Nouvelles Méthodes de Synthèse Logique et Application aux Réseaux Programmables

Mohammed Belrhiti Alaoui

Thèse

Présentée par

Mohammed BELRHITI ALAOUI

Pour obtenir le grade de **DOCTEUR**

de l'**INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE**

(arrêté ministériel du 30 mars 1992)

(Spécialité : **Recherches Opérationnelles**)

==========

Nouvelles Méthodes de Synthèse Logique et Application aux Réseaux Programmables

==========

Date de soutenance : 16 décembre 1996

Composition du Jury:

Madame	Gabrièle SAUCIER	
Messieurs	Guy MAZARÉ	Président
	Giovanni De MICHELI	Rapporteur
	Wolfgang ROSENSTIEL	Rapporteur
	Jean François AGAESSE	

Thèse préparée au sein du Laboratoire de Conception des Systèmes Intégrés.

Thèse

Présentée par

Mohammed BELRHITI ALAOUI

Pour obtenir le grade de **DOCTEUR**

de l'**INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE**

(arrêté ministériel du 30 mars 1992)

(Spécialité : **Recherches Opérationnelles**)

==========

Nouvelles Méthodes de Synthèse Logique et Application aux Réseaux Programmables

==========

Date de soutenance : 16 décembre 1996

Composition du Jury:

Madame	Gabrièle SAUCIER	
Messieurs	Guy MAZARÉ	Président
	Giovanni De MICHELI	Rapporteur
	Wolfgang ROSENSTIEL	Rapporteur
	Jean François AGAESSE	

Thèse préparée au sein du Laboratoire de Conception des Systèmes Intégrés.

Remerciements

Je tiens tout d'abord à remercier Madame Gabrièle SAUCIER, professeur à l'ENSIMAG et Directrice du Laboratoire de Conception de Systèmes Intégrés, pour m'avoir accueilli dans son laboratoire, pour avoir suivi mon travail durant ces années de recherche.

Je remercie Monsieur Guy Mazaré, Professeur et Directeur de l'ENSIMAG, de me faire l'honneur de présider ce jury.

Je remercie Monsieur Giovanni De Micheli, Professeur à l'Université de Standford, et Monsieur Wolfgang Rosenstiel Professeur à l'Université de Tuebingen, pour avoir accepté d'être rapporteurs de mon travail, ce malgré le court délai accordé pour rédiger le rapport.

Je remercie Jean-François Agaesse, chef de projet à la société Thomson-CSF, pour avoir accepté d'être membre de ce jury.

Je tiens également à remercier chaleureusement tous lés membres et anciens membres du laboratoire CSI pour leur collaboration et tous ceux qui ont contribué de près ou de loin à l'élaboration de ce travail. Je remercie tout particulièrement Gilles Bosco et Bassem Sakkour pour leurs participations aux travaux exposés dans cette thèse et Khalid, Jean-Pierre, Kacem, Hichem, Ion, Viet, Sid-Ahmed, Bobach, Ali, Bernard, Thierry, Abdelkader, Aadil, Tarik et tant d'autres, pour leur amitié et leur soutien.

A mes parents.

A ma femme.

Résumé

Cette thèse propose et analyse de nouvelles méthodes de synthèse logique. L'analyse concerne des outils de la "troisième génération" d'écriture de bases irrédondantes de fonctions booléennes, à savoir les minimiseurs dits symboliques.
Cette génération de minimiseurs conduit à la solution optimale plus rapidement et avec moins d'espace mémoire que les heuristiques de la minimisation explicite. Elle permet également le calcul de la forme complémentée minimale sans être exposée à des problèmes d'explosion en complexité, ce qui permet d'aboutir à un choix efficace entre une fonction et son complément.

Nous avons abordé ensuite les problèmes de granularité des expressions factorisées. Nous avons proposé une méthode originale de réinjection qui intègre d'une façon concurrente une phase de minimisation symbolique des expressions booléennes. Cette méthode a permis de "corriger" la granularité : d'une part, des expressions booléennes obtenues par la factorisation, d'autre part, des équations obtenues par une description de haut niveau de type VHDL. La méthode proposée peut être également appliquée en tant que minimiseur logique qui tient compte du partage de la logique entre les expressions booléennes, ce qui n'est pas possible avec un minimiseur logique local ou global. Les expériences pratiques et l'application sur les réseaux programmables de type CPLD sont concluantes.

Enfin, nous avons proposé une méthode originale de l'exploration de l'espace des solutions des macro-générateurs de type additionneur. Cette méthode est fondée sur le filtrage des solutions générées et l'amélioration par dérivation d'une solution donnée. Cette approche peut être efficacement appliquée sur la macro-génération sous contraintes temporelles.

Mots clés : Synthèse logique, Minimisation symbolique, Calcul de De Morgan, Réinjection, Granularité technologique, Macro-génération sous contraintes temporelles.

Abstract

This thesis proposes and analyzes new methods of logic optimization. First of all, we attempt to provide a consistent presentation and an implementation of the last generation of Boolean minimization tools called "Symbolic Minimizer". These minimizers are a "second step" result of renewed interest in the representation of Boolean functions by Binary Decision Diagrams. They lead to optimal solutions for very complex problems in terms of product terms. Moreover, only these types of methods allow an efficient De Morgan computation. Consequently, a phase selector may choose for each function whether the direct or complemented form is suitable for further manipulation.

We propose innovative methods for Boolean functions collapsing. The starting point is either factorized equations or set of functions obtained at the output of the VHDL compiler. The granularity at that point is related to the VHDL granularity in terms of component signals. Such expressions are commonly very redundants. The proposed approach is based on iterative improvement by calling symbolic minimizer and elementary collapsing concurrently. Selection criteria for choosing a sub-function to be collapsed takes into account its Logic sharing and its complexity in terms of predicted number of cells in target technology. All the sub-functions are not processed simultaneously. Instead, a clustering strategy is proposed. Iterations of this optimization procedure allow to converge to the optimized solution. One of the application studied in more detail is synthesis on CPLD programmable Logic circuits, experimental results obtained for a significant set of examples show the interest of the proposed method.

Finally, we propose an innovative Solution Space Exploration based on the filtering of intermediate solutions of the adder macrogeneration. For a given solution, we propose a procedure of derivation based on Tabu search approach. Our method can be efficiently applied in timing constraint macrogeneration.

Key words : Logic synthesis, Symbolic Minimization, De Morgan computation, Partial collapsing, Granularity, Timing constraint macrogeneration.

TABLE DES MATIÈRES

Introduction générale

Cette thèse s'intéresse aux problèmes de synthèse logique et plus particulièrement à la préparation des phases de décomposition technologique sur des cibles du type réseaux programmables.

Dans un premier chapitre, les techniques de base de la minimisation booléenne sont revisitées. En effet, nous assistons depuis quelques années à la renaissance d'une troisième génération d'outils de synthèse logique. La première génération d'outils, fort connue, a été introduite par Quine-McCluskey [Quine59] et [McCluskey59] et par [Bartee62]. Il s'agissait d'informatiser des méthodes manuelles de minimisation logique utilisées par les ingénieurs et fondées sur des représentations graphiques du type tableau de Karnaugh. L'objectif était l'obtention de bases irrédondantes par des méthodes de couvertures de points. Ces méthodes étaient alors rigoureuses et exactes mais ne permettaient de traiter que des exemples de complexité limitée.

Une deuxième génération d'outils : [Kuntzmann68], [Hong74], [Brayton84], [McMullen86], [Rudell87], [Sicard88] [Rudell89], [Hong91], a proposé une nouvelle formulation du problème permettant d'échapper à cette notion de couverture de points. La théorie des consensus [Kuntzmann68] et [Tison67] a permis de proposer une vision fondée sur la couverture de monômes. En parallèle, l'école américaine (IBM, Berkeley) a attaqué le problème de la complexité par des propositions d'heuristiques efficaces dans un contexte informatique moderne (programme ESPRESSO).

Dans le premier chapitre, on tente de faire une présentation consistante de la troisième génération d'outils ([Brayton93], [Coudert93-b], [Swammy93]) appelés minimiseurs symboliques. Cette troisième génération d'outils a fait suite à un regain d'intérêt pour la représentation de fonctions booléennes par des graphes de décision binaires, qui est une arme puissante pour maîtriser la complexité des problèmes. Des techniques dérivées fondées sur la représentation d'un ensemble de monômes par des graphes proches du graphe de décision binaire permettent cette fois de traiter des problèmes de haute complexité

(jusqu'à des millions de monômes) et de surcroît d'atteindre des solutions optimales.

Dans le cadre de cette thèse, un minimiseur symbolique a été réalisé et son efficacité s'est mesurée par rapport à des minimiseurs de la génération précédente.

Le deuxième chapitre concerne les formes factorisées des équations booléennes. Ce chapitre fait d'abord l'état de l'art de la factorisation des fonctions booléennes fondées sur les techniques de recherche de noyaux. Ces techniques ont suscité d'importants développements au cours des années 80-90 et ont principalement été utilisées dans les outils de conception d'ASICs. Dans cette thèse, une implantation de la division algébrique a été réalisée. L'apport innovatif de ce chapitre réside dans les techniques de réinjection de fonctions booléennes qui sont en fait le processus inverse de la factorisation. Il s'agit de réinjecter des sous-fonctions booléennes à des fins de correction de granularité liée à la taille des cellules de la technologie cible.

Une formulation du problème et une implantation logicielle paramétrée sont proposées. L'application étudiée plus en détail est la synthèse sur réseaux programmables du type CPLD. Dans ce cas précis, la fonction coût considérée au moment de la réinjection est directement liée au nombre prédictif de macrocellules elles-mêmes caractérisées par une somme d'un nombre limité de monômes et d'un nombre limité de variables. L'application a donné lieu à un logiciel efficace et les performances de l'implantation finale sur deux types de CPLD (AMD/MACH et XILINX) sont analysées.

Enfin, le dernier chapitre s'intéresse à un problème complémentaire, à savoir la génération et la synthèse de macroblocs. Il faut remarquer que dans la synthèse automatique de circuit à partir d'un langage de haut niveau (VHDL, Verilog), la majorité du circuit est composée de blocs RTL (bloc arithmétique, compteur, décodeur, multiplieur, ...). Il est important que ces blocs soient synthétisés de façon efficace. Leur synthèse ne répond pas aux mêmes critères que la logique aléatoire. Pour chacun des blocs considérés, il est nécessaire de connaître parfaitement l'état de l'art pour le bloc considéré. Dans ce chapitre, on s'intéresse uniquement à la synthèse d'additionneurs qui est bien entendu un bloc de base fondamental. L'objectif de ce chapitre est de proposer une méthode innovative d'exploration de l'espace des solutions en terme de complexité

surface/performance. En effet, dans les outils de synthèse récents, l'accent est mis sur l'obtention de contraintes d'optimisation temporelle sophistiquées et précises. Il est alors important de pouvoir proposer des macroblocs répondant exactement aux critères fixés. Il est proposé un environnement spécialisé de génération et filtrage de solutions efficaces.

En conclusion, cette thèse reprend les techniques de base de la synthèse logique (minimisation, factorisation), tout en reconnaissant l'importance des macroblocs et la nécessité des méthodes spécialisées.

Chapitre I. Écriture polynomiale et minimisation de fonction booléenne

I.1. Introduction

Depuis l'apparition des premières techniques informatisées de minimisation logique dans les années 50 ([Quine52], [Quine59], [McCluskey59], [Bartee62]), de nombreux travaux successifs ont cherché à surmonter la complexité de ce problème. Toutes ces approches ont été confrontées à deux difficultés majeures: la première est le nombre de monômes premiers, la seconde est la résolution du problème sous-jacent de couverture qui est NP-COMPLET [Garey79]. Avec le regain d'intérêt pour la représentation de fonctions booléennes par des graphes de décision binaires [Akers78], des techniques dérivées fondées sur la représentation d'un ensemble de monômes par des graphes proches du graphe de décision binaire ([Coudert92-a], [Coudert92-b], [Coudert93-a], [Minato93]), permettent cette fois de traiter des problèmes de haute complexité (jusqu'à des millions de monômes) et de surcroît d'atteindre des solutions optimales. Ces techniques dites symboliques ont connu un essor considérable dans les années 90 ([Brayton93], [Coudert93-b], [Swamy93]).

Dans ce chapitre, on s'intéresse à l'écriture irrédondante de fonctions booléennes en sommes de monômes. Tout d'abord, les deux représentations symboliques, concernant les fonctions booléennes et les ensembles de monômes sont exposées. Ensuite, le problème général de la minimisation logique est discuté; un algorithme et une implantation logicielle fondés sur la méthode présentée dans [Coudert94] sont proposés. Enfin, l'analyse des résultats permet de montrer l'intérêt pratique de l'application de cette génération de méthodes.

I.2. Rappel sur l'algèbre de Boole

I.2.1. Définitions préliminaires

• Algèbre de Boole

L'ensemble B = {0,1} muni des opérations logiques binaires ET (.), OU (+) et de l'opération unaire NON (! ou $\bar{}$) est une algèbre appelée ALGÈBRE DE BOOLE.

• Variable booléenne

Soit B = {0,1} l'espace de Boole. Une variable booléenne générale est un n-uplet $(x_1, x_2, ..., x_n)$ de B^n. Une variable booléenne simple représente une coordonnée (ou un point) dans B. Une variable booléenne générale peut prendre 2^n valeurs possibles.

• Littéral

Un littéral est une variable booléenne simple dans sa forme directe ou complémentée.
a et $\bar{a}$ sont deux littéraux distincts.

• Monôme

Un monôme m (dit également cube) est un produit de p variables simples distinctes sous forme normale (x) ou complémentée ($\bar{x}$), p est le degré (ou longueur) du monôme, et sa taille est le nombre de points qu'il couvre.
$m = a.b.\bar{c}$ est un monôme alors que $a.\bar{a}$ ne l'est pas, m est de degré 3 et de taille 1.

• Fonction booléenne

Une fonction booléenne F est une fonction de B^n dans B^m. Si m=1, c'est une fonction booléenne simple, si m > 1, F est générale.

• Points couverts

Les points couverts par une fonction booléenne F, sont ceux pour lesquels la valeur de F vaut 1.

• *Fonction φ-booléenne*

Une fonction booléenne F est dite φ-booléenne si elle est définie par :

$F : B^n \rightarrow Y^m$ ou $Y = \{0,1,\phi\}$

ϕ représente la ***Valeur Indéfinie*** (appelée "*Don't Care*" en anglais) et signifie que la fonction peut prendre indifféremment la valeur logique 0 ou 1.

En général, une fonction booléenne F peut être définie par trois ensembles disjoints :

$C_1(F)$, $C_0(F)$, $C_\phi(F)$ représentent respectivement la couverture à 1, 0 et ϕ d'une fonction F, c'est-à-dire l'ensemble des points pour lesquels la valeur de la fonction est 1, 0 et ϕ.

La borne inférieure (notée Inf (F)) et la borne supérieure (notée Sup(F)) de F sont définies de la façon suivante:

$$\text{Inf}(F) = C_1(F)$$
$$\text{Sup}(F) = C_1(F) \cup C_\phi(F)$$
$$\text{Inf}(F) \subseteq F \subseteq \text{Sup}(F)$$

• *Expression booléenne*

Toute fonction booléenne simple est exprimable par une expression booléenne. Une expression booléenne est construite à partir des variables booléennes simples de la fonction et de l'ensemble des opérateurs logiques. Une expression booléenne est dite polynomiale si elle est sous forme de somme de monômes. Elle est sous forme factorisée si elle est sous forme de produit de sommes de monômes.

• *Relation d'ordre entre monômes*

On dit qu'un monôme m_1 est inférieur à un monôme m_2, que l'on note $m_1 \subseteq m_2$, si tous les points couverts par m_1 sont couverts par m_2.

