la figure 5.3. La sortie du point mémoire D1 n'est utilisée que dans l'expression de SF1, c'est pourquoi l'arc (SF2, SF1) a un poids égal à 2. les sous fonctions SF2 et SF3 (Resp SF1 et SF4) sont utilisées dans l'expression de SF5 (resp F), l'arc correspondant a un poids égal à 1.

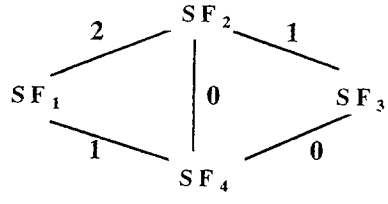


Figure 5.4: Graphe de compatibilité pondéré.

5.2.2 Regroupement pour la série 4000

L'approche proposée dans [Mur94] pour la série 3000 peut être utilisée pour résoudre le problème de regroupement pour la série 4000 en créant les motifs appropriés tenant compte de la structure du CLB de la série 4000. Vu la complexité de cette méthode, on propose une méthode qui ne traite que les parties séquentielles et les sous-fonctions qui présentent des configurations multiples (figure 5.5). Ceci est justifié par le fait que la décomposition technologique proposée pour la série 4000 oriente déjà le regroupement des sous-fonctions par la création des fonctions "H" (dépendant de trois variables) et des fonctions "F" (dépendant de quatre variables).

Exemple :

Soient $h_1=f(f_1,f_2,f_3)$ et $h_2=f(f_1,f_2)$ ou f_1, f_2 et f_3 sont des sous-fonctions de quatre variables.

Il y a trois configurations possibles pour implanter h_1 sur un CLB (figure 5.5-a) et trois configurations possibles pour implanter h_2 (figure 5.5-b).

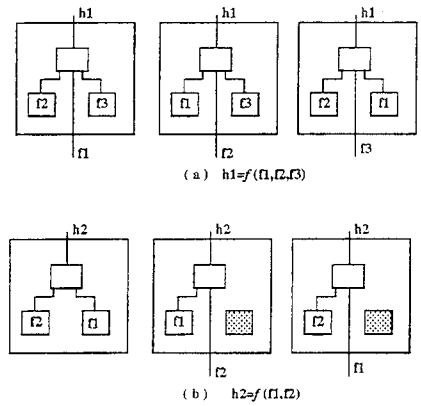


Figure 5.5: Sous-fonctions présentant des configurations multiples

Les sous-fonctions affectées par le regroupement sont:

- Les sous-fonctions dépendantes de trois variables et d'aucune entrée primaire.

- Les sous-fonctions dépendantes de quatre variables et qui sont variables de sous-fonctions de trois variables.

La représentation du graphe de compatibilité doit tenir compte de ces deux types de sous-fonctions. Pour simplifier le traitement, le graphe sera un graphe

bipartie. La première partition P_h est formée par les sous-fonctions "H" et la deuxième partition P_f par les sous-fonctions "F". Un arc entre un noeud f de P_f vers un noeud h de P_h indique que f est une variable de h.

Le but du regroupement est alors d'optimiser l'utilisation des blocs combinatoires du CLB et donc de maximiser le nombre de fonction "H" puisque l'affectation d'une fonction de moins de quatre variables dans un bloc "F" entraîne la sous utilisation du CLB. L'objectif de l'heuristique est d'affecter un noeud f de P_f au noeud h qui a le moins de configurations possibles. Pour cela, on affecte aux noeuds de h un poids qui est le nombre de noeuds de f requis pour être affecté à un bloc H du CLB. Les noeuds de P_f sont pondérés par le nombre de noeuds h avec lesquels il sont compatibles. L'algorithme de cette heuristique est le suivant:

Tant que P_h et P_f ne sont pas vides

- *prendre le noeud N_f de poids minimal*
- *Affecter N_f à son successeur N_h de poids maximal.*
- *Mettre à jour le poids des noeuds.*
- *Enlever N_f de P_f;*
- *Enlever de P_h les noeud de poids Nul.*

fin tant que

Exemple :

La figure 5.6 représente les étapes d'affectations de cet algorithme pour le système:

$h_1=f(f_1,f_3)$

$h_2=f(f_2,f_3,f_4)$

$h_3=f(f_3,f_4)$

ou f_1, f_2, f_3 et f_4 sont des sous-fonctions de quatre variables.

Le résultat de ce regroupement est illustré dans la figure 5.7 qui montre la disponibilité de deux LUT de type "F" qui peuvent être utilisés par la deuxième partie du regroupement que nous allons décrire par la suite.

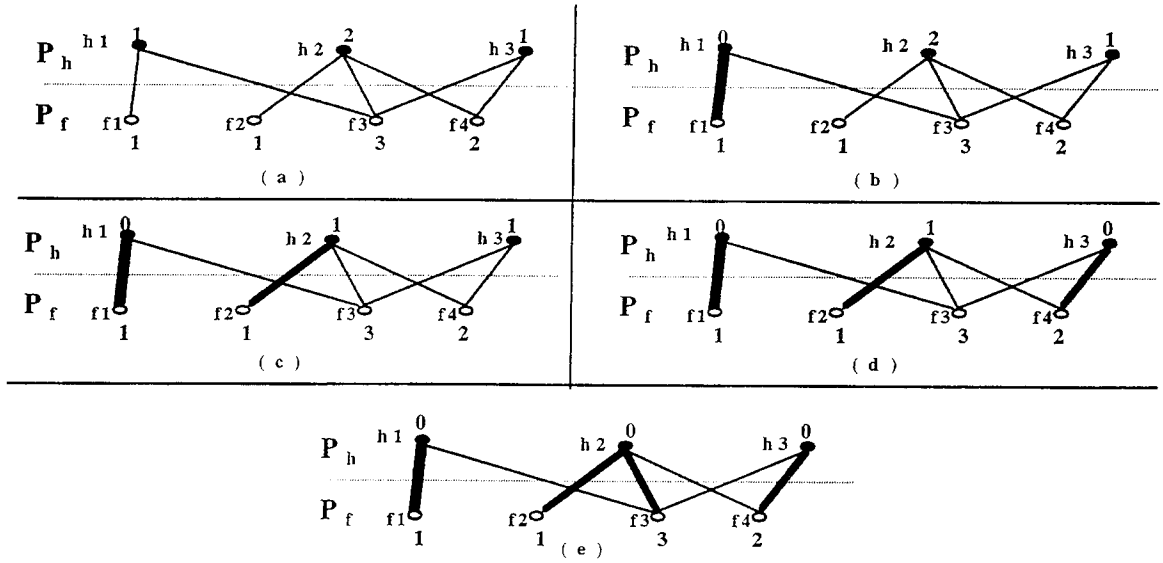


Figure 5.6: Heuristique de regroupement pour la série 4000

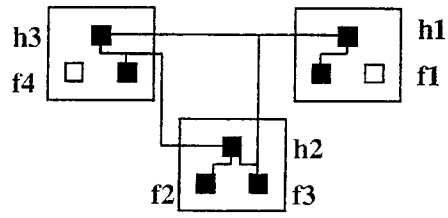


Figure 5.7: Implantation sur CLB

Les sous-fonctions qui ne sont pas affectées par l'algorithme précédent et les sous-fonctions dépendantes de quatre variables, peuvent être traitées par un deuxième algorithme de regroupement suivant deux critères; le premier optimisant la surface et le deuxième optimisant le routage. Pour le premier critère, il suffit d'affecter les sous-fonctions aux CLB partiellement utilisés par l'algorithme précédent. Pour tenir compte du routage, le critère pour grouper deux sous-fonctions dans un CLB sera le nombre d'entrées communes aux deux sous-fonctions.