On dit aussi que m_1 est un multiple de m_2.

$m_1 = a.b.c.d$ $m_2 = a.b$ $m_1 \subset m_2$

• *Relation d'ordre sur les fonctions (phi-) booléennes*

Étant données deux fonctions (phi-) booléennes F et G, une relation d'ordre sur ces fonctions est définie par :

$F \subseteq G \Leftrightarrow \{F.G = F;\ F + G = G\}$ ($\{F \subseteq G <=> \text{Inf}(F) \subseteq \text{Sup}(G)\}$)

Autrement dit, tout point couvert par F (la borne inférieur de) l'est par G (la borne supérieur de). Cette relation d'ordre est partielle.

• *Support d'une expression*

Le support d'une expression d'une fonction booléenne F noté SUPP(F) est défini par :

$$SUPP(F) = \{a / \exists \text{ un monôme } m \subseteq F \text{ tel que } a \in m \text{ ou } \bar{a} \in m\}$$

• *Réseau de sous-fonctions booléennes*

Un réseau de sous-fonctions booléennes est un graphe acyclique orienté représentant une fonction booléenne dans lequel chaque noeud i est associé à une variable y_i et à une expression associée à une sous-fonction f_i de f.

Dans le graphe, deux noeuds i et j sont connectés par un arc (i, j) si $y_i \in supp(f_j)$.

Les entrées primaires $x_1, x_2, ..., x_n$ et les sorties primaires $z_1, z_2, ..., z_n$ correspondent à des noeuds spéciaux dans le graphe: les noeuds sources et les noeuds finaux. Aucune expression booléenne n'est associée à ces noeuds.

L'entrance d'un noeud i est l'ensemble de tous les noeuds pointant vers i. La sortance d'un noeud i est l'ensemble de tous les noeuds vers lesquels le noeud i pointe. L'entrance transitive d'un noeud i est définie récursivement par l'union de ses noeuds d'entrance transitive.

Il existe une correspondance biunivoque entre une fonction dans le réseau booléen et un noeud du graphe.

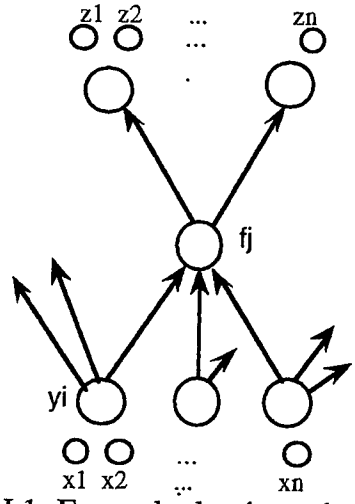


Figure I.1: Exemple de réseau booléen

D'après la figure I.1, on peut constater que :

fj a une sortance de 2 et une entrance de 3.

fj apparaît dans les expressions correspondant aux noeuds destination (fan-out) et l'expression de fj est exprimée en fonction des noeuds source (fan-in).

I.2.2. Base Irrédondante

• *Monôme canonique*

Un monôme canonique est un monôme qui couvre un seul point de Sup(f), donc contient toutes les variables du support de la fonction.

• *Monôme premier*

Un monôme m est premier dans une fonction (phi-) booléenne f si et seulement si :

- $m \subseteq f$ ($m \subseteq Sup(f)$).
- Il n'existe pas de monôme m' différent de m tel que:

  $m' \subseteq f$ ($m' \subseteq Sup(f)$) et $m \subseteq m'$

L'ensemble des monômes premiers de f est noté Premier (f) (Premier (Sup(f))).

• *Base d'une fonction (phi-) booléenne*

Une base d'une fonction booléenne est la somme S de monômes premiers égale à la fonction f ($Inf(f) \subseteq S \subseteq Sup(f)$). ***Une base complète*** est la somme de tous les monômes premiers. ***Une base est irrédondante*** si elle cesse d'être une base dès qu'on lui enlève un monôme.

• *Monôme premier obligatoire*

Un monôme premier est dit monôme premier obligatoire d'une base d'une fonction (phi-) booléenne f si et seulement si il est le seul des monômes de la base à couvrir au moins un des points (de la borne inférieur Inf(f)) de la fonction. Tous les monômes d'une base irrédondante sont obligatoires.

• *Monôme premier essentiel*

Un monôme premier est dit monôme premier essentiel d'une fonction (phi-) booléenne f s'il est obligatoire dans la base complète. Il est le seul à couvrir un des points couverts par (Inf(f)) f parmi tous les monômes premiers de la fonction. Un monôme premier essentiel appartient donc forcément à toutes les bases.

I.3. Représentation en diagramme de décision binaire (BDD, ROBDD)

Les graphes de décision binaires ont été proposés par LEE [Lee59] et par la suite étudiés par [Kuntzmann68] et [Akers78]. Les algorithmes de manipulation des graphes de décision binaires ont été largement discutés par [Bryant86]. Cette représentation a l'avantage d'être canonique par rapport à une forme polynomiale ou une forme factorisée pour un ordre donné des variables. La complexité des graphes de décision binaire est directement liée à l'ordre des variables [Bryant86]. Malheureusement, le problème de trouver un ordre optimal des variables est NP-Complet ([Bryant86], [Friedman87], [Boolig94]), pouvant entraîner l'échec de la construction du graphe de décision binaire.
Par la suite, nous exposons rapidement l'état de l'art sur les approches proposées pour aborder ce problème, en mettant plus l'accent sur celles appliquées dans le cadre de l'algorithme implanté de la minimisation symbolique.

I.3.1. Construction du ROBDD

Le graphe de décision binaire (dorénavant noté BDD) est construit à partir de la décomposition de Shannon ([Boole1854], [Shannon49]) et décrite par le théorème suivant:

Théorème de Shannon:
Une fonction booléenne F se décompose d'une façon unique, par rapport à une variable x selon l'expression :
$F = x.F_x + \overline{x}.F_{\overline{x}}$
où F_x est l'expression de F pour x=1 et $F_{\overline{x}}$ celle pour x=0. F_x et $F_{\overline{x}}$ sont appelés cofacteurs de F par rapport à x et $\overline{x}$. La décomposition de Shannon est illustrée dans la figure I.2.

Cette décomposition est récursive. En l'itérant sur les cofacteurs F_x et $F_{\overline{x}}$, par rapport aux autres variables restant du support, on obtient un arbre binaire, appelé ***graphe de décision binaire*** ou ***arbre de Shannon***, dont les feuilles sont les constantes 0 et 1.

L'unicité de la décomposition de Shannon implique la canonicité de la représentation en forme de BDD et ceci pour un ordre donné des variables[Akers78].

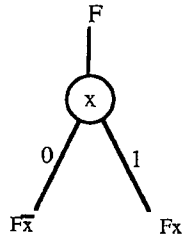


Figure I.2 : Règles de décomposition de Shannon

Pour éliminer les noeuds redondants du BDD, on lui applique la première règle de réduction ([Akers78], [Bryant86]) décrite dans la figure I.3.

Règle 1: Si un noeud a comme successeur 2 feuilles étiquetées par la même valeur, ce noeud peut être supprimé.

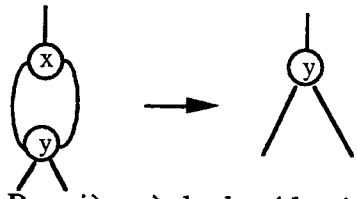


Figure I.3 : Première règle de réduction du BDD

Un graphe de décision binaire G est dit ordonné si et seulement si la séquence de décomposition de Shannon suit le même ordre des variables sur tous les chemins de G reliant la racine aux feuilles. Il est noté OBDD. La figure I.5 représente un OBDD où l'ordre des variables de la décomposition est (a >> b >> c >> d).
La représentation d'une fonction par un OBDD peut être simplifiée par la règle de simplification décrite dans la figure I.4.

Règle 2: Si 2 noeuds sont prédécesseurs de 2 sous-arbres identiques ils peuvent être fusionnés.

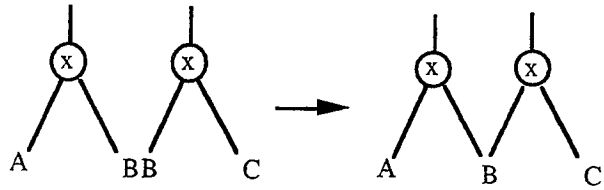


Figure I.4 : Deuxième règle de réduction du BDD

En appliquant cette règle, nous pouvons réduire la taille de la représentation en partageant les sous-arbres identiques. C'est ce qu'on appelle ROBDD (Reduced OBDD). Cette forme est canonique [Bryant86].

Exemple

Soit la fonction $f = a.(c \oplus d) + b.\overline{(c \oplus d)}$

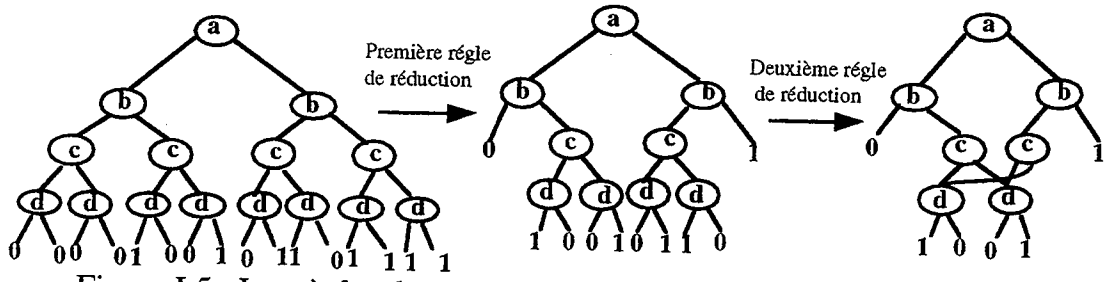


Figure I.5 : Les règles de réduction du graphe de décision binaire de F

Règle 3: Passage aux ROBDDs à arcs typés ([Akers78], [Billon87], [Madre88], [Minato90]).

Considérons 2 noeuds prédécesseurs de 2 sous-arbres représentant respectivement une sous-fonction SF et son complément $\overline{SF}$. Ces deux sous-arbres sont fusionnés en un seul. Les noeuds y sont connectés respectivement par un arc normal et un arc étiqueté par un inverseur. La figure I.6 illustre l'introduction de l'arc typé dans le ROBDD de F.

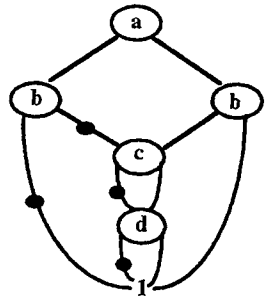


Figure I.6 : Introduction des arcs typés dans le ROBDD de F

Pour conserver la canonicité de la représentation, il faudrait prendre comme convention de mettre toujours l'arc qui pointe vers le fils droit sous une même forme (directe ou complémentée). La figure I.7 montre les transformations nécessaires pour assurer la canonicité.

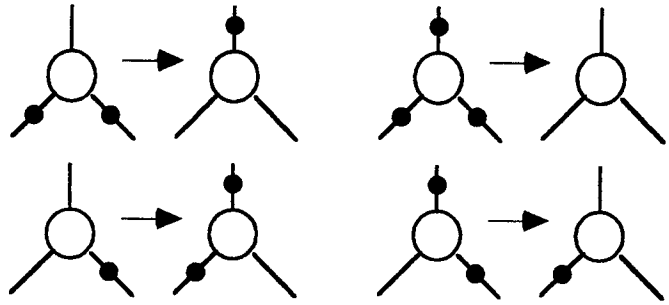


Figure I.7 : Règles de transformation pour conserver la canonicité

I.3.2. Ordonnancement des variables

La complexité de la représentation en ROBDD est directement liée à l'ordre selon lequel s'effectue la décomposition de Shannon. Il a été démontré que le problème de trouver un ordre qui minimise la taille du ROBDD est NP-Complet ([Bryant86], [Friedman87], [Bollig94]); l'algorithme exhaustif pour déterminer l'ordre optimal est de complexité O ($n^2 3^n$) [Friedman87] et [Friedman90].

Exemple:

Pour illustrer l'importance de l'ordre pour la construction du ROBDD, considérons la fonction booléenne suivante:

$F = x_1.x_2 + x_3.x_4 + x_5.x_6 + \ldots + x_{2n-1}.x_{2n}$

Si on suppose que l'ordre est $x_1, x_2, \ldots, x_{2n-1}, x_{2n}$ (*) de la figure I.8, le ROBDD généré contient 2.n + 2 noeuds alors que l'ordre $x_1, x_3, x_5, \ldots, x_{2n-1}, x_2, x_4, \ldots, x_{2n-2}, x_{2n}$ (**) de la figure I.8 génère un graphe de $O(2^n)$noeuds. La complexité du ROBDD passe d'une taille linéaire à exponentielle rien qu'en changeant l'ordre. Pour n = 3, les deux ROBDDs sont les suivants:

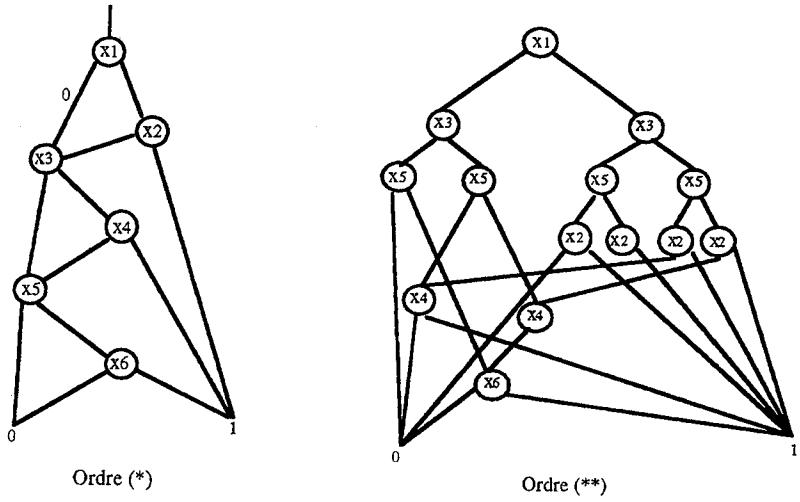


Figure I.8 : Impact de l'ordre sur la complexité du ROBDD

Il existe dans la littérature plusieurs approches pour aborder ce problème, on peut citer :

1. Les méthodes exactes et exactes améliorées [Friedman87]

Ces méthodes énumèrent tous les ordres, retiennent ensuite le meilleur. Il existe des raffinements lors de l'exploration de l'espace des solutions grâce aux résultats intermédiaires, qui permettent d'obtenir une borne inférieure pour éviter un parcours complet. Après raffinement, on peut atteindre l'optimum pour des exemples allant jusqu'à douze entrées.

2. Des méthodes fondées sur les propriétés des fonctions booléennes

Parmi les propriétés les plus utilisées: L'occurrence et l'occurrence inverse des variables, la plus petite profondeur, ...etc. Ces méthodes tentent en général de favoriser le partage pour réduire la complexité du ROBDD ([Jeong92], [Besson93-a], [Besson93-b]).

3. Des heuristiques fondées sur des informations venant d'une réalisation multicouches préalable des fonctions booléennes [Malik88] [Minato90].

Elles utilisent la structure topologique d'un circuit de portes interconnectées.
Parmi les heuristiques les plus performantes, on peut citer les méthodes d'assignement de poids statique et dynamique [Minato90].
L'idée générale dans ces deux méthodes est de trouver les variables les plus importantes (comme dans le cas de la méthode basée sur les occurrences), ou celles qui ont le plus de "poids". La méthode d'assignement de poids statique est en réalité un cas particulier de la méthode dynamique.

L'algorithme général se déroule comme suit: Une constante (1 en général) est propagée depuis le sommet de la fonction jusqu'à ses feuilles. Comme l'illustre la figure I.9, à un noeud donné n_i, cette constante se partage équitablement entre les fils du noeud n_i, et ainsi de suite jusqu'aux entrées primaires. Dans le cas statique, on ordonne l'ensemble des variables primaires (les feuilles du graphe) dans l'ordre décroissant de leur poids. Cet ordre est celui recherché.
Dans le cas de l'assignement de poids dynamique, on réalise pour la première étape le même partage de la constante jusqu'aux racines. Mais ici, on sélectionne la variable qui a le plus grand poids, on la supprime dans tout le graphe, et on réitère l'opération jusqu'à la dernière variable. La méthode statique revient en fait à arrêter la méthode dynamique à la première étape.

Exemple:
Considérons la fonction booléenne : $F = a.\bar{b} + a.b.\bar{c} + a.\bar{c}$

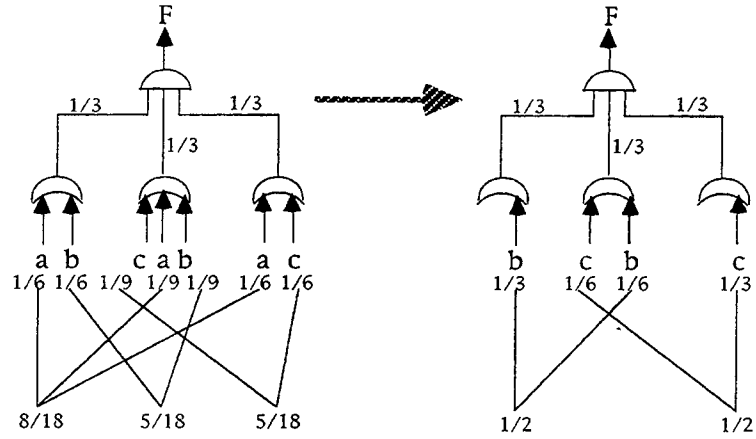


Figure I.9 : Assignement de poids dynamique.

Dans le cas de l'assignement statique (premier schéma) comme dans le cas de l'assignement dynamique, on obtient l'ordre:

$a >> (b, c)$.

L'efficacité de cet algorithme est illustrée par l'exemple donné précédemment:

$F = x_1.x_2 + ... + x_{n-1}.x_n$

En effet, l'ordre trouvé sera:

$x_1 >> x_2 >> ... >> x_n$

Cet ordre correspond à l'ordre optimal. Avec l'assignement statique, on n'aurait pas pu trouver cet ordre, car le poids de toutes les variables est le même à la première étape de l'assignement. La méthode des occurrences aurait aussi échoué, la relation d'ordre n'étant que partielle pour cet exemple.

Les résultats de cette approche sont médiocres dans le cas où la structure du circuit n'est pas bien optimisée.

4. Des méthodes par améliorations itératives fondées sur des échanges de variables ([Calazans91], [Calazans92], [Felt93], [Fujita91], [Ishiura91], [Jacobi91], [Jacobi93], [Mercer92]).

Ces méthodes sont destinées à améliorer un ordre initial résultat d'une approche heuristique. Les échanges de variables peuvent intervenir une fois que la construction du ROBDD est achevée, ou pendant la construction, le "sifting" de [Rudell93]. La qualité des résultats dépend donc de la solution initiale.
Parmi les méthodes les plus performantes fondées sur des échanges de variables on peut citer :

- Méthodes par amélioration incrémentale

L'idée de base de cette méthode est d'échanger deux variables adjacentes du ROBDD, afin de produire un nouvel ordre qui conduit à un nouveau ROBDD (figure I.10). Dans la suite, nous appellerons *SWITCH(i)* la fonction qui échange deux variables adjacentes aux niveaux i et $i+1$.
L'échange de l'ordre des variables implique la mise à jour du ROBDD. Cette opération n'impliquera pas la reconstruction complète du ROBDD, mais une modification locale de sa structure, et le cas échéant, des simplifications et réductions de la représentation. Si l'on considère une fenêtre de taille 2, seuls les noeuds présents dans cette fenêtre sont susceptibles d'être modifiés. De fait, l'échange de deux variables n'obligera pas à reconstruire tout le ROBDD [Fujiwara85].

Équation associée à l'échange des deux variables adjacentes x et y:

$$F = x.F_x + \bar{x}.F_{\bar{x}} = x.\left(y.F_{x_y} + \bar{y}.F_{x_{\bar{y}}}\right) + \bar{x}.\left(y.F_{\bar{x}_y} + \bar{y}.F_{\bar{x}_{\bar{y}}}\right)$$

donc

$$F = y.\left(x.F_{x_y} + \bar{x}.F_{\bar{x}_y}\right) + \bar{y}.\left(x.F_{x_{\bar{y}}} + \bar{x}.F_{\bar{x}_{\bar{y}}}\right)$$

Cette équation implique, suivant la configuration de départ, l'ensemble des modifications décrites dans figure I.11.

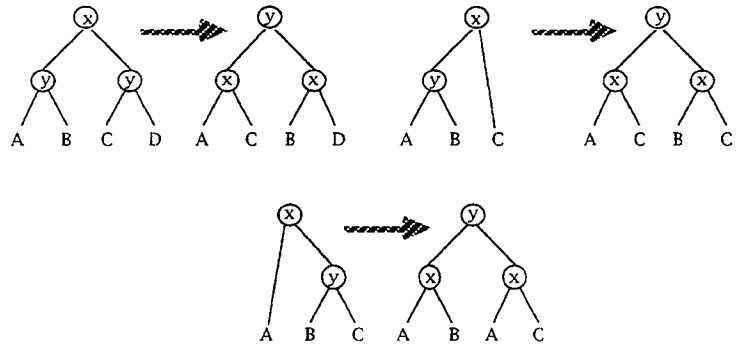


Figure I.10 : Opération *SWITCH*(*i*).

L'échange de deux variables adjacentes a donné lieu à l'élaboration de plusieurs méthodes de minimisation de la taille d'un ROBDD [Fujita91]. Une extension à été proposée en généralisant la méthode à m variables adjacentes au lieu des deux classiques [Ishiura91].

Exemple:

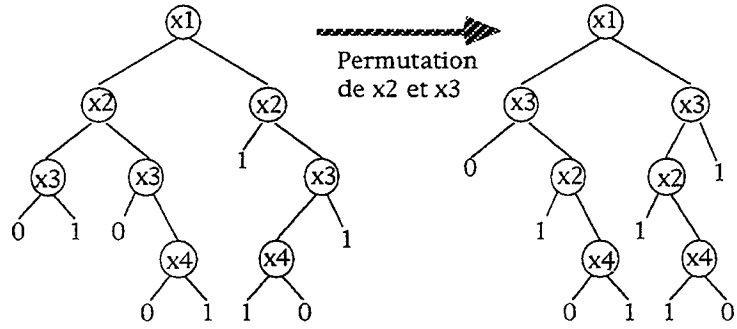


Figure I.11: Réduction de la taille par échange de noeuds.

- Méthode par fenêtre glissante[Besson96]

Cette approche consiste à réaliser l'opération sur une fenêtre glissante de taille W qui parcourt l'ensemble des variables en permettant l'échange de la variable en position i avec toutes les variables de position allant de $i+1$ à $i+W$. L'algorithme consiste à calculer la taille du ROBDD lorsque sont échangées deux variables comme décrit dans la figure I.11. Si la taille est plus faible, on conserve cet ordre et l'on réitère l'opération pour toutes les variables de la fenêtre avant de la faire glisser (figure I.12).

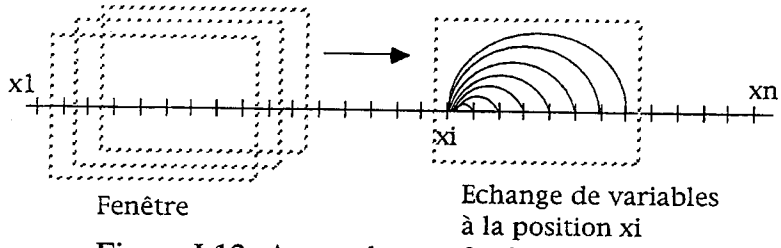


Figure I.12: Approche par fenêtre glissante.

Il existe cependant des fonctions booléennes qui n'admettent pas d'ordre permettant d'obtenir un ROBDD de taille polynomiale. Ceci est démontré en particulier pour les multiplieurs combinatoires. En effet, il existe au moins une sortie primaire qui a un ROBDD de taille exponentielle pour tous les ordres possibles des variables [Bryant88].

I.4. Représentation d'un ensemble des monômes par MBDD

Nous nous intéressons dans ce paragraphe à une représentation compacte d'un ensemble de monômes qui, en l'occurrence, seront des monômes associés à une expression polynomiale des fonctions booléennes. On notera MBDD (appelé également Zero-suppresed BDD [Minato93]) le graphe de représentation des monômes. La ressemblance avec l'appellation BDD vient du fait que des techniques binaires similaires à celles utilisées pour la construction des ROBDD sont appliquées. La lettre M rappelle qu'il s'agit d'énumérer des monômes.

I.4.1. Construction du MBDD [Minato93]

Soit P un ensemble de monômes de support $\{x_k\}$, P peut se décomposer suivant que ses monômes contiennent ou pas la variable x_k :

$P = \{x_k\} \otimes P_{xk} \cup \{\bar{x}_k\} \otimes P_{\bar{x}k} \cup P_{1k}$

avec

$$P \otimes Q = \{p.q \,/\, p \in P, q \in Q\},$$

et

$P_{xk} = \{p \,/\, x_k.p \in P\}$

$P_{\bar{x}k} = \{p \,/\, \bar{x}_k.p \in P\}$

$P_{1k} = \{p \,/\, p \in P,$ p monôme ne contenant ni x_k ni $\bar{x}_k\}$

Un MBDD est construit de façon itérative (voir figure I.12).

Soit x_k la variable ordonnée en première position. La règle d'itération nous permet de construire le noeud x_k et son noeud complément $\bar{x}_k$ de la façon suivante:

a) Au noeud successeur droit de x_k est associé l'ensemble des monômes MBDD(P_{xk}) (monômes contenant x_k).

b) Au noeud successeur gauche de x_k est associé $\bar{x}_k$. Ce noeud a comme successeurs:

- Le noeud droit associé à l'ensemble des monômes MBDD($P_{\bar{x}k}$) (monômes contenant $\bar{x}_k$)
- Le noeud gauche associé à l'ensemble des monômes MBDD(P_{1k}) (monômes ne contenant ni x_k ni $\bar{x}_k$.)

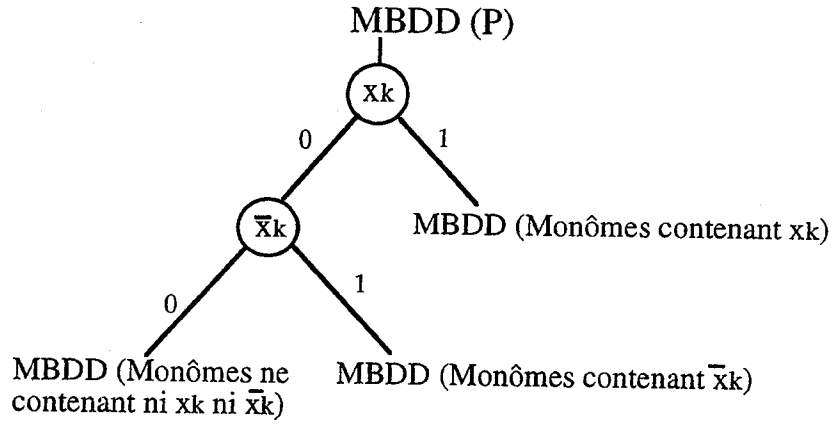


Figure I.12 : Construction du MBDD de P

Exemple:

$P = \{x_1.\bar{x}_4,\ x_2,\ x_2.x_4,\ \bar{x}_2.x_3\}$;

une décomposition par rapport à la variable x_2 donne les ensembles suivants:

$P_{x2} = \{1,\ x_4\}$, $P_{\bar{x}2} = \{x_3\}$, $P_{12} = \{x_1.\bar{x}_4\}$

La figure I.13 décrit le MBDD de P.

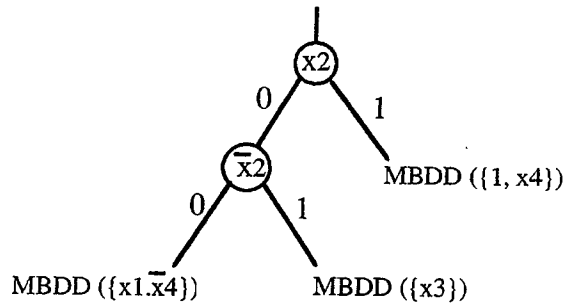


Figure I.13: MBDD de P

Interprétation et lecture du graphe

Dans le graphe chaque chemin qui va de la racine à une feuille '1' (1-terminal) correspond à un monôme (un tel chemin est appelé 1-chemin). Pour exprimer explicitement l'ensemble de monômes à partir d'une représentation MBDD, on part de la racine; si on passe par le 1-arc alors le littéral représenté par la racine appartiendra au monôme associé au 1-chemin, sinon on passe par le 0-arc. Il y aura alors autant de littéraux dans le monôme que de 1-arc dans le 1-chemin qui représente ce monôme. Le noeud 0-feuille représente l'ensemble vide.

Exemple:

Soit l'ensemble de monômes suivant:

$$P = \{x_1.x_2,\ \bar{x}_1.\bar{x}_2.x_3,\ x_3\}$$

Le MBDD associé à cet ensemble est montré dans la figure I.14.

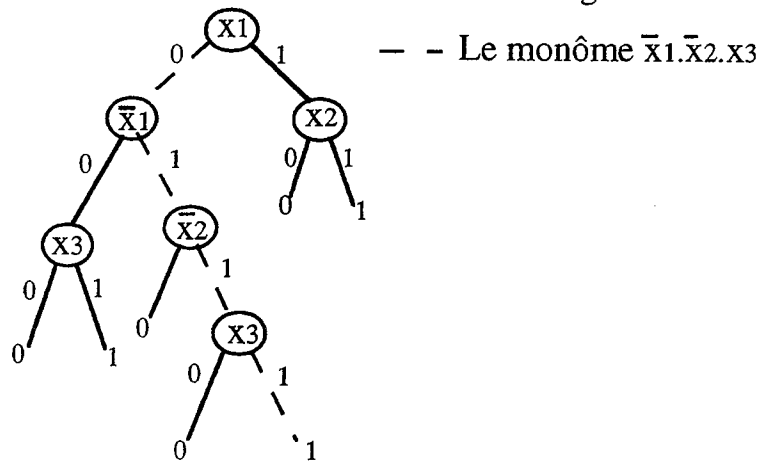
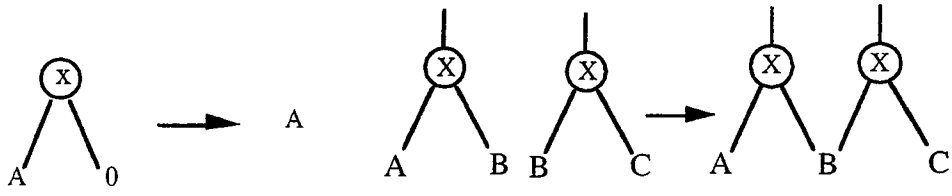


Figure I.14 : Représentation de P en MBDD

I.4.2. Les règles de réduction du MBDD

La première règle consiste à éliminer tous les 1-arc qui pointent vers une feuille étiquetée par la valeur 0. La deuxième règle de réduction est identique à celle appliquée pour les ROBDD, elle consiste à partager les sous-graphes identiques. Ces deux règles sont illustrées dans la figure I.15.