5.3 Les Buffers globaux

Huit connections globales peuvent potentiellement atteindre toutes les cellules d'un circuit de la série 4000. Ces connections sont optimisées pour la distribution des signaux d'horloges et autres signaux critiques ou de sortance importante.

Quatre parmi les huit connections globales sont appelées connexions globales primaires. Elles offrent un délai minimal et une distorsion temporelle (*skew*) négligeable. Les autres sont appelées connexions globales secondaires. A cause de leur connectique qui est plus complexe, les connections globales secondaires introduisent un délais un peu plus important (1ns) et une distorsion temporelle additionnelle.

Les connections globales primaires et secondaires sont pilotées par des buffers appelés "BUFGP" et "BUFGS" respectivement. Ces buffers peuvent être reliés directement à des plots spécifiques du circuit ou à des signaux internes.

Les matrices de commutateurs au centre de chaque colonne connectent les huit buffers globaux aux quatre lignes verticales. Ces lignes sont utilisées pour distribuer les signaux aux colonnes. Chaque ligne verticale est commandée par un seul BUFGP et chaque BUFGP commande une ligne verticale différente. Un BUFGS peut commander une ou toutes les lignes verticales (figure 5.8).

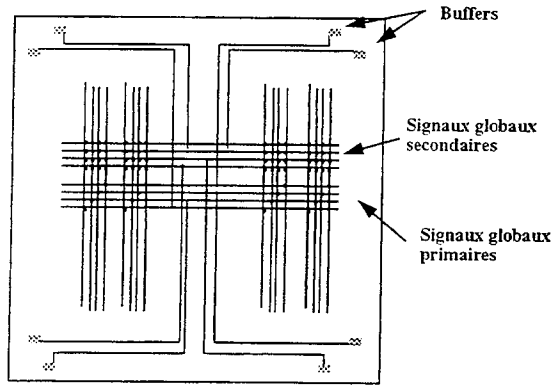


Figure 5.8: Buffers globaux (Xilinx 4000)

Par conséquent, les BUFGS sont plus flexibles pour le routage. Cette flexibilité est particulièrement intéressante pour des signaux autres que les signaux d'horloges. En effet, les lignes verticales ont des connections limitées aux ports d'entrées des CLB distincts de l'horloge. La possibilité de commander une ou toutes les lignes verticales à partir d'un seul buffer est très avantageuse.

Pour la série 3000, les mêmes remarques peuvent être faites pour les deux buffers globaux. L'un de ces derniers est appelé buffer global "GCLK" et l'autre est appelé buffer alternatif "ACLK".

Pour exploiter ces ressources, on détecte la liste des signaux d'horloge et des signaux de sortance importante. Cette liste est alors triée selon la criticité décroissante des signaux puis on insère les buffers globaux appropriés sur les premiers signaux de la liste triée.

5.4 les ressources des blocs d'entrées-sorties

Nous avons remarqué dans le chapitre I que chaque bloc d'entrée sortie des série 3000 et 4000, englobe deux points mémoire. Pour l'utilisation de ces point-mémoires, on a trois cas à considérer:

-a- Un point mémoire dont la sortie est une sortie primaire.

-b- Un point mémoire dont l'entrée est une entrée primaire.

-c- Un point mémoire interne au circuit.

Cas a: Point mémoire dont la sortie est une sortie primaire.

Si la sortie n'est pas utilisée à l'intérieur du circuit, il est évident que l'utilisation du point mémoire du plot associé à la sortie primaire est optimale.

Dans le cas où cette sortie est utilisée à l'intérieur du circuit, deux solutions sont possibles:

-1- Utiliser le buffer d'entrée du bloc pour le retour de la sortie à l'intérieur du circuit si le point mémoire ne peut pas être placé dans le CLB qui génère le signal d'entrée du point mémoire.

-2- Utiliser le point mémoire du CLB qui génère le signal d'entrée du point mémoire.

Le choix entre les deux solutions se fait suivant la liste des sous fonctions compatibles avec la sous fonction qui alimente le point mémoire. Si en ajoutant le point mémoire à la sous fonction associée, la liste des sous fonctions compatibles diminue, on place alors le point mémoire dans le bloc d'entrée/sortie.

Cette solution n'est pas recommandée dans le cas d'une optimisation du chemin critique et surtout si le chemin critique passe par le point mémoire. Dans ce cas, on préfère toujours placer le point mémoire dans un CLB.

Cas b: Point mémoire dont l'entrée est une entrée primaire.

Dans ce cas la sortie du point mémoire peut être routée en utilisant la boucle interne du CLB (*feed back*) (figure 5.9). Le point mémoire est alors placé dans le CLB, ce qui minimise le délai sur le signal de sortie ainsi que le routage surtout dans le cas où l'entrée primaire n'est utilisée que dans le point mémoire.

Dans le cas échéant, le point mémoire est placé dans le bloc d'entrée/sortie associé à l'entrée primaire.

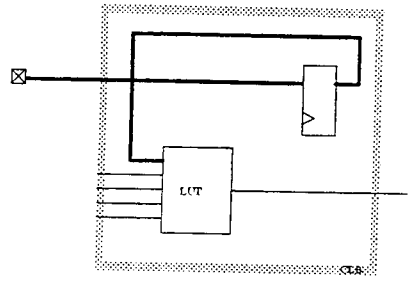


Figure 5.9: Rebouclage interne de la sortie du point mémoire

Cas c: Point mémoire interne au circuit.

Dans ce cas, on considère les point mémoires qui restent solitaires après regroupement, c'est-à-dire, les points mémoires qui n'ont pas été placés avec les fonctions qui les alimentent à cause des signaux de commande ou du fait de la présence d'autres points mémoires sur le même signal d'entrée. Dans ces cas, c'est le pourcentage des CLB utilisés par le circuit qui détermine s'il est plus intéressant de placer le point mémoire dans un bloc d'entrée-sortie ou dans un CLB. Le choix peut aussi être contrôlé par l'utilisateur suivant les besoins en performances du circuit car l'utilisation du point mémoire d'un bloc d'entrée-sortie est coûteux en délai.

5.5 Le placement relatif

Pour les CLB compris dans la zone critique du circuit, il est important d'utiliser au maximum les interconnexions directes pour minimiser les retards

sur les signaux critiques. Pour cela, on utilise les contraintes de placement relatives sur la zone critique. Ces contraintes sont passées, dans le réseau Booléen résultat, au placement-routage en affectant des positions contiguës aux CLB compris dans la zone critique.