Première régle de réduction — Deuxième régle de réduction

Figure I.15 : Règles de réduction du MBDD

Exemple:

Soit $P = \{x_1.x_2,\ \bar{x}_1.\bar{x}_2,\ x_2\}$

En prenant l'ordre des variables $x_1 >> \bar{x}_1 >> x_2 >> \bar{x}_2$, le graphe initial ainsi que les graphes réduits sont représentés en figure I.16 :

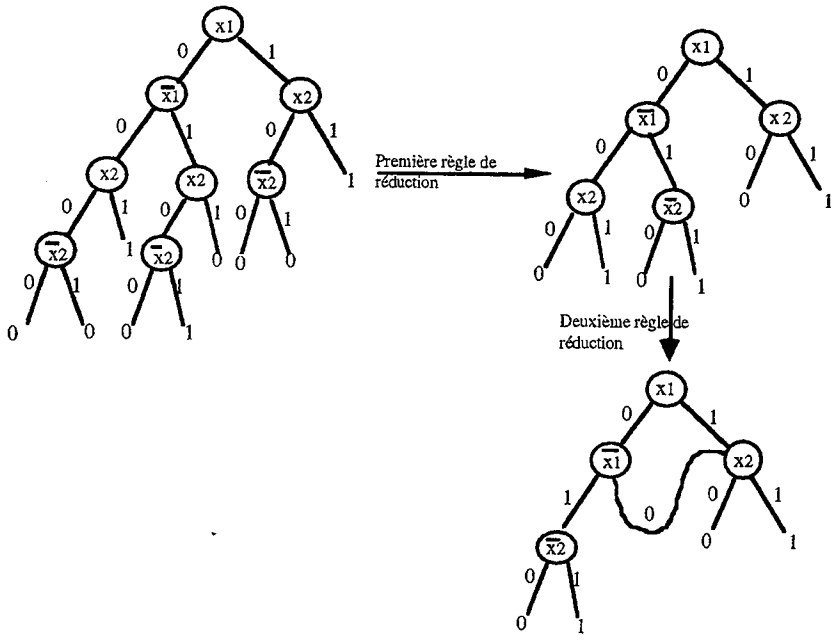


Figure I.16 : Application des règles de réduction

Les règles de réduction sont appliquées pendant la construction du MBDD à chaque création d'un nouveau noeud.

I.4.3. Opérations ensemblistes sur les MBDD

Soit la procédure CréerNoeud(top, PThen, PInv, PElse).

-"PThen" représente le graphe du MBDD des monômes contenant la variable sous sa forme normale et de rang "top".

-"PInv" représente le graphe du MBDD des monômes contenant la variable sous sa forme complémentée et de rang "top".

- "PElse" représente le graphe du MBDD qui ne contient pas la variable de rang "top".

Au cours du déroulement de cette procédure les règles de réduction du MBDD sont appliquées simultanément.

Comme pour les ROBDD [Bryant92], la manipulation des MBDD passe par les opérations de base, à savoir l'union, l'intersection et la disjonction (partie non commune) de sous-ensembles. Ces opérations ensemblistes sont utilisées très

souvent, dans la recherche des monômes premiers (la disjonction) et dans le calcul du noyau cyclique (l'union et l'intersection). Toutes ces procédures font appel à celle qui permet de créer un noeud CréerNoeud. Dans la description qui suit, P.top désigne le rang de la variable qui a la plus forte priorité dans le MBDD P. Pour préserver l'ordre des variables dans le MBDD résultat, pour chaque opération, il est nécessaire de comparer les rangs des variables de première priorité des MBDD opérandes.

```
Union (P, Q)
{
 Si (P = φ) retourner Q;
 Si (Q = φ) retourner P;
 Si (P = Q) retourner P;
 Si (P.top < Q.top) retourner CréerNoeud (P.top, PThen, PInv, Union (PElse,Q));
 Si (P.top > Q.top) retourner CréerNoeud (Q.top, QThen, QInv, Union (P, QElse));
 Si (P.top = Q.top)
   retourner CréerNoeud (P.top, Union (PThen, QThen),
                              Union (PInv, QInv), Union (PElse, QElse));
}

Intersec (P, Q)
{
 Si (P = φ) retourner φ ;
 Si (Q = φ) retourner φ;
 Si (P = Q) retourner P;
 Si (P.top < Q.top) retourner Intersec (PElse, Q);
 Si (P.top > Q.top) retourner Intersec (P, QElse);
 Si (P.top = Q.top)
    retourner CreerNoeud (P.top, Intersec (PThen, QThen),
                              Intersec(PInv, QInv), Intersec(PElse, QElse));
}
```

```
Disjonct (P, Q)
{
  Si (P = φ) retourner φ;
  Si (Q = φ) retourner P;
  Si (P = Q) retourner φ;
  Si (P.top < Q.top) retourner CréerNoeud (P.top, PThen, PInv, Disjonct (PElse, Q));
  Si (P.top > Q.top) retourner Disjonct (P, QElse);
  Si (P.top = Q.top)
      retourner CréerNoeud (P.top, Disjonct(PThen, QThen),
                            Disjonct (PInv, QInv), Disjonct (PElse, QElse));
}
```

Ces algorithmes sont accélérés en utilisant des caches de mémoire. Ils permettent la sauvegarde des résultats des dernières opérations. En cherchant dans le cache avant chaque appel récursif, si le calcul a déjà été réalisé, on peut éviter de le refaire pour des sous-graphes équivalents. Avec cette technique, on peut exécuter des opérations en un temps quasi proportionnel à la taille des graphes.

Les représentations symboliques de somme de monômes ou d'ensemble de monômes ont été présentées (ROBDD, MBDD). Dans le paragraphe suivant on expose le problème de la minimisation booléenne et on aborde le traitement de ce problème à l'aide de ces représentations.

I.5. Minimisation symbolique d'une fonction simple

I.5.1. Définition du problème de la minimisation

La minimisation logique consiste à trouver la représentation minimale en terme d'ensemble de monômes d'une fonction booléenne. Ce problème peut se ramener à celui de la couverture suivant [McCluskey59]:

Soit P et Q deux ensembles de monômes. On note le problème de couverture <P, Q> qui consiste à trouver un sous-ensemble S de cardinalité minimale tel que pour tout monôme q élément de Q il existe un monôme de S qui le couvre.
Pour mieux illustrer, on peut citer le cas particulier du problème de couverture <P, Q>, qui est représenté par la matrice de couverture de points, ayant comme lignes Q (l'ensemble des monômes canoniques d'une fonction booléenne f), et les colonnes P (l'ensemble des monômes de la représentation polynomiale de f).

La case (p, q) est égale à 1 si et seulement si le monôme q est couvert par le monôme p. Le problème de couverture <P, Q> consiste à trouver un sous-ensemble de colonnes de cardinalité minimale qui couvre toutes les lignes.

Exemple:
Considérons la fonction booléenne suivante :
$f = b.\bar{d} + a.b + b.\bar{c} + a.d + \bar{b}.c.d$

$\subseteq$	$b.\bar{d}$	a.b	$b.\bar{c}$	a.d	$\bar{b}.c.d$
0100	1		1		
1100	1	1			
0101			1		
1101		1	1	1	
1001				1	
0011					1
1111		1		1	
1011				1	1
0110	1				
1110	1	1			

Exemple de matrice de couverture

Considérons une fonction booléenne F. On note Premier (Sup(F)) l'ensemble des monômes premiers de la borne supérieure de F. La minimisation de F revient donc à résoudre **le problème de couverture <Premier (Sup(F)), F>**.

I.5.2. Algorithme de Quine-McCluskey

L'algorithme de Quine-McCluskey passe par les étapes suivantes pour trouver la représentation minimale de la fonction booléenne F :

1- Calculer les monômes premiers de F.
2- Construire la matrice de couverture dont les lignes sont les monômes canoniques de F, les colonnes sont les monômes premiers de F.
3 - Trouver un sous-ensemble de coût minimal de colonnes qui couvre tous les points de la fonction.
La première tâche a une complexité exponentielle, le nombre de monômes premiers d'une fonction peut atteindre $O(\frac{3^n}{\sqrt{n}})$ [Quine53].

La deuxième tâche a de même une complexité exponentielle.
La troisième tâche est un problème de couverture à coût minimal, qui est NP-COMPLET ([Garey79], [Xuong91]).
La première tâche sera traitée, en utilisant la représentation implicite des monômes MBDD .

I.5.3. Calcul implicite des monômes premiers

Beaucoup de travaux ont été consacrés à un algorithme efficace pour calculer les monômes premiers. Karnaugh [Karnaugh53] a développé une technique manuelle basée sur une représentation des fonctions booléennes par tableaux. Les monômes premiers sont les plus grands groupements de 2^k points dans le tableau. Le nombre d'entrées est limité à 5. Des techniques informatisées étaient élaborées par Quine et McCluskey [McCluskey59], fondées sur l'expression de la fonction booléenne en monômes canoniques qui permet le calcul des monômes premiers. L'expression explicite en monômes canoniques basée sur le consensus pose des problèmes d'explosion.
Une nouvelle théorie a été développée par Tison et Kuntzmann ([Tison67], [Kuntzmann68]). Elle permet de déterminer tous les monômes premiers à partir d'un ensemble quelconque de monômes couvrant la fonction booléenne. Cette méthode n'a pas besoin de redescendre aux points élémentaires (les monômes canoniques) de la fonction, une amélioration de cette méthode a été présentée dans [Loui82]. Une autre méthode, fondée sur la résolution sémantique, a été exposée dans[Slage69] et [Slage70].

La majorité de ces travaux sont fondés sur l'amélioration incrémentale de l'algorithme de Quine. Il consiste à prendre un monôme de f, essayer de lui enlever le plus possible de littéraux en préservant sa couverture par f. Ces procédures calculent un monôme premier à la fois. Ces techniques sont limitées à des fonctions booléennes ne dépassant pas 20000 monômes premiers [Coudert94].

La puissance de la méthode symbolique est son aptitude à manipuler un plus grand nombre de monômes premiers, et ceci grâce à la représentation compacte du MBDD[Coudert92-b].
L'algorithme de calcul de monômes premiers implémenté dans le cadre de cette thèse est fondé sur la décomposition canonique de la fonction booléenne f

(théorème de Shannon) et la représentation de l'ensemble des monômes par les MBDD.

Théorème:

Soit f une fonction booléenne, on note Premier (f) l'ensemble des monômes premiers de f, le calcul de Premier (f) se fait à l'aide des formules suivantes:

$$(a)\ (\text{Premier}\ (f))_{1k} = \text{Premier}\ (f_{\bar{x}k}\ .\ f_{xk})$$
$$(b)\ (\text{Premier}\ (f))_{\bar{x}k} = \text{Premier}(f_{\bar{x}k}) - (\text{Premier}(f))_{1k}$$
$$(c)\ (\text{Premier}\ (f))_{xk} = \text{Premier}(f_{xk}) - (\text{Premier}(f))_{1k}$$

et d'après la formule d'expansion (voir figure I.17):

$$\text{Premier}\ (f) = \{xk\} \otimes (\text{Premier}\ (f))_{xk} \cup \{\bar{x}k\} \otimes (\text{Premier}\ (f))_{\bar{x}k} \cup (\text{Premier}\ (f))_{1k}$$

La preuve de ce théorème est donnée dans [Coudert94].

Si on note P les éléments de Premier ($f_{xk}.f_{\bar{x}k}$), P0 les éléments de Premier ($f_{\bar{x}k}$) et P1 ceux de Premier (f_{xk}), on crée les noeuds suivants:

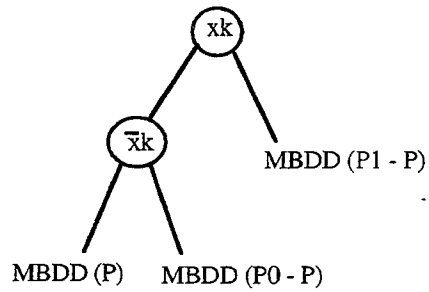


Figure I.17 : Représentation des monômes premiers de P

Exemple:

Soit $f = \bar{a}.\bar{b}.c + a.b.c + b.\bar{c}$

Supposons que a soit la première variable à considérer : On calcule les monômes premiers de f_a, $f_{\bar{a}}$ et $f_a.f_{\bar{a}}$.

c.d \ b	0	1
00	0	1
01	0	1
11	0	1
10	0	1

fa

c.d \ b	0	1
00	0	1
01	0	1
11	1	0
10	1	0

$f_{\bar{a}}$

c.d \ b	0	1
00	0	1
01	0	1
11	0	0
10	0	0

$f_a.f_{\bar{a}}$

On obtient donc :

Premier (f_a) = {b}, Premier ($f_{\bar{a}}$) = {$b.\bar{c}$, $\bar{b}.c$}, Premier ($f_a.f_{\bar{a}}$) = {$b.\bar{c}$}.

D'après la relation (a), l'ensemble des monômes premiers de f qui ne contiennent ni le littéral a ni le littéral $\bar{a}$ est : Premier ($f_a.f_{\bar{a}}$) = {$b.\bar{c}$} (le consensus).

D'après la relation (b), on déduit l'ensemble des monômes premiers de f qui contiennent le littéral a : {a} $\otimes$ (Premier (f_a) - Premier ($f_a.f_{\bar{a}}$)) = {a.b}, et grâce à la relation (c) on obtient l'ensemble des monômes premiers de f qui contient le littéral $\bar{a}$: {$\bar{a}$} $\otimes$ (Premier($f_{\bar{a}}$) - Premier($f_a.f_{\bar{a}}$)) = {$\bar{a}.\bar{b}.c$}

L'ensemble des monômes premiers de f est donc sur cet exemple : {$b.\bar{c}$, a.b, $\bar{a}.\bar{b}.c$}

La représentation en MBDD associée aux monômes premiers est donnée dans la figure I.18 :

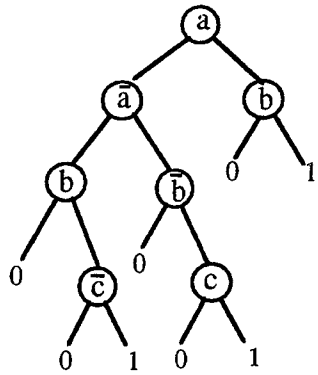
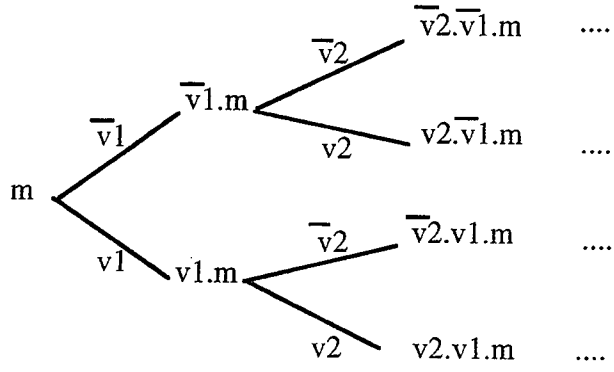


Figure I.18 : MBDD des monômes premiers

I.5.4. Calcul implicite des monômes canoniques

Soit f une fonction booléenne, M l'ensemble des monômes qui représentent f. À partir de chaque monôme m de M, on peut générer explicitement l'ensemble des monômes canoniques. Soient V_1, ..., V_p les variables du support de f qui

n'appartiennent pas à m; les monômes canoniques associés à m se construisent explicitement de la façon suivante:



La représentation implicite des monômes canoniques est construite globalement à partir du ROBDD de la fonction booléenne au lieu de traiter monôme par monôme dans le cas de la génération explicite.
La première étape consiste à transformer les noeuds ROBDD en noeuds MBDD (voir figure I.19). Au cours de cette transformation, une deuxième manipulation consiste à insérer le noeud du MBDD associé à une variable du support absente du ROBDD.