5.6 Passage des contraintes de placement-routage

Pour le passage des contraintes de placement routage, on doit considérer que les contraintes utilisateurs existantes dans le réseau Booléen initial doivent être conservées en tenant compte des modifications de la netlist après optimisation sur les signaux et les portes concernées. A ces contraintes, on ajoutera des contraintes supplémentaires sur les signaux de la zone critique détectée après la décomposition technologique.

5.7 Evaluation du regroupement

Dans l'expérience suivante, on utilise la série 3000, ASYL+ génère deux fichiers au format XNF (format d'entrée pour les outils Xilinx). Dans le premier fichier, le regroupement est réalisé par ASYL+. Dans le deuxième fichier, la phase de la décomposition seulement est faite par ASYL+ et le programme XNFMAP version 5.1.0 (outil Xilinx) réalise le regroupement. Le but de cette expérience est de comparer le regroupement seulement. Le tableau 5.1 sur les exemples combinatoires montre un gain moyen dans ASYL+ de 11.2%. La colonne "*% de partage*" donne le pourcentage des fonctions compatibles (trouvées par ASYL+) par rapport au nombre total des fonctions réalisables après la phase de décomposition. Ce pourcentage confirme que notre approche avantage le partage de variables entre les sous fonctions.

	Nb. Fonctions réalisables	Nb. CLB XNFMAP	Nb. CLB ASYL+	Gain	% de partage
vg2	24	23	20	15,0	16,7
donfile	26	23	20	15,0	23,1
dk14	28	22	19	15,8	32,1
filglue	70	50	42	19,0	40,0
e64	99	62	52	19,2	47,5
tbk	106	77	73	5,5	31,1
s1	100	76	73	4,1	27,0
duke2	122	86	78	10,3	36,1
scf	263	197	181	8,8	31,2
zeegers	436	343	325	5,5	25,5
seq	533	394	376	4,8	29,5
			Moyenne	**11,2**	**30,9**

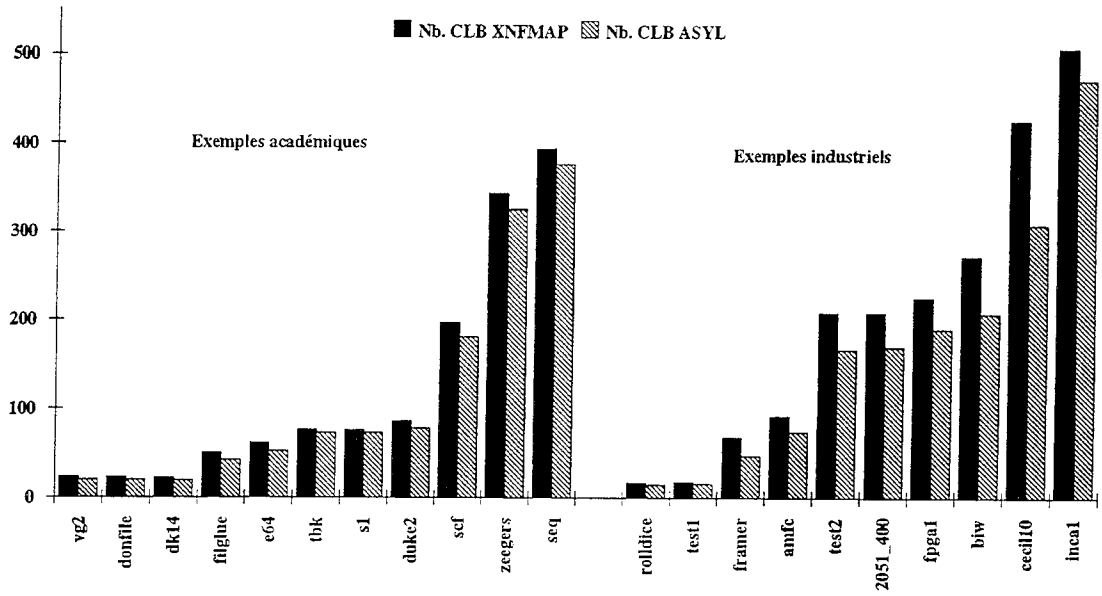
Tableau 5.1: Comparaison regroupement XNFMAP/ASYL+ (exemples combinatoires)

Le tableau 5.2 donne les résultats de la même expérience avec des exemples industriels. Avec notre approche, le gain est de 23.7%. Nous remarquons que notre méthode est efficace pour le traitement des parties séquentielles du CLB.

	Nb. fonctions réalisables	Nb. points mémoires	Nb. CLB XNFMAP	Nb. CLB ASYL+	Gain	% de partage
rolldice	28	15	17	15	13,3	86,7
test1	29	15	18	16	12,5	81,3
framer	76	49	68	47	44,7	61,7
amfc	121	50	92	74	24,3	63,5
test2	214	133	208	167	24,6	28,1
2051_400	220	133	208	170	22,4	29,4
fpga1	278	240	225	190	18,4	46,3
biw	324	131	272	207	31,4	56,5
cecil10	336	426	426	308	38,3	9,1
inca1	657	4	507	472	7,4	39,2
				Moyenne	**23,7**	**50,2**

Tableau 5.2: Comparaison regroupement XNFMAP/ASYL+ (exemples séquentiels)

L'histogramme 5.1 résume les résultats des tableaux 5.1 et 5.2. Ces résultats sont très importants et jouent un rôle primordial dans les différentes comparaisons que nous ferons avec d'autres outils commerciaux car pour ces outils, c'est généralement XNFMAP qui fait le regroupement.



Graphe 5.1: Evaluation du regroupement

Nous avons essayé de faire cette comparaison avec les seuls résultats de la littérature concernant le regroupement [Mur93] mais la comparaison n'est pas possible car dans la phase de décomposition, les résultats d'ASYL+ sont meilleurs. Nous ne pouvons pas donc évaluer l'algorithme de regroupement de façon isolée.

5.8 Conclusion

L'étape de regroupement et d'optimisation des ressources spécialisées (horloge, blocs d'entrées-sorties...) est fortement liée à la cible technologique. Il faut noter que certaines standardisations commencent à apparaître dans la structure des blocs d'entrées-sorties et des ressources séquentielles de la cellule de base dans certains FPGA (blocs d'entrées-sorties de la famille ACT3 d'Actel et celui de la série 4000 de Xilinx...). Ces similarités sont dictées par les besoins des concepteurs.

Les méthodes de regroupement proposées sont fondées sur la notion de graphe de compatibilité qui reflète les caractéristiques de la cible. Ce graphe est facilement adaptable pour d'autres architectures et permet donc l'utilisation de notre approche dans un cadre plus général (cette méthode a été appliqué dernièrement avec succès à la nouvelle série Xilinx 5000).

Chapitre VI

Résultats et évaluations

6.1 Introduction

Comme annoncé au début de cette étude, ces travaux ont comme objectif de proposer des techniques de factorisation, de décomposition technologique et d'optimisation de réseaux Booléens sur FPGA à base de LUT. Toutes ces techniques de synthèse logique sont implantées dans le système ASYL+. Remarquons que la construction des ROBDD et la recherche d'un meilleur ordre ont été améliorées dernièrement dans le système ASYL+, ce qui affecte les résultats pour la série 4000. Ces améliorations ne sont pas rapportées dans cette thèse, nous donnerons ici les résultats du système ASYL+ version 3.0.0 (Aout 1994), qui utilise les méthodes décrites dans cette étude.

Pour illustrer ces techniques de synthèse, nous comparons nos résultats avec les deux outils de synthèse logique les plus connus dans le commerce provenant des sociétés SYNOPSYS Inc. et EXEMPLAR Logic Inc. Cette comparaison est faite soit sur un ensemble des d'exemples réels provenant de différentes sociétés et conçus sur FPGA, soit des benchmarks académiques internationaux [MCN89].

Ces comparaisons sont faites sur Sparc-10 pour ASYL+/Synopsys et sur PC 486DX2 (16Mo) pour ASYL+/Exemplar. Pour les trois systèmes et les différents critères d'optimisation, nous avons utilisé les options qui donnent la meilleure solution. Dans les tableaux des résultats nous ne rapportons pas les temps CPU et l'occupation mémoire, puisque les expériences sont faites sur deux plateformes. Sur un ensemble d'exemples réduit, nous remarquons que le gain moyen en temps CPU d'ASYL+ est de 2% par rapport à Synopsys et de 40% pour l'occupation mémoire. Nous n'avons pas l'occupation mémoire d'Exemplar. L'arrêt d'Exemplar sur certains exemples pour mémoire inssuffisante nous montre quil utilise plus de mémoire qu'ASYL+. Le même problème d'evaluation se pose pour donner une moyenne du gain en temps CPU d'ASYL+ par rapport à Exemplar pour lequel plusieurs exemples ont été arrétés après 24h d'éxecution. Nous pouvons donc affirmer qu'ASYL+ a un gain CPU très important par rapport à Exemplar. Ces constatations montrent

que notre approche maitrise le problème de la complexité et que les différents étapes de la synthèse sont cohérentes et efficaces.

6.2 Comparaisons orientées surface

6.2.1 Comparaisons ASYL+/Synopsys/Exemplar pour la série 3000

Les résultats de comparaison, pour l'optimisation orientée surface, des trois systèmes ASYL+, Synopsys et Exemplar pour les FPGA Xilinx série 3000 sont représentés dans les tableaux 6.1 et 6.2 . Ces résultats sont donnés en nombre de CLB respectivement pour les exemples MCNC et les exemples industriels. Les gains moyens avec notre méthode par rapport à Synopsys sont respectivement 8.5% et 29,3% pour les deux séries d'exemples. Les gain respectifs par rapport à Exemplar sont 33,8% et 77%. Le gain moyen sur tous les exemples par rapport à Synopsys est de 15.7% et de 40.1% par rapport à Exemplar. Sur l'histogramme 6.1, qui résume ces résultats nous remarquons que le gain pour les exemples industriels (séquentiels) est plus important que pour les exemples MCNC (combinatoires). Ceci s'explique par la cohérence des phases de notre approche (factorisation, décomposition et regroupement) et l'efficacite de notre traitement pour les exemples complexes et les parties sequentielles des circuits. Ceci est accentué par le gain de la phase de regroupement par rapport à XNFMAP (voir évaluation dans le tableau 5.1). Rappelons que les deux systèmes Synopsys et Exemplar ne font pas le regroupement complet mais guident seulement XNFMAP pour la configuration des CLB.

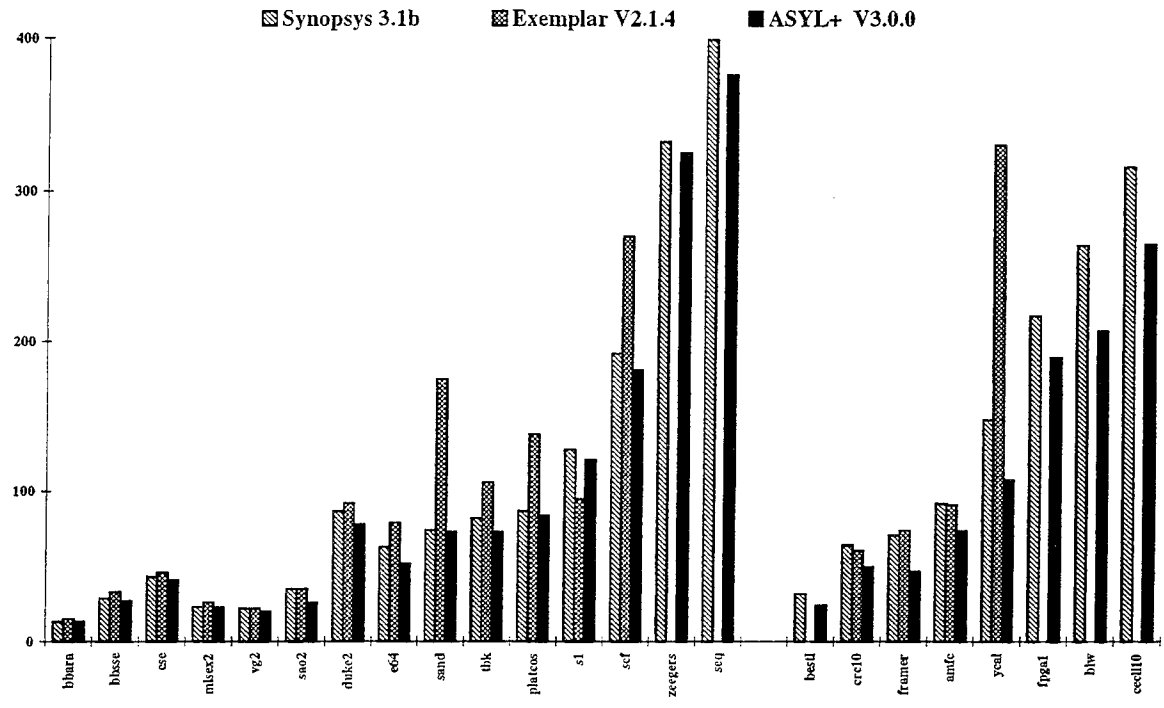
	Synopsy V3.1b	Exemplar V2.1.4	ASYL+ v3,0,0	**Gain ASYL+/ Synopsys**	**Gain ASYL+/ Exemplar**
bbara	13	15	13	**0**	**15,4**
bbsse	29	33	27	**7,4**	**22,2**
cse	43	46	41	**4,9**	**12,2**
misex2	23	26	23	**0**	**13,0**
vg2	22	22	20	**10,0**	**10,0**
sao2	35	35	26	**34,6**	**34,6**
duke2	87	92	78	**11,5**	**17,9**
e64	63	79	52	**21,2**	**51,9**
sand	74	175	73	**1,4**	**139,7**
tbk	82	106	73	**12,3**	**45,2**
platcos	87	138	84	**3,6**	**64,3**
s1	128	95	121	**5,8**	**-21,5**
scf	192	270	181	**6,1**	**49,2**
zeegers	332	A	325	**2,2**	
seq	399	M	376	**6,1**	
			Moyenne	**8,5**	**33,8**