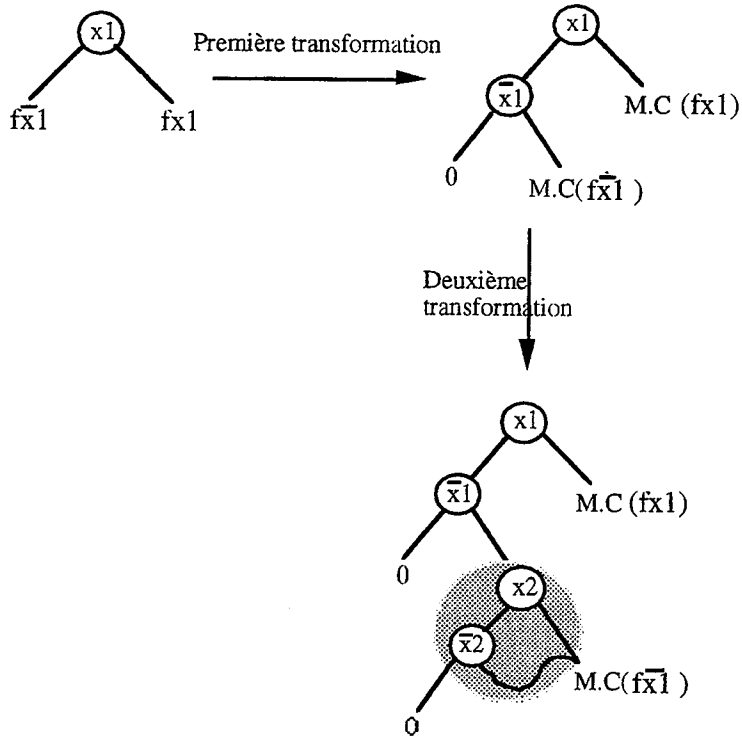


Figure I.19 : Génération des monômes canoniques à partir du ROBDD de f.

Le parcours du ROBDD se fait de la racine vers les feuilles, l'ordre du traitement des noeuds est celui du ROBDD.

Exemple:

Soit f = $x_1.x_2.x_3 + \bar{x}_1.\bar{x}_2$

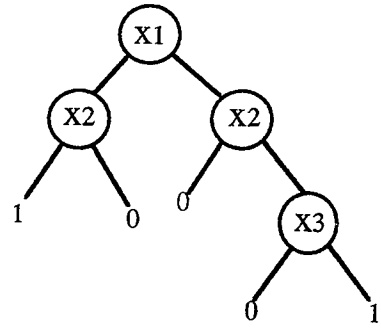


Figure .I.20 : Représentation de f en ROBDD

L'ordonnancement initial des variables - donné dans la figure I.20 - est supposé être $x_1 >> x_2 >> x_3$. Le MBDD associé aux monômes canoniques est donné dans la figure I.21.

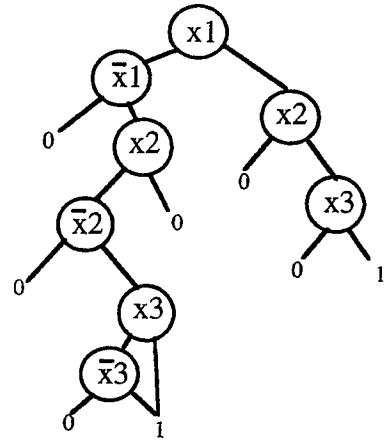


Figure I.21 : Représentation des monômes canoniques de f en MBDD

I.5.5. Résolution du problème de couverture

I.5.5.1 Réduction du problème de couverture

Une fois le problème de couverture posé, il peut être réduit par un ensemble de transformations qu'on exposera dans ce paragraphe. Pour mieux illustrer les réductions possibles nous raisonnons sur la matrice de couverture en enlevant itérativement des lignes et des colonnes.

Deux notions fondamentales sont utilisées pour la réduction du problème de couverture, l'***essentialité*** et la ***dominance*** .

Nous avons vu que les monômes premiers essentiels appartiennent forcément à toutes les bases irrédondantes, donc nécessairement à la solution minimale. Cette propriété se traduit sur la matrice de couverture en enlevant toutes les colonnes qui représentent les éléments essentiels et les lignes qui en sont couvertes. La solution minimale du problème est l'union des colonnes enlevées et la solution de la matrice de couverture réduite.

Exemple:

	y1	y2	y3	y4	y5
x1	1	1	1	1	
x2	1	1	1	1	
x3	1			1	
x4	1			1	1
x5	1			1	
x6					1

→

	y1	y2	y3	y4
x1	1	1	1	1
x2	1	1	1	1
x3	1			1
x5	1			1

Réduction par essentialité

La relation de dominance a été introduite pour réduire la taille du problème de couverture en enlevant les lignes et les colonnes de la matrice de couverture et produire une nouvelle matrice de couverture réduite équivalente à l'initiale. Soit <X, Y> la matrice de couverture, soit x, x' éléments de X. On dit que x' domine x si et seulement si tous les éléments de Y qui couvrent x' couvrent aussi x. Cela veut dire qu'une fois que x' est couverte, il en est de même pour x; donc la ligne x peut être enlevée de la matrice, c'est la réduction par dominance sur X. On dit que y' élément de Y domine y si et seulement si tous les éléments de X couverts par y le sont aussi par y' et le coût (y') ≤ coût (y). Cela veut dire que y' couvre au moins tous les éléments de X couverts par y sans qu'il soit plus coûteux, c'est la réduction par dominance sur Y.

Exemple:

	y1	y2	y3	y4	y5
x1	1	1	1	1	
x2	1	1	1	1	
x3	1			1	
x4	1			1	1
x5	1			1	
x6					1

→

	y1	y2	y3	y4	y5
x3	1			1	
x6					1

Réduction par dominance sur X

	y1	y2	y3	y4	y5
x1	1	1	1	1	
x2	1	1	1	1	
x3	1			1	
x4	1			1	1
x5	1			1	
x6					1

→

	y1	y5
x1	1	
x2	1	
x3	1	
x4	1	1
x5	1	
x6		1

Réduction par dominance sur Y

Lorsque les processus de réduction par essentialité et dominance sont appliqués itérativement jusqu'à ce que la matrice de couverture ne puisse plus être réduite, on atteint un point fixe appelé ***noyau cyclique*** "cyclic core". Si ce noyau cyclique est vide, les éléments essentiels trouvés lors des réductions constituent la solution minimale du problème. Sinon il faut y ajouter les éléments solutions du problème du noyau cyclique.

I.5.5.2 Formalisation des règles de réduction du problème

Soit <P, Q> le problème de couverture. On considère $\tau_P(Q)$ et $\tau_Q(P)$ les ensembles images des applications τ_P et τ_Q définies par :

$$\forall q \in Q \ \tau_P(q) = \inf_{\subseteq} \{ p \in P \ / \ q \subseteq p \}$$

$$\forall p \in P \ \tau_Q(p) = \sup_{\subseteq} \{ q \in Q \ / \ q \subseteq p \}$$

La première expression donne le plus petit élément de P qui couvre q.
La deuxième expression donne le plus grand élément de Q qui est couvert par p.

Théorème [Coudert94]

On considère initialement que P est l'ensemble des monômes premiers et Q l'ensemble des monômes canoniques. En appliquant les trois transformations ci-après itérativement, la réduction du problème <P, Q> converge vers le noyau cyclique; de plus, les réductions (1) et (2) sont équivalentes à la réduction par dominance et (3) à la réduction par essentialité.

$$<P, Q> \xrightarrow{1} <P, \max \tau_P(Q)>$$

$$<P, Q> \xrightarrow{2} <\max \tau_Q(P), Q>$$

$$<P, Q> \xrightarrow{3} <P-E, Q-E> \qquad \text{avec } E = P \cap Q$$

I.5.5.3 Interprétation de la recherche de noyau cyclique par la théorie des treillis

Un treillis est un ensemble ordonné (T, <) tel que toute partie de T admette une borne inférieure et une borne supérieure dans T.
En particulier, T admet un élément minimal égal à sa borne inférieure, et un élément maximal égal à sa borne supérieure.

Soit Q l'ensemble des monômes canoniques d'une fonction booléenne F, $\mathscr{P}(Q)$ l'ensemble des parties de Q. P l'ensemble des monômes premiers de la borne supérieure de F. ($\mathscr{P}(Q)$, $\subseteq$) est un treillis ayant comme élément minimal le monôme 0 et élément maximal le monôme 1. $P \subseteq \mathscr{P}(Q)$ et $Q \subseteq \mathscr{P}(Q)$.
P admet un élément minimal, et Q admet un élément maximal dans $\mathscr{P}(Q)$.

La réduction (1) consiste à remplacer l'ensemble Q (initialement l'ensemble des monômes canoniques) par l'ensemble maximal - au sens de l'inclusion - des plus petits éléments de P (initialement l'ensemble des monômes premiers) qui couvrent les éléments de Q.

La réduction (2) consiste à remplacer l'ensemble P par l'ensemble maximal - au sens de l'inclusion - des plus grands éléments de Q qui sont couverts par les éléments de P.

La réduction (3) extrait les éléments qui appartiennent à P et Q, qui sont les résultats de la réduction (1) et (2). Ces éléments s'appellent ***les éléments essentiels***.,

Lorsque le processus des trois réductions converge ($P \cap Q = \phi$), <P, Q> constitue alors le noyau cyclique.

I.5.5.4 Algorithme de recherche de noyau cyclique

Dans ce paragraphe, nous présentons l'algorithme des trois réductions citées plus haut qui calculent le noyau cyclique.
En entrée nous désignons par P le MBDD de l'ensemble des monômes premiers de la borne supérieure de la fonction booléenne F, par Q le MBDD de l'ensemble des monômes canoniques de la borne inférieure de F.

L'algorithme général est le suivant:

```
F = φ;
P1 = φ;
Q1 = φ;
Tant que ((P ≠ P1) ou (Q ≠ Q1)) faire
{
 P1 = P;
 Q1 = Q;
 Q = max τP(Q);
 E = P ∩ Q;
 F = F ∪ E; /* Mémorisation des éléments essentiels */
 Q = Q - E;
 P = P - E;
 P = max τQ(P);
} /* à la sortie de la boucle, < P, Q > constitue le noyau cyclique */
```

En faisant appel aux opérations élémentaires sur les MBDD décrites auparavant (ex: intersection, union, ...), nous calculons les applications max $\tau_P(Q)$ et max $\tau_Q(P)$. Pour ce faire, considérons les ensembles X et Y; nous avons besoin de calculer l'**ensemble des éléments de X qui ne sont contenus dans aucun élément de Y** :

$$\text{NotSubSet}(X, Y) = \{x \in X / \forall y \in Y, x \not\subset y\}$$

L'algorithme qui calcule cet ensemble est le suivant:

```
NotSubSet (X, Y)
{
 Si (Y = φ) retourner X;
 Si ((X = φ) ou (1 ∈ Y)) retourner φ;
 Si (X = {1}) retourner {1};
 Si (X.top < Y.top)
    retourner CréerNoeud (X.top, NotSubSet (XThen, Y),
                                 NotSubSet (XInv, Y),
                                 NotSubSet (XElse, Y));
 Si (X.top > Y.top)
    retourner NotSubSet(X, YElse);
 Si (X.top = Y.top)
    retourner CréerNoeud (X.top, NotSubSet (XThen, Union (YThen, YElse)),
                                 NotSubSet (XInv, Union (YInv, YElse)),
                                 NotSubSet (XElse, YElse));
}
```

Le calcul de $\max\tau_P(Q)$ s'effectue alors en évaluant la procédure MaxTauP (P, Q, 1).

```
MaxTauP (P, Q, k)
{
 Si ((P = φ) ou (Q = φ)) retourner φ ;
 Si (P = {1}) retourner {1};
 Si ((1 ∉ P) et (Q = {1})) retourner φ;
 K = Union (Q1k, Union (NotSubSet (Qx̄k, Px̄k), NotSubSet (Qxk, Pxk)));
 R = MaxTauP (P1k, K, k + 1);
 R0 = MaxTauP(Union(P1k, Px̄k), Qx̄k, k + 1);
 R1 = MaxTauP (Union(P1k, Pxk), Qxk, k + 1);
 retourner CréerNoeud (xk, NotSubSet(R1, R), NotSubSet(R0, R), R);
}
```

Dans cet algorithme, K est l'ensemble des monômes q de Q tels que $\tau_P(q)$ ne contient ni le littéral x_k ni le littéral $\bar{x}_k$, R est l'ensemble des éléments maximaux de $\tau_P(Q)$ qui ne contiennent ni le littéral x_k ni le littéral $\bar{x}_k$, R0 est $\max(\tau_P(Q)_{\bar{x}_k} \cup W)$ où W est un faux terme complémentaire qui est un sous-ensemble de R. Comme on doit garder uniquement les monômes maximaux de $\tau_P(Q)$, on doit enlever les monômes qui sont contenus dans un monôme de R pour obtenir l'ensemble $(\max(\tau_P(Q)))_{\bar{x}_k}$. Ce dernier est alors NotSubSet(R0, R). Le traitement pour R1 est similaire.

La seconde procédure de base dont on a besoin pour le calcul de $\max \tau_P(Q)$ et $\max \tau_Q(P)$est la procédure SupSet qui calcule l'**ensemble des monômes de X qui contiennent au moins un élément de Y**:

$$\text{SupSet}(X, Y) = \{x \in X / \exists y \in Y, y \subseteq x\}$$

L'algorithme qui calcule cet ensemble est le suivant:

```
SupSet (X, Y)
{
        Si ((X = ø) ou (Y = ø)) retourner ø;
        Si (X = {1}) retourner {1};
        Si ((1 ∉ X) et (Y = {1})) retourner ø;
        Si (X.top < Y.top) retourner SupSet (XElse, Y);
        Si (X.top > Y.top) retourner SupSet (X, Union (YElse,
                                                Union (YThen, YInv)));
        Si (X.top = Y.top)
           retourner  CréerNoeud ( X.top,
                                   SupSet (XThen, YThen),
                                   SupSet (XInv, YInv),
                                   SupSet (XElse,
                                      Union (YElse, Union ( YThen, YInv))));
}
```

Le calcul de $(\max\tau_Q(P))$ s'effectue en évaluant MaxTauQ (Q, P, 1). Dans cet algorithme, K est l'ensemble des monômes de P_{1k} qui contiennent un monôme de Q_{1k}, ou qui contiennent un monôme de Q_{xk} et un monôme de $Q_{\bar{x}k}$. Cela signifie que K est l'ensemble des monômes p de P tels que $\tau_Q(P)$ ne contient ni le littéral x_k ni le littéral $\bar{x}_k$. L'ensemble R est alors l'ensemble des monômes de $(\max\tau_Q(P))$qui ne contiennent ni le littéral x_k ni le littéral $\bar{x}_k$. Il reste à calculer l'ensemble des éléments maximaux de $(\max\tau_Q(P))$ qui contiennent soit le littéral x_k soit le littéral $\bar{x}_k$. L'ensemble R0 est l'ensemble des monômes de $(\max\tau_Q(P))$ qui contiennent soit le littéral x_k, soit le littéral $\bar{x}_k$, et qui sont maximaux dans l'ensemble $(\tau_Q(P))_{\bar{x}k}$. En plus de ces monômes, R0 contient d'autres monômes contenant le littéral $\bar{x}_k$, qui appartiennent à $\tau_Q(P)$, mais qui ne sont pas maximaux dans $(\tau_Q(P))_{\bar{x}k}$ par rapport à des monômes de max $\tau_Q(P)$. Comme on ne doit garder que des éléments maximaux de $\tau_Q(P)$, il s'ensuit que NotSubSet(R0, R) est l'ensemble des monômes de max $\tau_Q(P)$ qui contiennent le littéral $\bar{x}_k$.

```
MaxTauQ (Q, P, k)
{
 Si ((Q = φ) ou (P = φ)) retourner φ;
 Si ((1 ∈ Q) et (1 ∈ P)) retourner {1};
 K = Union (SupSet(P1k, Q1k), Intsec (SupSet(P1k, Qx̄k), SupSet(P1k, Qxk)));
 R = MaxTauQ (Union (Q1k, Union(Qxk, Qx̄k)), K, k + 1);
 R0 = MaxTauQ (Qx̄k, Union (P1k, Px̄k), k + 1);
 R1 = MaxTauQ (Qxk, Union (P1k, Pxk), k + 1);
 retourner CréerNoeud (xk, NotSubSet(R1, R), NotSubSet (R0, R), R);
}
```

Pour chacune de ces quatre procédures on utilise un cache-mémoire pour stocker les derniers calculs qu'on vient de faire, et pour éviter ainsi de les refaire.

I.5.5.4 Recherche de l'optimum par séparation et évaluation

Cette méthode classique, appelée aussi "branch and bound" ([Gondran85], [Finke91]) est une méthode d'exploration intelligente de l'espace de solutions. Elle ne s'autorise à parcourir une arborescence qu'à la condition de vérifier localement une certaine condition. Cette condition est dynamique et peut évoluer lors de l'exploration de l'arbre.

Nous supposons que le noyau cyclique n'est pas vide. L'approche consiste à choisir un monôme premier p, et générer deux sous-problèmes, un qui suppose que p est dans la solution minimale, l'autre qui l'élimine. Ces deux sous-problèmes sont réduits par l'algorithme décrit plus haut. Leur noyau cyclique est alors calculé.

Plusieurs heuristiques ([McCluskey59], [Brayton84], [Rudell89]) ont concerné le problème du choix des monômes premiers, la fonction objectif étant de minimiser tout d'abord le nombre de monômes de la base irrédondante et ensuite choisir parmi celles qui ont le même nombre de monômes une base qui a le nombre minimal de littéraux . L'idée naturelle consiste à choisir un monôme premier qui couvre le plus grand nombre de monômes canoniques. Malheureusement, l'expérience montre que ce n'est pas la meilleure heuristique. Une meilleure approche est proposée dans [Rudell89] qui choisit le monôme premier qui minimise la fonction suivante:

$$\sum_{q \text{ couvert par } p} \frac{1}{|\{p \text{ premier / } p \text{ couvre } q\}|-1}$$

Cette fonction coût s'inspire de la remarque sur les monômes premiers : Les plus difficiles à couvrir sont les moins couverts. Elle favorise les monômes premiers qui couvrent les monômes canoniques 'difficiles'.

L'exploration d'une branche de l'arbre des résultats s'arrête quand la solution en cours dépasse le meilleur résultat déjà trouvé. Il est donc nécessaire d'avoir une borne inférieure précise pour éviter le plus tôt possible des branches n'amenant pas à la solution optimale.
Soit Q' un sous-ensemble de Q, soit q_1 et q_2 deux éléments de Q', tel qu'un monôme qui couvre q_1 ne couvre pas q_2, Q' est appelé ***sous-ensemble indépendant***. La borne inférieure s'écrit:

$$\sum_{q \in Q'} \min_{\substack{p \\ p \text{ couvre } q}} \text{Coût}(p)$$

Maximiser cette borne inférieure est un problème NP-complet. Plusieurs heuristiques ont permis de trouver une bonne borne, sauf dans le cas ou le nombre de monômes premiers est très grand par rapport au nombre de monômes canoniques.

I.5.5.5 Organigramme général

En entrée, la procédure de la minimisation symbolique locale reçoit un ensemble d'expressions booléennes sous forme polynomiale, le but est de rendre cet ensemble sous une forme polynomiale optimale (en terme du nombre de monômes). La minimisation optimise une expression à la fois. Tout d'abord on cherche un ordre des variables local associé à l'expression à optimiser. Pour ce faire, on utilise l'heuristique de l'assignement dynamique pour construire un ROBDD initial, puis on améliore sa taille par la méthode de la fenêtre glissante. À partir du ROBDD optimisé on construit d'une part le MBDD qui représente les monômes canoniques, et d'autre part celui des monômes premiers. À ce stade le problème de couverture est bien formulé, nous procédons ensuite à l'application des règles de réduction d'une façon itérative jusqu'à la convergence au noyau cyclique <P, Q>. Au cours de l'application des règles de réduction, les éléments essentiels sont générés et stockés. Si la taille du problème du noyau cyclique est raisonnable (moins de 100 monômes de P) la résolution se fait par la méthode de séparation et évaluation pour obtenir la solution exacte. Dans le cas contraire, nous faisons appel à une heuristique fondée sur l'expansion des monômes de Q. La solution obtenue est exprimée par un MBDD, qui est à son tour transformé en expression polynomiale.

L'organigramme général de la minimisation symbolique se présente donc de la façon suivante:

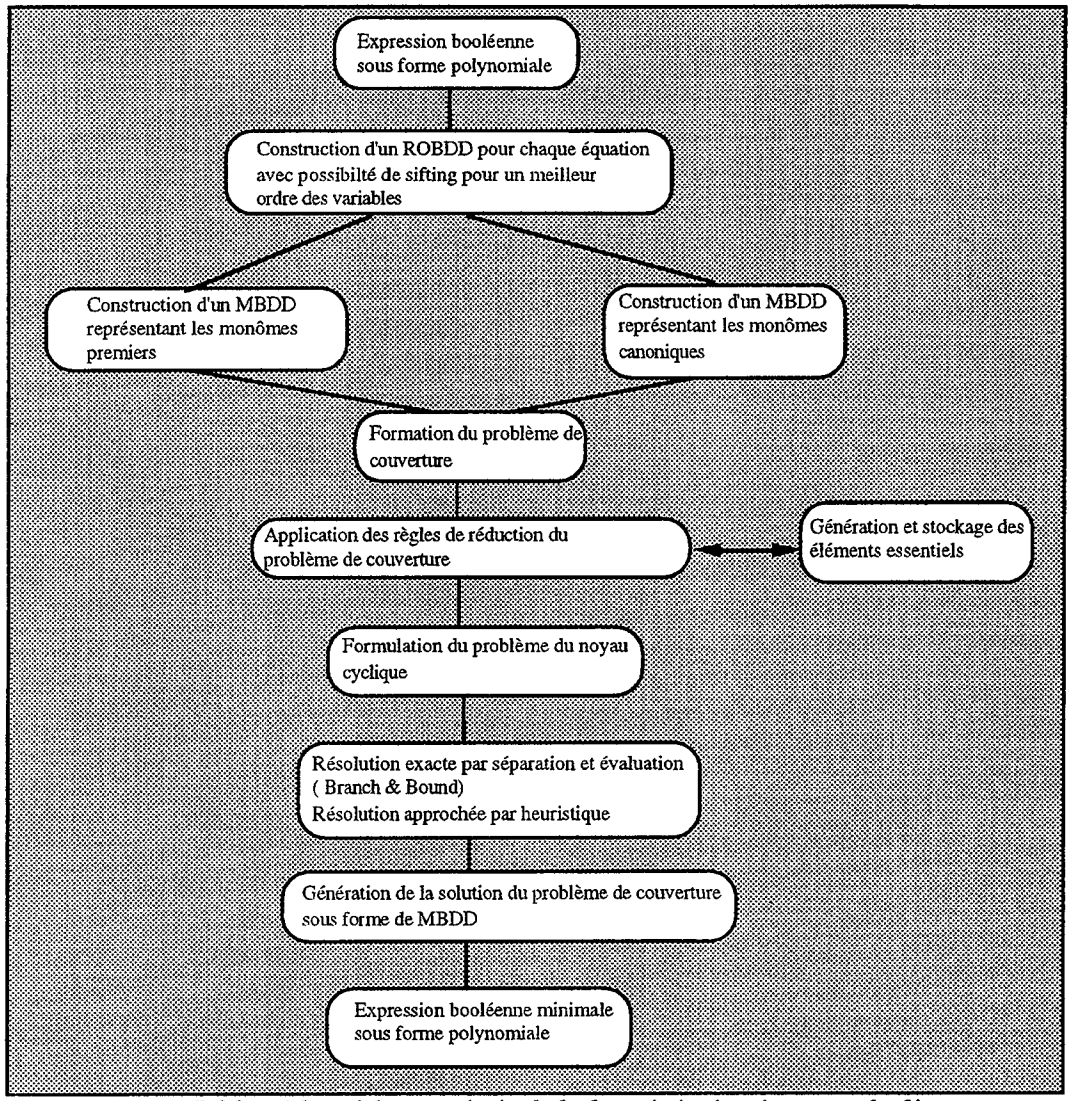


Figure I.22 : Algorithme général de la minimisation symbolique

I.6. Extension aux fonctions booléennes générales

L'extension de la minimisation à un ensemble de fonctions consiste à exprimer l'ensemble des fonctions par un ensemble minimal de monômes tout en tenant compte du partage des monômes entre les différentes expressions.
Afin de mieux définir le problème de la minimisation générale, commençons par définir les notions principales.

I.6.1. Définitions

• Monôme général

Soit f = [f1, f2, ..., fn] un ensemble de fonctions booléennes. Un monôme général est défini par un couple (m, I) où m est un monôme et I un vecteur booléen (e1, e2, ..., en), appelé vecteur d'inclusion, tel que si $e_i = 1$ alors le monôme m appartient à la fonction f_i.

Exemple:

soit [f1, f2, f3] un ensemble de fonctions booléennes défini comme suit:

$f_1 = a.b + c.d$

$f_2 = a.b + \bar{c}.e$

$f_3 = a + \bar{c}.e + c.d$

En faisant appel aux vecteurs d'inclusion, on peut exprimer f de la façon suivante :

$f = [f_1, f_2, f_3] = a.b\ [110] + c.d\ [101] + \bar{c}.e\ [011] + a\ [001]$

• Relation d'ordre sur les monômes général

Une relation d'ordre sur les monômes généraux est définie comme suit:

$(m_1, I_1) \subseteq (m_2, I_2) \neq ((m_1 \subseteq m_2)$ et $(I_1 \subseteq I_2))$

Soit f = [f1, f2, ..., fn], et M = (m, I) un monôme général avec I = (e1, ..., en). On dit que $M \subseteq f$ si pour tout i tel que $e_i = 1$ on a $m \subseteq f_i$.

• Monôme premier général

M est monôme premier dit général si et seulement si:

- $M \subseteq f$
- il n' existe pas de monôme général $M_1 \subseteq f$ tel que $M \subseteq M_1$ et $M \neq M_1$.

• Définition de monôme canonique général

Un monôme général (m,I) est dit canonique si et seulement si I ne contient qu'un seul 1 à la jième position, et m est un monôme canonique de la jième fonction.

I.6.1. Algorithme de la minimisation générale

La minimisation globale (ou générale) d'une fonction générale F = [f1, f2, ..., fn] consiste à résoudre le problème de couverture <P, Q> où P est l'ensemble des monômes premiers généraux et Q est l'ensemble des monômes canoniques généraux de F.

L'algorithme implanté dans le cadre de cette thèse n'utilise pas les vecteurs d'inclusion. Il introduit un vecteur de variable [y1, y2, ..., yn] représentant l'ensemble de fonctions booléennes [f1, f2, ..., fn].
Par exemple le monôme général $x_1.\bar{x}_2$ [101] s'ecrira $x_1.\bar{x}_2.y_1.\bar{y}_2.y_3$.
Soit la fonction générale F = [f1, f2, ..., fn] de n composants et dépendant de m variables. La minimisation globale de F se ramène à la minimisation de la fonction booléenne ff définie de $\{0, 1\}^{n+m}$ dans {0, 1, *} par :

$$ff^{1*}(x,y) = (\bar{y}_1 + f_1^{1*}(x)).(\bar{y}_2 + f_2^{1*}(x))...(\bar{y}_n + f_n^{1*}(x))$$

$$ff^{1}(x,y) = y_1.\bar{y}_2.\bar{y}_3...\bar{y}_n.f_1^{1}(x) + \bar{y}_1.y_2.\bar{y}_3...\bar{y}_n.f_2^{1}(x) +....+ \bar{y}_1.\bar{y}_2.\bar{y}_3...yn.f_1^{1}(x)$$

Le calcul de Prime (ff^{1*}) nous donne l'ensemble des monômes premiers généraux de F, le calcul des monômes canoniques de ff donne l'ensemble des monômes canoniques généraux.

Exemple:
Soit la fonction booléenne générale suivante:
F(a, b, c) = (f1, f2, f3) où f1, f2 et f3 sont définies par les expressions suivantes:

$f_1 = a.c + a.\bar{b} + \bar{b}.c$
$f_2 = \bar{a}.b + \bar{a}.c + b.c$
$f_3 = a.b + b.\bar{c}$

La liste des monômes généraux est donc:
a.c [1, 0, 0]; $a.\bar{b}$ [1, 0, 0]; $\bar{b}$ .c [1, 0, 0]; $\bar{a}$.b [0, 1, 0]; $\bar{a}$.c [0, 1, 0]; b.c [0, 1, 0]; a.b[0, 0, 1]; $b.\bar{c}$ [0, 0, 1]

$ff^{1*} = (\bar{y}_1 + f_1(x)).(\bar{y}_2 + f_2(x)).(\bar{y}_3 + f_3(x))$

$ff = y_1.\bar{y}_2.\bar{y}_3.f1(x) + \bar{y}_1.y_2.\bar{y}_3.f_2(x) + \bar{y}_1.\bar{y}_2.y_3.f_3(x)$

Les monômes premiers généraux de F sont :

a.c [1, 0, 0]; a.$\bar{b}$ [1, 0, 0]; $\bar{b}$.c [1, 0, 0]; $\bar{a}$.b [0, 1, 0]; $\bar{a}$.c [0, 1, 0]; b.c [0, 1, 0]; a.b [0, 0, 1]; b.$\bar{c}$ [0, 0, 1]; $\bar{a}$.$\bar{b}$.c [1, 1, 0]; $\bar{a}$.b.$\bar{c}$ [0, 1, 1]; a.b.c[1, 1, 1].

La base irrédondante minimale est :

{a.$\bar{b}$[1, 0, 0]; $\bar{a}$.$\bar{b}$.c[1, 1, 0]; $\bar{a}$.b[0, 1, 0]; a.b.c[1, 1, 1]; b.$\bar{c}$[0, 0, 1]}

I.7. Calcul de De Morgan

La formule de De Morgan consiste à calculer le complément d'une fonction booléenne F sous sa forme polynomiale $\sum_{1 \le i \le n} mi$.

$$\bar{F} = \overline{\sum_{1 \le i \le n} mi} = \prod_{1 \le i \le n} \overline{mi}$$

Le calcul du complément d'une façon explicite pose des problèmes de complexité car il est nécessaire d'exprimer tous les monômes de la fonction complémentaire avant de les minimiser. En effet, si on utilise la formule de De Morgan pour calculer le complément d'une somme de n monômes ayant au moins m littéraux chacun, la forme explicite avant minimisation est de m^n monômes. Pour éviter ce problème d'explosion en complexité, la représentation implicite en ROBDD typé (voir figure I.23) permet le calcul du complément en un temps constant [Madre88]. On commence tout d'abord par construire le ROBDD en mettant l'inversion à sa racine, et en la propageant jusqu'aux feuilles, comme il est indiqué dans la figure I.24. Par exemple si le ROBDD de la fonction de la figure I.23.