Tableau 6.1: Résultat série 3000 optimisation surface (exemples combinatoires MCNC)

A: Arrété après 24h d'éxecution.
M: Mémoire insuffisante.
*: Rejeté au placement routage

	Synopsys 3.1b	Exemplar V2.1.4	ASYL+ V3.0.0	**Gain ASYL+/ Synopsy**	**Gain ASYL+/ Exemplar**
bestl	32		24	**33,3**	**0,0**
crc10	64	61	50	**28,0**	**22,0**
framer	71	74	47	**51,1**	**57,4**
amfc	92	91	74	**24,3**	**23,0**
ycal	148	330	108	**37,0**	**205,6**
fpga1	217	A	190	**14,2**	
biw	264	A	207	**27,5**	
cecil10	316	M	265	**19,2**	
			Moyenne	**29,3**	**77,0**

Tableau 6.2: Résultat série 3000 optimisation surface (exemples industriels)



Graphe 6.1 Comparaison orientée surface

6.2.2 Comparaison ASYL+/Synopsys/Exemplar pour Xilinx série 4000

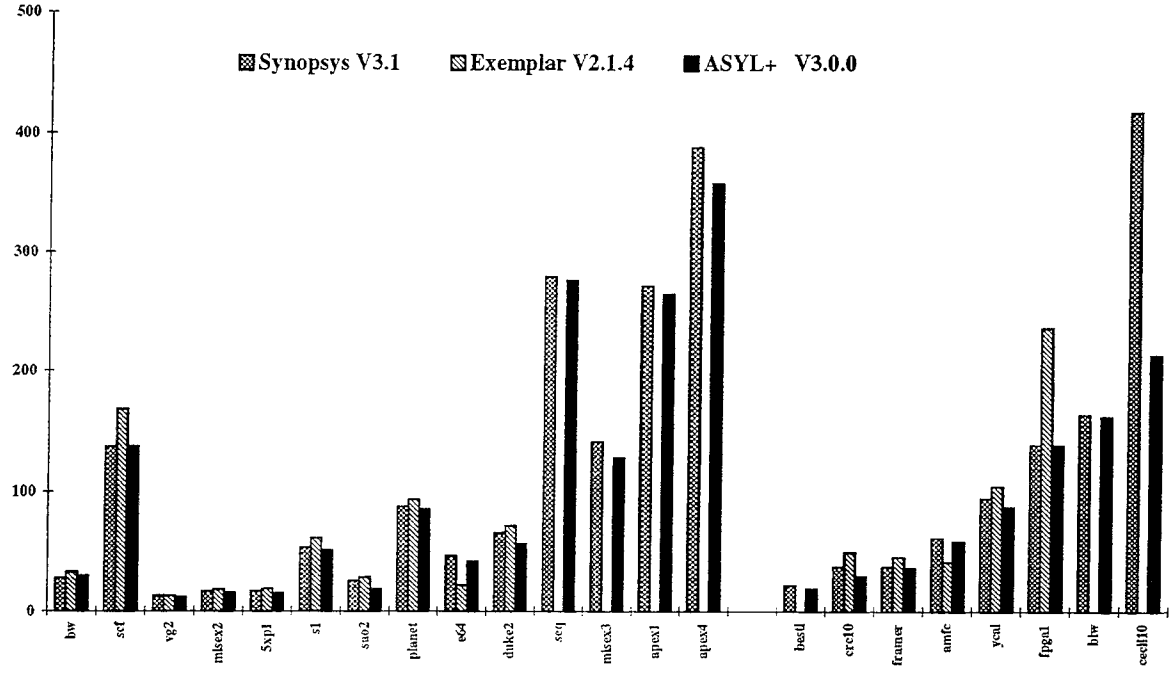
Les tableaux 6.4 et 6.5 répresentent les résultats de comparaison des trois systèmes ASYL+, Synopsys et Exemplar pour les FPGA Xilinx série 4000. Ces tableaux sont donnés respectivement en nombre de CLB pour les deux séries d'exemples MCNC et les exemples industriels. Les gains moyens de notre méthode par rapport à Synopsys sont respectivement de 8.2% et de 19,5%. Les gains respectifs par rapport à Exemplar sont de 14% et de 30.3%. Le gain moyen sur tous les exemples par rapport à Synopsys est de 12.3% et de 19.4% par rapport à Exemplar. L'histogramme 6.2 résume ces résultats.

	Synopsys V3.1	Exemplar V2.1.4	ASYL+ V3.0.0	**Gain ASYL/ Synopsys**	**Gain ASYL/ Exemplar**
bw	28	33	30	**-6,7**	**10,0**
scf	137	168	137	**0,0**	**22,6**
vg2	13	13	12	**8,3**	**8,3**
misex2	17	18	16	**6,3**	**12,5**
5xp1	17	19	15	**13,3**	**26,7**
s1	54	62	52	**3,8**	**19,2**
sao2	26	29	19	**36,8**	**52,6**
planet	88	94	86	**2,3**	**9,3**
e64	47	22	42	**11,9**	**-47,6**
duke2	66	72	57	**15,8**	**26,3**
seq	279	M	276	**1,1**	
misex3	141	M	128	**10,2**	
apex1	271	M	264	**2,7**	
apex4	388	M	358	**8,4**	
			Moyenne	**8,2**	**14,0**

Tableau 6.3: Résultat série 4000 optimisation surface (exemples combinatoires MCNC)

	Synopsys 3.1b	Exemplar V2.1	ASYL+ V3.0.0	**Gain ASYL+/ Synopsys**	**Gain ASYL+/ Exemplar**
bestl	22		19	**15,8**	
crc10	38	50	30	**26,7**	**66,7**
framer	38	46	37	**2,7**	**24,3**
amfc	62	42	59	**5,1**	**-28,8**
ycal	95	105	88	**8,0**	**19,3**
fpga1	139	236	139	**0,0**	**69,8**
biw	164	*	162	**1,2**	
cecil10	418	M	213	**96,2**	
			Moyenne	**19,5**	**30,3**

Tableau 6.4: Résultat série 4000 optimisation surface (exemples industriels).