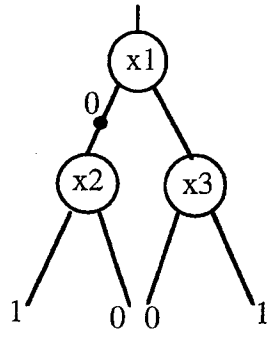


Figure I.23 : Représentation en ROBDD typé

On effectue la transformation suivante:

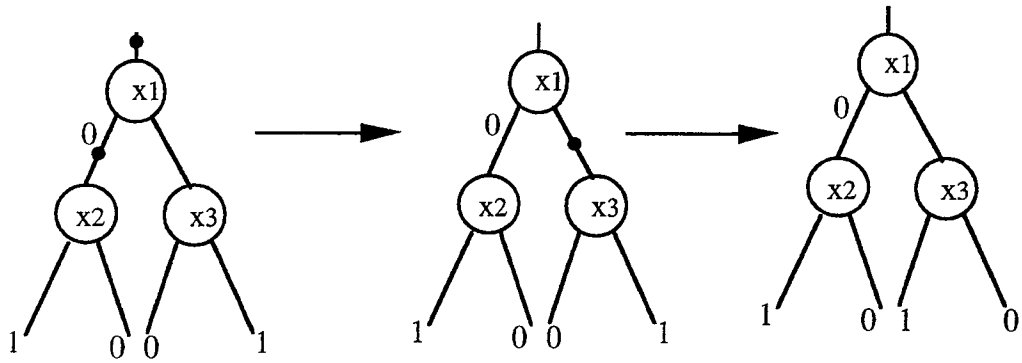


Figure I.24 : Propagation de l'inverseur aux feuilles du ROBDD

Après que l'inversion soit propagée, on transforme le ROBDD en MBDD qui représente donc la couverture à 0 de la fonction booléenne F. Ensuite, on appelle la procédure de minimisation symbolique. Le résultat est un MBDD qui représente implicitement une base minimale du complément de la fonction.. En conséquence, on peut choisir la forme de taille en nombre de monômes est la plus petite, cette étape s'appelle **la sélection de la phase** (positive ou négative).

I.8. Expertise des résultats

Dans cette section nous avons analysé les résultats de deux familles d'exemples:
- Les gros contrôleurs, qui sont de la logique aléatoire pure.
- Une sélection d'exemples MCNC [Yang91], qui comprend des exemples plus variés caractérisés par une structure de logique régulière, et de logique aléatoire.

Les tests ont été effectués sur une station de travail SUN SPARC 20 avec 64 Mo de mémoire vive .

Nous voulons montrer à travers ces tests :
- Les performances en terme de temps de calcul et d'occupation mémoire du minimiseur symbolique implanté face à une méthode heuristique Espresso-like.
- Le gain en terme de monômes et de littéraux que peut apporter la minimisation symbolique avec le calcul de De Morgan. Le calcul de De Morgan ne peut être envisagé dans le cas d'une minimisation d'expressions sous forme polynomiale explicite de part sa complexité explosive.

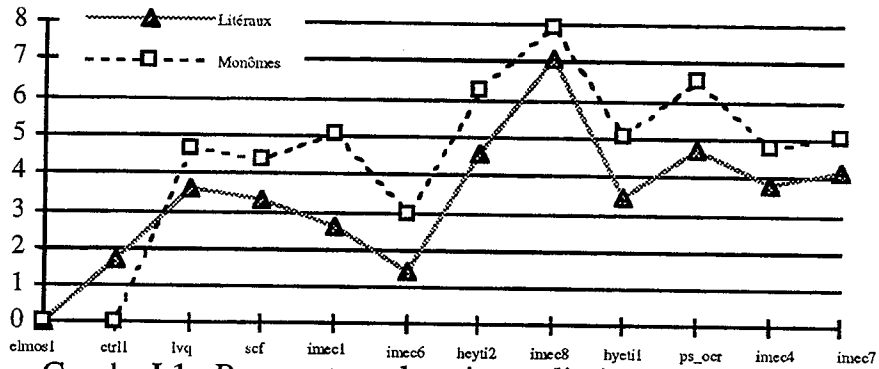
I.8.1. Comparaison entre le minimiseur symbolique et Espresso-like

Name	NbLit_old	NbLit_new	NbPT_old	NbPT_new	Cpu_old	Cpu_new	MaxMem_old	MaxMem_new
elmos1	76	76	27	27	0,11	0,06	104120	122668
ctrl1	118	116	44	44	0,3	0,15	201676	170808
lvq	1192	1149	260	248	4,28	1,74	1306032	672252
scf	1849	1788	342	327	11,48	4,06	3425524	1505404
imec1	3254	3169	610	579	16,86	6,06	4132716	1897812
imec6	2407	2373	500	485	16,52	5	4345372	1938564
heyti2	5806	5539	942	883	29,96	24,53	5120144	2648792
imec8	6287	5842	1036	954	27,09	12,06	5432336	2523032
hyeti1	7762	7493	1244	1181	31,21	41,55	7288948	4541532
ps_ocr	6195	5901	1119	1046	39,92	20,16	8409904	3919344
imec4	9897	9520	1549	1475	38,54	13,18	8677960	4028452
imec7	16791	16091	2746	2606	77,18	29,34	14728296	7183168

Tableau I.1 : Résultats comparatifs entre le minimiseur symbolique et Espresso-like (Gros contrôleurs)

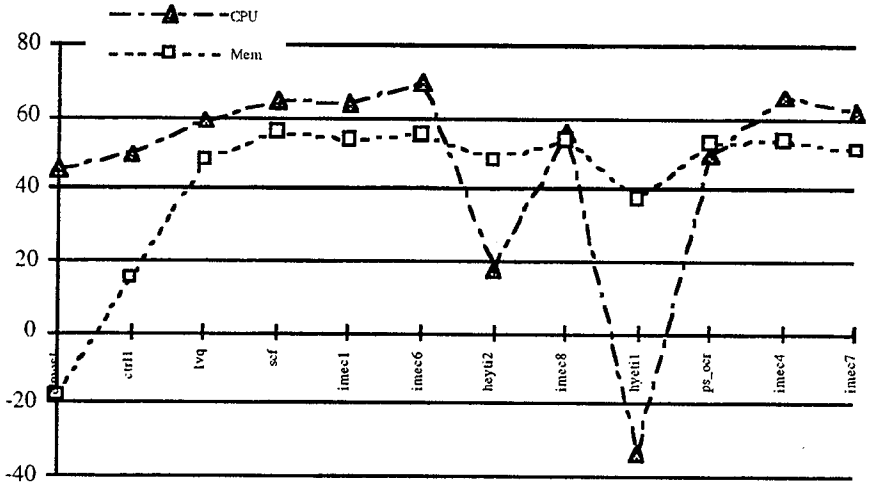
Légende:

NbLit_old : Nombre de littéraux obtenu par le minimiseur Espresso-like
NbLit_new : Nombre de littéraux obtenu par le minimiseur symbolique
NbPT_old: Nombre de monômes obtenu par le minimiseur Espresso-like
NbPT_new: Nombre de monômes obtenu par le minimiseur symbolique
Cpu_old: Temps de calcul en secondes du minimiseur Espresso-like
Cpu_new: Temps de calcul en secondes du minimiseur symbolique
MaxMem_old: Espace mémoire nécessaire pour le minimiseur Espresso-like
MaxMem_new: Espace mémoire nécessaire pour le minimiseur symbolique



Graphe I.1 : Pourcentage de gains en littéraux et monômes du minimiseur symbolique par rapport à Espresso-like

Gain moyen en littéraux = 3,7%
Gain moyen en monômes = 4,4%



Graphe I.2 : Pourcentage de gains en temps de calcul et espace mémoire du minimiseur symbolique par rapport à Espresso-like

Gain moyen en CPU = 55%
Gain moyen en espace mémoire = 42%

Dans cette première série de test, les graphes I.1 et I.2 montrent que les résultats de la solution optimale (en terme de nombre de monômes) de la minimisation symbolique (entre 0 et 8% de mieux que l'heuristique Espresso-like) est atteinte avec un gain dans la plupart des cas entre 40 et 70% en temps de calcul et en espace mémoire par rapport à l'heuristique Espresso-like. On note, que le temps de calcul de "hyeti1" est plus long, ce cas est expliqué par le fait que ce temps de calcul comprend la construction et l'amélioration de la taille du ROBDD par échanges de variables (méthode des fenêtres glissantes) en plus des procédures de minimisation. Le temps de calcul de la construction et l'amélioration du ROBDD prend 25 secondes (65 % du temps de calcul global), dont la quasi-totalité est passée dans l'amélioration. Il faut noter également le cas de "elmos1" où l'occupation mémoire est de 20% de plus que la méthode Espresso-like. En décortiquant cet exemple on a remarqué que la plus grande sortie n'a que sept monômes, sa représentation explicite occupe moins de mémoire que la taille mémoire initiale des cache-mémoires utilisés pour la minimisation symbolique. Le remède à ce problème est de faire une allocation dynamique des cache mémoires dépendante de la taille du problème à traiter au lieu d'une allocation statique.

Name	NbLit_old	NbLit_new	NbPT_old	NbPT_new	Cpu_old	Cpu_new	MaxMem_old	MaxMem_new
ex4	1649	1649	279	279	4,14	4,85	959568	748464
ibm	882	882	173	173	0,67	1,22	247792	270064
jbp	1181	1181	189	189	0,98	0,76	356768	321728
misg	180	180	75	75	0,21	0,14	156016	168924
mish	164	164	91	91	0,16	0,13	194888	202904
misj	77	77	48	48	0,07	0,06	102500	132212
pdc	1950	730	306	137	343,37	102,16	71929872	26000980
shift	399	399	105	105	0,34	0,23	162648	162088
ti	4140	4119	514	512	8,14	5,05	1615160	952372
ts10	896	896	128	128	0,57	0,45	224272	194468
x7dn	4734	4062	622	538	3,9	26,39	900092	793116
rckl	1143	1143	97	97	1,09	0,55	342384	287720
m181	166	166	53	53	1,36	0,22	448224	303924
al2	441	441	89	89	0,73	0,20	206284	184740
alcom	184	184	45	45	0,16	0,09	134916	145360
alu2	506	506	87	87	0,36	0,16	151848	150572
alu3	284	284	68	68	0,21	0,12	113412	131740
b10	1316	1300	173	170	4,11	2,12	1181692	723472
b12	166	166	53	53	1,41	0,22	448796	303816
b2	5432	5407	702	698	7,33	5,19	1801312	1150592
b4	734	734	96	96	0,79	1,04	311136	284312
b9	754	754	119	119	0,55	0,37	196640	228472
chkn	1654	1601	145	140	1,31	5,61	364624	619900
ex7	754	754	119	119	0,59	0,35	196640	228472
exp	725	673	117	108	3,92	1,05	1285968	581668
exps	3261	3112	529	504	12,56	2,70	3016248	1371292
in0	1524	1496	195	191	4,02	1,89	984536	651012

Tableau I.2.a : Résultats comparatifs entre le minimiseur symbolique et Espresso-like (Exemple MCNC)

Légende:

NbLit_old : Nombre de littéraux obtenu par le minimiseur Espresso-like
NbLit_new : Nombre de littéraux obtenu par le minimiseur symbolique
NbPT_old: Nombre de monômes obtenu par le minimiseur Espresso-like
NbPT_new: Nombre de monômes obtenu par le minimiseur symbolique
Cpu_old: Temps de calcul en secondes du minimiseur Espresso-like
Cpu_new: Temps de calcul en secondes du minimiseur symbolique
MaxMem_old: Espace mémoire nécessaire pour le minimiseur Espresso-like
MaxMem_new: Espace mémoire nécessaire pour le minimiseur symbolique

Name	NbLit_old	NbLit_new	NbPT_old	NbPT_new	Cpu_old	Cpu_new	MaxMem_old	MaxMem_new
in1	5432	5407	702	698	8,13	5,18	1801312	1150592
in2	2032	1975	237	230	2,74	1,40	592756	443936
in3	1324	1325	214	214	1,7	1,86	494144	402440
in5	1523	1524	175	175	1,46	1,25	401852	339596
in6	744	744	97	97	0,6	0,60	240240	215272
in7	552	552	79	79	0,67	0,55	224836	205576
m2	581	547	110	104	5,5	0,64	1302632	623960
m3	727	698	136	131	9,25	0,83	1648588	746700
m4	1089	999	232	211	13,37	1,77	2971140	1248600
mark1	117	115	72	33	10,03	3,16	2265604	905264
max128	842	804	201	191	8,5	1,03	2033264	891180
max512	1026	935	181	164	11,14	1,79	2478992	1166892
pope	1046	961	316	292	7,35	1,04	1801360	777944
spla	4544	4563	458	458	266,27	45,65	38855368	15078260
t2	349	349	68	68	3,15	0,71	1074544	503752
x6dn	1296	1277	175	172	1,87	2,38	510724	517852
fout	353	309	95	85	1,53	0,40	543484	294112
test1	1225	1024	218	184	7,59	2,27	2295304	1036556
test4	1936	1355	498	355	36,07	16,98	9550924	3522916
addm4	1332	1307	216	212	8,64	1,42	2005480	1014764
dist	900	843	161	150	3,19	0,65	840616	519736
f51m	321	321	76	76	6,06	0,61	1434340	682632
root	372	351	75	71	3,24	0,48	871544	498836
co14	196	196	14	14	0,19	0,05	101324	145588
sqr6	221	221	58	58	1,04	0,15	317908	216156
z4	252	252	59	59	1,15	0,16	354440	265636
m1	148	148	39	39	0,89	0,13	309792	205632
sqn	210	201	45	43	0,65	0,15	229020	214812
t4	102	76	28	23	10,2	2,71	3187136	1328996

Tableau I.2.b : Résultats comparatifs entre le minimiseur symbolique et Espresso-like (Exemple MCNC)

Légende:

NbLit_old : Nombre de littéraux obtenu par le minimiseur Espresso-like
NbLit_new : Nombre de littéraux obtenu par le minimiseur symbolique
NbPT_old: Nombre de monômes obtenu par le minimiseur Espresso-like
NbPT_new: Nombre de monômes obtenu par le minimiseur symbolique
Cpu_old: Temps de calcul en secondes du minimiseur Espresso-like
Cpu_new: Temps de calcul en secondes du minimiseur symbolique
MaxMem_old: Espace mémoire nécessaire pour le minimiseur Espresso-like
MaxMem_new: Espace mémoire nécessaire pour le minimiseur symbolique

Le tableau I.2.a et le tableau I.2.b indiquent que le cas de l'exemple de "hyeti1" revient dans certains exemples MCNC: "x7dn"..(en tout 7 cas sur 55). On revoit également un cas similaire à celui de l'exemple de "elmos1" pour "misg" "mish" et "misj" (en tout 9 cas sur 55).