Graphe 6.2: Comparaisons série 4000 Surface

6.3 Comparaisons orientées chemin critique

6.3.1 Comparaisons ASYL+/Synopsys/Exemplar pour la série 3000

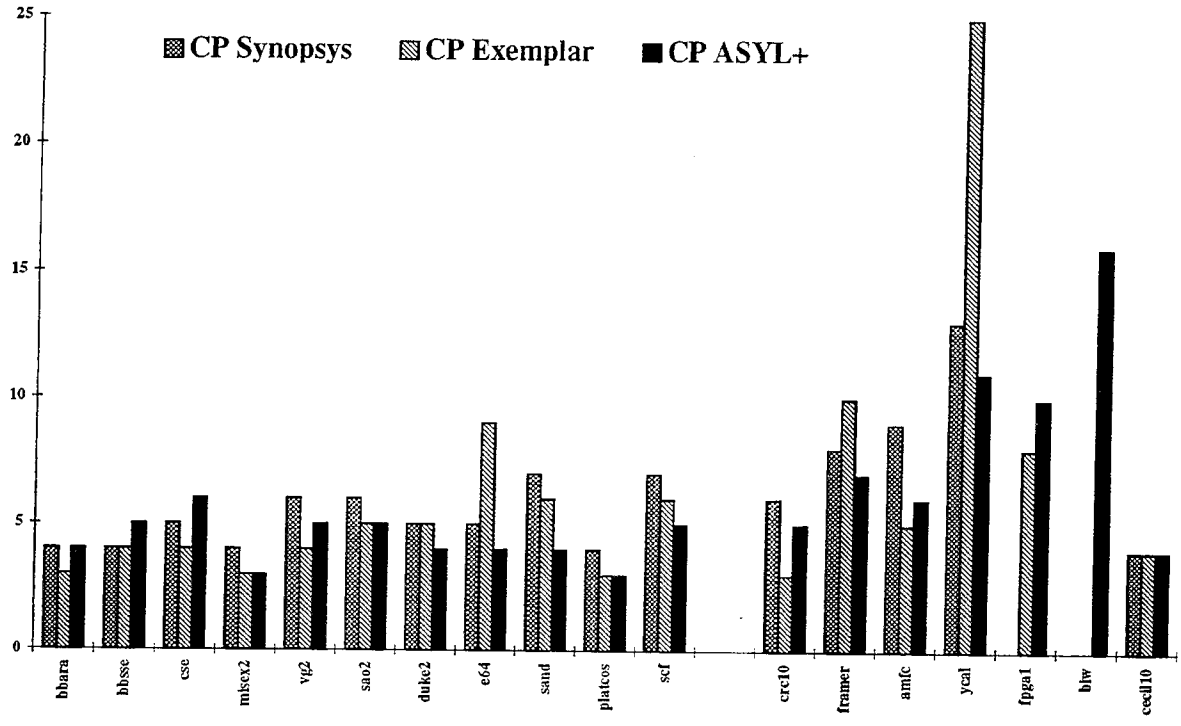
Les résultats comparant l'optimisation du chemin critique des trois systèmes ASYL+, Synopsys et Exemplar pour les FPGA Xilinx série 3000 sont donnés dans les tableaux 6.5 et 6.6. Les colonnes marquées "CP" donnent la profondeur du réseau résultant en terme du nombre de CLB traversées dans le plus long chemin. Les gains moyens de notre méthode par rapport à Synopsys sont respectivement de 21.4% et de 20.5% pour les deux séries d'exemples. Les gains respectifs par rapport à Exemplar sont de 11.1% et de 15.6%. Le gain moyen sur tous les exemples par rapport à Synopsys est de 21.1% et de 12.7% par rapport à Exemplar. L'histogramme 6.3 résume ces résultats.

	Synopsys V3.1		Exemplar V2.1.4		ASYL V3.0.0		Gain ASYL/ Synopsys	Gain ASYL/ Exemplar
	CLB	CP	CLB	CP	CLB	CP		
bbara	13	4	20	3	13	4		**-25,0**
bbsse	30	4	40	4	27	5	**-20,0**	**-20,0**
cse	45	5	62	4	41	6	**-16,7**	**-33,3**
misex2	23	4	36	3	23	3	**33,3**	
vg2	24	6	26	4	22	5	**20,0**	**-20,0**
sao2	35	6	35	5	45	5	**20,0**	
duke2	98	5	195	5	133	4	**25,0**	**25,0**
e64	255	5	41	9	87	4	**25,0**	**125,0**
sand	129	7	187	6	176	4	**75,0**	**50,0**
platcos	83	4	138	3	86	3	**33,3**	
scf	199	7	304	6	286	5	**40,0**	**20,0**
Moyenne							**21,4**	**11,1**

Tableau 6.5: Résultat série 3000 optimisation chemin critique (exemples MCNC).

	Synopsys V3.1		Exemplar V2.1.4		ASYL+ V3.0.0		Gain ASYL+/ Synopsys	Gain ASYL+/ Exemplar
	CLB	CP	CLB	CP	CLB	CP		
crc10	63	6	66	3	49	5	**20,0**	**-40,0**
framer	73	8	61	10	58	7	**14,3**	**42,9**
amfc	97	9	144	5	76	6	**50,0**	**-16,7**
ycal	174	13	460	25	144	11	**18,2**	**127,3**
fpga1	*	*	316	8	328	10		**-20,0**
biw	*	*	*	*	227	16		
cecil10	313	4	421	4	274	4	**0,0**	**0,0**
Moyenne							**20,5**	**15,6**

Tableau 6.6: Résultat série 3000 optimisation chemin critique (exemples industriels).



Graphe 6.3 Comparaison orientées chemin critique

Conclusions

Ce travail a comme objectif la synthèse de réseaux Booléens sur réseaux programmables de type Xilinx, séries 3000 et 4000. Rappelons que les étapes classiques de la synthèse sont l'optimisation logique (minimisation et factorisation) et la décomposition technologique. L'optimisation logique est souvent considérée comme phase de "préparation" indépendante de la technologie cible. L'enjeu théorique est de montrer que la phase de "préparation" est critique. Il s'agit de trouver les expressions Booléennes les mieux adaptées à la décomposition ultérieure et de trouver le meilleur mode de représentation.

Un effort d'intégration est fait dans cette thèse, en étudiant trois types de factorisation (factorisation lexicographique, factorisation algébrique restreinte et factorisation fondée sur les ROBDD) et en comparant les résultats après placement routage. Cette comparaison a confirmé l'efficacité de la factorisation lexicographique pour préparer le routage, celle de la représentation en ROBDD pour l'optimisation des exemples symétriques et celle des filtres dans le choix des candidats diviseurs, selon les critères d'optimisation, de la factorisation algébrique restreinte. Cette étude met en évidence d'une part, la richesse en termes de représentations de fonction Booléennes et de traitements que doit offrir un environnement de synthèse logique. Ceci est nécessaire pour atteindre une bonne optimisation sur des cibles technologiques variées et pour couvrir l'espace des solutions. D'autre part, l'optimisation pour chacune de ces représentations doit être très poussée: une minimisation efficace pour la représentation en somme de monômes, une technique efficace de division pour les expressions factorisées et un bon ordre de variables pour les expressions lexicographiques et les graphes de décision binaires.

L'approche proposée pour la décomposition technologique s'est avérée très rapide et conduit à d'excellents résultats. Remarquons que sa principale caractéristique est également la grande cohérence avec l'étape de factorisation. En effet, en exploitant l'ordre des variables induit par la factorisation lexicographique et la factorisation fondée sur les ROBDD, l'insertion des points de coupure respecte les cônes logiques créés par ces factorisations. De plus, l'étape de décomposition technologique est optimisée non seulement en

fonction de chaque représentation mais aussi pour chaque cible et chaque critère d'optimisation (surface ou performance).

Il est à noter que peu de standardisations ont vu le jour ces dernières années dans le domaine des FPGA. Certaines grandes tendances architecturales sont nettes, architectures à base de LUT (Xilinx, ATT, Flex d'Altera), architectures à base de Multiplexeurs (Actel, Quicklogic) par exemple. Malgré cela, les optimisations restent spécifiques aux cibles technologiques et demandent des efforts d'innovation importants. Typiquement, un problème ouvert pour les réseaux reprogrammables est celui de la prédiction des délais de routage pour la détection exacte des zones critiques. Une solution à ce problème assurera une amélioration certaine de l'optimisation du chemin critique et permettra l'optimisation sous contraintes.

Les expériences sur un grand nombre de circuits types industriels et académiques ont montré qu'une étape finale qui consiste au regroupement et à l'optimisation des ressources est toujours requise. Cette étape tient compte des ressources de la cellule de base, des ressources de routage et des algorithmes de placement/routage. Les méthodes proposées pour la réalisation de cette étape sont rapides et ont contribué grandement à la qualité des résultats.

La conjugaison de la complexité des problèmes de synthèse, classés comme étant NP-complets, et de la taille sans cesse croissante des circuits à traiter, a fait que le développement d'heuristiques performantes, rapides et de complexité raisonnable, était un des problèmes critiques dans la réalisation pratique. Les comparaisons en terme de temps CPU pour la synthèse complète par rapport aux deux systèmes Synopsys et Exemplar montrent la cohérence et l'efficacité des approches proposées.

Ce travail a fait l'objet d'un transfert technologique vers un système industriel de CAO, appelé ASYL+. Ceci a nécessité un suivi constant des formats de sorties et des différentes versions des outils de placement/routage. Ce transfert technologique est une preuve de la qualité des résultats en comparaison aux deux systèmes les plus compétitifs sur le marché de la synthèse logique sur FPGA (Synopsys et Exemplar).

BIBLOGRAPHIE

[Abo90] P.Abouzeid, L. Bouchet, K. Sakouti, G. Saucier, P. Sicard, "*Lexicographical expression of Boolean function for multilevel synthesis of high-speed circuits*", SASIMI'90, pp. 31-39, Kyoto, Japan, Octobre 1990.

[Abo92] P.Abouzeid, "Méthode de Factorisation Algébrique Dédiées aux Circuits Intégrés Complexes", Thèse de Doctorat, Institut National Polytechnique de Grenoble, Décembre 1992.

[Abo93] P.Abouzeid, B. Babba, M. Crates de Paulet, G. Saucier, "*Input-Driven Partitioning Methods and Application to Synthesis On Table-Lookup-Based FPGAs*", IEEE Transaction on CAD of integrated circuit and system, Vol 12, N° 7, pp 913-925, Juillet 1993.

[Act94] ACT Family Field Programmable Gate Array Databook, April 1994.

[Bab92] B. Babba, M. Crastes, "*Automatic Synthesis on Table Lookup-based PGAs*", Proc. Euro Asic 92, pp 25-31, Juin 1992.

[Bab92a] B. Babba , M. Crastes, "*Area and timing oriented technology mapping for table-lookup based FPGAs*", IFIP International Workshop on application-oriented Synthesis, Dresden, Germany. , Mars 1992

[Bab93] B. Babba, M.C. Bertrand, F. Preaud, M. Schnebelen, "*Synthesis from a VHDL description using Xilinx Blox library*", Proc. of The User Forum and EUROASIC Prizes Sessions, pp 117-120 , Fevrier 1993.

[Bab94] B. Babba, M. Crastes, "*Technology mapping on FPGAs*", Proc. JJC en Architecture de machine et systèmes 94, pp 25-31, Décembre 1994.

[Bes93] T. Besson, R.K. Brayton, G. Saucier, "*Sharing analysis for BDD Variable Ordering*", Proc. SASIMI 93, pp. 353-365, Octobre 1993.

[Bol94] B. Bollig, P. Savicky, I. Wegener, "*On The Improvement of Variable Orderings For OBDDs*", IFIP Workshop on Logic and Architecture Synthesis, pp 71-80, Decembre 1994.

[Bou90] L. Bouchet, G. Saucier, "*Multi-Level Synthesis on PALs and on PGAs*", IFIP Working Conference on Logic and Architecture Synthesis, Paris, France, Mai 1990.

[Bou93] H. Bouzouzou, T. Besson, R. Roane, G. Saucier, "*Input order for ROBDDs based on kernel analysis*", Proc. EDAC 93, Fevrier 1993.

[Bra86] R.K. Brayton and all, "*Multiple-level logic optimization system*", Proc. ICCAD 86, pp. 356-359, Novembre 1986.

[Bra87a] R. Brayton, "*Factoring Logic Functions*", IBM J. Research, vol 31, pp 187-198, Mars 1987.

[Bra87b] R.K. Brayton and all, "*MIS, a multilevel logic optimization system*", IEEE Trans. on Comp. vol CAD 6, n°6, pp. 1062-1081, Novembre 87.

[Bra90] K. Brace, R. Rudell, R. Bryant, "*Efficient Implementation of a BDD Package*", Proc. 27th DAC, Juin 1990.

[Bra92] R.K. Brayton, C.T. McMullen, "*The decomposition and factorization of boolean expressions*", Proc. ISCAS, pp. 49-54, Mai 1992.

[Bro93] S.D. Brown, R.J. Francis, J. Rose, Z.G. Vranesic, *"Field-Programmable Gate Arrays"*, Kluwer Academic Publishers, , 1993.

[Bry86] R.E. Bryant, *"Graph-based algorithms for Boolean function manipulation"*, IEEE Transaction on Computers, Vol. C-35, N°8, pp 677-692, Aout 1986.

[Cal91] N. Calazans, R. Jacobi, Q. Zhang, C. Trullemans, "*Improving Binary Decision Diagrams Through Incremental Reduction and Improved Heuristics*", Custom Integrated Circuits Conference, 1991.

[Car91] G. Caruso, "*Near Optimal Factorization of Boolean Functions*", IEEE Trans. on CAD, Vol 10, No 8, pp 1072-1078, Aout 1991.

[Con94] J. Cong, Y. Ding, *"FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimisation in Lookup-Table Based FPGA Designs"*, IEEE Transactions on CAD, Vol 13, No1, pp. 11-12, Juin 1994.

[Cra89] M. Crastes de Paulet, C. Duff, G. Saucier *"ASYL: a Logic and Architecture Design Automation System"*, Proc. Euro Asic 89, pp 183-209, Janvier 1989.

[Cra91] M. Crastes, K. Sakouti, G. Saucier, "*A technology mapping method based on perfect and semi perfect matchings* ", Proc. 28th ACM/IEEE DAC, San Francisco, USA, Juin 1991.

[Cra92] M. Crastes, B. Babba, G. Saucier, "*Input driven Synthesis on PLDs and PGAs*", Proc. EDAC, pp 48-52 Mars 1992.

[Duf91] C. Duff, "Codage d'Automates et théorie des Cubes Intersectants", Thèse de Doctorat, Institut National Polytechnique de Grenoble, Mars 1991.

[Fil91] D. Filo, J.C-Y Yang, F. Mailhot, G. De Micheli, *"Technology mapping for a Two-Output RAM-Based Field Programmable Gate Array* ", Proc. EDAC, pp. 534-538, Fevrier 1991.

[Fra90] R.J. Francis, J. Rose, K. hung, "*Chortle : a Technology Mapping Program for Lookup Table-based Field Programmable Gate Arrays*", Proc. 27th ACM/IEEE DAC, pp. 613-619, Juin 1990.

[Fra91a] R.J. Francis, J. Rose, Z. Vranesic, "*Technology Mapping for Lookup Table-Based FPGAs for Performance*", Proc. ICCAD, pp. 568-571, November 1991.

[Fra91b] R.J. Francis, J. Rose, Z. Vranesic, "*Chortle-crf: Fast Technology Mapping for Lookup Tables-Based FPGAs* ", Proc. 28th ACM/IEEE DAC, San Francisco, USA, pp. 227-233, Juin 1991.

[Fri87] S. Friedman, K. Supowit, "*Finding the Optimal Variable Ordering for Binary Decision Diagrams*", Proc. 24th ACM/IEEE DAC, Juin 1987.

[Fuj91] M. Fujita, Y. Matsunaga, T. Kakuda, "*Heuristics to compute variable ordering for efficient manipulation of Ordered Binary Decision Diagrams*", Proc. 28th ACM/IEEE DAC. Amsterdam, pp. 417-420, Fevrier 1991.

[He93] S. He, M. Torkelson, "*Logic synthesis for look Up Table Architecture through Multi-output Boolean Decomposition*", Proc. SASIMI'93, pp. 433-439, Octobre 1993.

[Hsu92] W.J. Hsu and W.Z. Shen, "*Coalgebraic Division for Multilevel Logic Synthesis*", Proc. 29th DAC, pp 438-442, Juin 1992.

[Ish91] Ishiura, H. Sawada, S. Yajima, "*Minimisation of Binary Decision Diagrams Based on Exchanges of Variables*", Proc. ICCAD, pp. 472-475, Novembre 1991.

[Ist95] IST Inc., "*ASYL+ User Manuel*", Mars 1995.

[Kar91] K. Karplus, "*Xmap: a Technology Mapper for Table-lookup Field-Programmable Gate Arrays*", Proc. 28th DAC, pp. 240-243, Juin 1991.

[Keu87] K. Keutzer, "*Dagon: Technology Binding and Local Optimization*", Proc. DAC , pp341-347, Juin 1987.

[Kun68] J. Kuntzmann. "*Algébre de Boole*", Edition Dunod, Paris, 1968.

[MCN89] Microelectronics Center of North Carolina, Standard Synthesis Benchmarks, International Workshop on Logic Synthesis, Mai 1989.

[Min90] S. Minato, N. Ishiura, S. Yajima, *"Shared Binary Diagram with attributed edges for efficient boolean function manipulation"*. Porc. 27th ACM/IEEE DAC, pp52-57, 1990.

[Mun87] R. R. Munoz, C.E. Stroud, "*Automatic partitioning of Programmable Logic Devices*", VLSI systems Design, pp. 74-86, Octobre 1987.

[Mur91a] R. Murgai, N. Shenoy, R.K. Brayton, A. Sangiovanni-Vincentelli, "*Improved Logic Synthesis Algorithms for Table Look Up Architectures*", Proc. ICCAD, pp. 564-567, November 1991.

[Mur91b] R. Murgai, N. Shenoy, R.K. Brayton, A. Sangiovanni-Vincentelli, "*Performance Directed Synthesis for Table Look Up Programmable Gate Arrays*", Proc. ICCAD, pp. 572-575, Novembre 1991.

[Mur90] R. Murgai, Y. Nishizaki, N. Shenoy, R.K. Brayton, A. Sangiovanni-Vincentelli, "*Logic Synthesis for Programmable Gate Arrays*", Proc. 27th ACM/IEEE DAC, pp. 620-625, Juin 1990.

[Mur93] R. Murgai, R.K. Brayton, A. S.-Vincentelli, "Sequential Synthesis for Table Look Up Programmable Gate Arrays", Proc. 30th ACM/IEEE DAC, pp. 224-229. 1993.

[Rud93] R. Rudell, *"Dynamic variable ordering for ordered binary decision diagrams"*, Proc. IEEE ICCAD, pp. 42-47, 1993.

[Sak90] K. Sakouti, "Synthèse et Optimisation Temporelle de Réseaux Booléens", Thèse de Doctorat, Institut National Polytechnique de Grenoble, Novembre 1993.

[Sau90] G. Saucier, P. Abouzeid et al, "*Multi-Level Synthesis minimizing the routing factor*", Proc. 27th DAC, pp. 365-368, Juin 1990.

[Sau91] G. Saucier, F. Poirot, "*A Good Input Ordering for Circuit Verification Based on Binary Decision Diagrams* ", Proc. Euro Asic 91, Paris, France 1991.

[Sau92] G. Saucier, "*Analysis of the Trends in Logic Synthesis* ", Sasimi 92, Kobe, Japan, Avril 1992.

[Sau93a] G. Saucier, J. Fron, P. Abouzeid, "*Lexicographical expression of boolean function and application to multilevel synthesis*", IEEE transaction on CAD, Vol. 12, N°11, pp 1642-1654, Novembre 1993.

[Sau93b] G. Saucier, H. Bouzouzou, B. Babba, R. Roane, F. Poirot, "*Use of Binary Decision Diagrams for Logic Synthesis*", Poster Session,

International Workshop on Logic Synthesis (IWLS'93), Iahoe City, CA. USA, Mai 1993.

[Sic88] P. Sicard, "Nouvelles Méthodes de Synthèse Logique", Thèse de Doctorat, Institut National Polytechnique de Grenoble, Meptembre 1988.

[Tou91] H.J.Touati, H.Sajov, R.Brayton, "*Delay Optimization of Combinational Logic Circuit by Clustering and Partial Collapsing*", Proc.ICCAD 91, pp. 188-191, Santa Clara, USA, Novembre 1991.

[Vas90] J. Vasudevamurthy and J. Rajski, "*A Method for Concurrent Decomposition and Factorization of Booleans Expressions*", Proc. ICCAD 90, pp 510-513, Novembre 1990.

[Xil94] Xilinx Programmable Gate Array User's Guide, Xilinx Inc, 1994.

TABLE DES MATIERES

AUTORISATION DE SOUTENANCE

Vu les dispositions de l'arrêté du 30 Mars 1992 relatif aux Etudes Doctorales

Vu les Rapports de présentation de :

Madame Anne-Marie TRULLEMANS ANCKAERT

Monsieur Michel ISRAEL

Monsieur Belgacem BABBA

est autorisé à présenter une thèse en soutenance en vue de l'obtention du diplôme de **Docteur de l'Institut National Polytechnique de Grenoble**, spécialité "INFORMATIQUE".

Fait à Grenoble, le

19 JUIN 1995