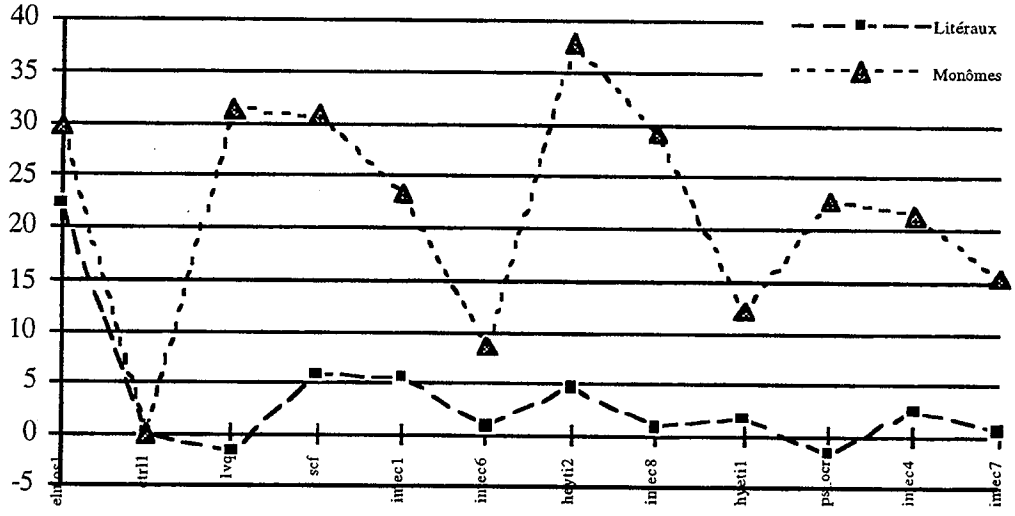
I.8.2. Amélioration apportée par le choix de la polarité

Name	NbLit_new	NbLit_inv	NbPT_new	NbPT_inv	Cpu_new	Cpu_inv	MaxMem_new	MaxMem_inv
elmos1	76	59	27	19	0,06	0,07	122668	125384
ctrl1	116	116	44	44	0,15	0,17	170808	176588
lvq	1149	1166	248	170	1,74	2,45	672252	691984
scf	1788	1684	327	226	4,06	5,02	1505404	1548660
imec1	3169	2992	579	443	6,06	7,83	1897812	1912044
imec6	2373	2354	485	444	5	7,62	1938564	1958024
heyti2	5539	5283	883	547	24,53	115,52	2648792	3176876
imec8	5842	5790	954	677	12,06	21,69	2523032	2535692
hyeti1	7493	7359	1181	1036	102,55	185,73	4541532	4545796
ps_ocr	5901	5986	1046	806	20,16	43,18	3919344	4172096
imec4	9520	9285	1475	1159	13,18	54,03	4028452	4017820
imec7	16091	16007	2606	2205	29,34	90,78	7183168	7190676

Tableau I.3 : Résultats comparatifs avec et sans choix de polarité (Gros contrôleurs)

Légende:

NbLit_inv : Nombre de littéraux obtenu avec choix de la polarité
NbLit_new : Nombre de littéraux obtenu avec polarité positive (sans choix)
NbPT_inv: Nombre de monômes obtenu avec choix de polarité
NbPT_new: Nombre de monômes obtenu avec polarité positive
Cpu_inv: Temps de calcul en secondes avec choix de polarité
Cpu_new: Temps de calcul en secondes avec polarité positive
MaxMem_inv: Espace mémoire nécessaire avec choix de polarité
MaxMem_new: Espace mémoire nécessaire avec polarité positive



Graphe I.3 : Gain en % du choix de polarité en littéraux et monômes

Gain moyen en littéraux = 3,5%
Gain moyen en monômes = 22%

Le calcul de De Morgan et le choix de polarité apportent un gain important, notamment en terme de nombre de monômes, il atteint plus de 35% dans le cas de l'exemple "hyeti2" par rapport à la forme directe (polarité positive). Ce gain reste assez irrégulier, difficilement prédictif, ce qui nous oblige à lancer effectivement la minimisation symbolique sur la forme directe et complémentée de l'expression booléenne. Toutefois, le résultat (en terme de nombre de monômes) de la minimisation de la forme directe peut être utilisé comme borne inférieure pour la minimisation de la forme complémentée, notamment à l'étape de la recherche des monômes essentiels ou la résolution du noyau cyclique.
Les résultats en terme de l'occupation mémoire montrent que le coût supplémentaire pour la minimisation de la forme complémentée est très bas. Ceci provient du fait qu'on utilise le même cache-mémoire que celui de la minimisation de la forme directe.
Les résultats en terme de temps de calcul restent très irréguliers. Ils varient de 5 à 500 % environ.

Name	NbLit_new	NbLit_inv	NbPT_new	NbPT_inv	Cpu_new	Cpu_inv	MaxMem_new	MaxMem_inv
ex4	1649	1504	279	59	4,85	6,66	748464	765148
ibm	882	882	173	173	1,22	11,13	270064	328384
jbp	1181	1181	189	189	0,76	1,49	321728	354176
misg	180	94	75	29	0,14	0,15	168924	174808
mish	164	164	91	56	0,13	0,13	202904	202440
misj	77	60	48	12	0,06	0,07	132212	134648
pdc	730	730	137	137	102,16	109,81	26000980	26020356
shift	399	399	105	105	0,23	0,4	162088	169304
ti	4119	4074	512	486	5,05	48,77	952372	1282256
ts10	896	896	128	128	0,45	2,78	194468	241336
x7dn	4062	4062	538	538	26,39	404,77	793116	1433248
rckl	1143	1143	97	97	0,55	0,57	287720	290620
m181	166	106	53	12	0,22	0,23	303924	302260
al2	441	420	89	84	0,20	0,25	184740	186128
alcom	184	183	45	42	0,09	0,11	145360	151192
alu2	506	284	87	35	0,16	0,38	150572	196312
alu3	284	212	68	36	0,12	0,17	131740	149376
b10	1300	1257	170	143	2,12	4,24	723472	738740
b12	166	103	53	12	0,22	0,23	303816	302344
b2	5407	5407	698	698	5,19	83,77	1150592	1990116
b4	734	734	96	96	1,04	2,44	284312	292932
b9	754	754	119	119	0,37	0,64	228472	254216
chkn	1601	1638	140	116	5,61	77,1	619900	1033260
ex7	754	754	119	119	0,35	0,67	228472	254216
exp	673	673	108	108	1,05	8,35	581668	623232
exps	3112	3120	504	448	2,70	7,46	1371292	1391740

Tableau I.4.a : Résultats comparatifs avec et sans choix de polarité (MCNC)

Légende:

NbLit_inv : Nombre de littéraux obtenu avec choix de la polarité
NbLit_new : Nombre de littéraux obtenu avec polarité positive (sans choix)
NbPT_inv: Nombre de monômes obtenu avec choix de polarité
NbPT_new: Nombre de monômes obtenu avec polarité positive
Cpu_inv: Temps de calcul en secondes avec choix de polarité
Cpu_new: Temps de calcul en secondes avec polarité positive
MaxMem_inv: Espace mémoire nécessaire avec choix de polarité
MaxMem_new: Espace mémoire nécessaire avec polarité positive

Name	NbLit_new	NbLit_inv	NbPT_new	NbPT_inv	Cpu_new	Cpu_inv	MaxMem_new	MaxMem_inv
in0	1496	1453	191	164	1,89	4,47	651012	740516
in1	5407	5407	698	698	5,18	76,91	1150592	1990116
in2	1975	1772	230	155	1,40	3,07	443936	573604
in3	1325	1310	214	198	1,86	12,07	402440	645708
in5	1524	1524	175	175	1,25	35,12	339596	981696
in6	744	744	97	97	0,60	1,49	215272	260244
in7	552	563	79	57	0,55	1,46	205576	259660
m2	547	536	104	98	0,64	0,74	623960	634632
m3	698	679	131	104	0,83	0,89	746700	755824
m4	999	1010	211	63	1,77	1,84	1248600	1257708
mark1	115	136	33	30	3,16	3,17	905264	908632
max128	804	761	191	122	1,03	1,18	891180	904960
max512	935	905	164	116	1,79	2,08	1166892	1219696
pope	961	1010	292	114	1,04	1,21	777944	778860
spla	4563	4254	458	406	45,65	46,17	15078260	15135800
t2	349	349	68	68	0,71	0,76	503752	513948
x6dn	1277	1277	172	172	2,38	151,25	517852	2216708
fout	309	305	85	69	0,40	0,58	294112	301556
test1	1024	1024	184	184	2,27	29,56	1036556	1079940
test4	1355	1446	355	97	16,98	20	3522916	3543764
addm4	1307	1307	212	212	1,42	1,84	1014764	1054248
dist	843	821	150	118	0,65	1,02	519736	543228
f51m	321	321	76	76	0,61	0,69	682632	692484
root	351	351	71	71	0,48	0,59	498836	513416
co14	196	196	14	14	0,05	0,06	145588	153544
sqr6	221	216	58	55	0,15	0,18	216156	220416
z4	252	252	59	59	0,16	0,19	265636	274336
m1	148	148	39	39	0,13	0,15	205632	210044
sqn	201	183	43	27	0,15	0,17	214812	232576
t4	76	76	23	23	2,71	2,06	1328996	1336140

Tableau I.4.b : Résultats comparatifs avec et sans choix de polarité (MCNC)

Légende:

NbLit_inv : Nombre de littéraux obtenu avec choix de la polarité
NbLit_new : Nombre de littéraux obtenu avec polarité positive (sans choix)
NbPT_inv: Nombre de monômes obtenu avec choix de polarité
NbPT_new: Nombre de monômes obtenu avec polarité positive
Cpu_inv: Temps de calcul en secondes avec choix de polarité
Cpu_new: Temps de calcul en secondes avec polarité positive
MaxMem_inv: Espace mémoire nécessaire avec choix de polarité
MaxMem_new: Espace mémoire nécessaire avec polarité positive

Les résultats exposés dans le tableau I.4 confirment les tendances enregistrées dans le cas du tableau I.3.

I.9. Conclusion

Les techniques de la minimisation symbolique sont très intéressantes du point de vue réduction de complexité et de robustesse, elles conduisent à la solution optimale plus rapidement et avec moins d'espace mémoire que les heuristiques de la minimisation explicite. De plus, seule cette génération de minimiseurs est capable de calculer la forme complémentée sans être exposée à des problèmes d'explosion en complexité de calcul. En effet, la possibilité de sélection de phase est très importante pour certaines applications comme la synthèse sur CPLD.

Chapitre II. Expression factorisée et réinjection de fonction Booléenne

II.1. Introduction

Dans le chapitre précédent nous nous sommes intéressés à l'écriture de fonctions booléennes sous forme de somme de monômes. La fonction objective était essentiellement le nombre de monômes et leur taille. Dans ce chapitre nous nous intéressons aux formes factorisées et plus précisément à la granularité des sous-expressions booléennes. Cette granularité a une relation directe avec la cible technologique sur laquelle la fonction booléenne sera implantée. En effet, les sous-expressions booléennes auront une "taille" adaptée à la cellule cible. Dans ce chapitre nous étudierons 2 processus complémentaires:
- La factorisation qui crée à partir d'une expression en somme de monômes des sous-fonctions avec une certaine complexité .
- La réinjection qui consiste à supprimer ces sous-fonctions dans l'hypothèse où la taille de la sous-fonction est trop réduite.
Dans le cadre de cette thèse, une méthode de factorisation algébrique a été implantée ainsi que des méthodes innovatives de réinjection de sous-fonctions booléennes.

II.2. Factorisation algébrique

II.2.1. Définitions préliminaires

• ***produits algébrique et booléen***
Le produit de deux fonctions f et g est dit algébrique si $supp(f) \cap supp(g) = \emptyset$. Par exemple, (a + b).(c + d) = a.c + a.d + b.c + b.d; sinon, le résultat du produit nécessite des opérations booléennes pour supprimer les redondances et le produit est dit booléen. Par exemple (a + b).(a + c) = a + b.c.

• ***Division algébrique et booléenne***

Soit f et g deux fonctions booléennes. g est un diviseur algébrique (booléen) de f si il existe un ensemble de monômes maximale h, tel que f = g.h + r et g.h est un produit algébrique (booléen), r est l'ensemble de monômes dit reste. Si r = Ø, g est dit facteur algébrique (booléen).

• ***Substitution algébrique***

Soit 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 $\overline{G}$.

• ***Expression libre "cube-free"***

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.

• ***Noyau et conoyau***

Le noyau et le conoyau sont l'ensemble des expressions:

K(F) = {G / G = F/C, C un monôme et G est libre}

Le monôme C utilisé pour obtenir le noyau K = F/C est appelé le 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 booléenne 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 :

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

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

Le couple conoyau/noyau de F_2 est :

$$\{(a, \underline{c + d})\}$$

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

• Compatibilité algébrique

Soit 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 :

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

Exemple :

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

$K = (c + d)$	Un noyau de l'expression de F.
$C = a.b$	Le conoyau associé au noyau K.
$K' = (b + c)$	Un noyau de l'expression de F.
$C' = a.d$	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.

II.2.2. Génération des noyaux algébriques et des monômes communs

L'algorithme proposé pour la génération des noyaux est celui présenté dans [Brayton82] et [Brayton87a, Brayton87b]. Cet algorithme est fondé sur l'ordonnancement des littéraux d'une fonction booléenne, puis sur la mise en facteur des littéraux un par un, afin de trouver les différents couples conoyaux/noyaux.

L'algorithme de génération des noyaux est le suivant :

```
Noyaux (F)

{
    m le monôme, de plus grand dégré, facteur algébrique
    noyauxList = rnoyaux(0, F/m);
    si m = 1 alors noyauxList = noyauxList + F
}



rnoyaux (j, F)

{
    noyauxList = {F}
    pour i de j+1 à n faire
    {
     si li apparaît dans plus d'un monôme de F alors
      {
        m est le monôme, de plus grand degré, facteur algébrique de F/li
        si lk n'appartient pas à m pour tous k≤i alors
           noyauxList = noyauxList + rnoyaux(i, F/(li.m))
      }
    }
  retourner (noyauxList)
}
```

II.2.2.1 Explication de l'algorithme de l'extraction des noyaux

Dans cet algorithme de génération de noyaux algébriques, on commence par ordonnancer les littéraux de la fonction booléenne $\{l_1, l_2, l_3,..., l_n\}$. Ensuite on met en facteur le plus grand monôme m facteur algébrique de la fonction f, on appelle enfin la procédure rnoyaux (0, f/m).

Dans la procédure rnoyaux (j, f) on divise l'expression f par tous les littéraux l_{j+1} jusqu'à l_n. Pour chaque division par l_{k+1} et par le facteur algébrique m de $f/(l_{j+1}.l_{j+2}....l_k)$ $(j < k \leq n)$ on appelle récursivement rnoyaux(k, $f/(l_{j+1}.l_{j+2}...l_k.m)$) pour extraire tous les noyaux de $f/(l_{j+1}.l_{j+2}...l_k.m)$. L'appel récursif est redondant si le facteur algébrique m contient un littéral l_i avec $i \leq j$, car les noyaux associés sont déjà générés. Cet algorithme pourrait être utilisé également pour générer les conoyaux.

Le nombre de noyaux d'une expression peut croître d'une façon exponentielle par rapport au nombre de littéraux du support de celle-ci, ceci est particulièrement évident dans le cas où la fonction a des variables symétriques, car chaque permutation des variables symétriques donne un nouveau noyau. Cependant, dans la pratique l'ensemble des noyaux est relativement petit.

Exemples:

Soit la fonction f :

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

L'arbre généré par l'algorithme ci-dessus est le suivant :

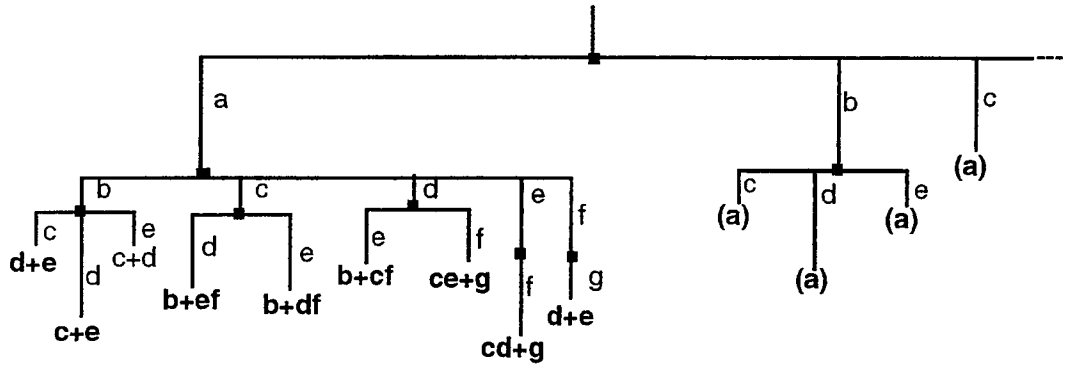


Figure II.1. Représentation des noyaux générés

La racine de l'arbre est la fonction f qui est le noyau du plus haut degré, les feuilles de l'arbre correspondent aux noyaux de degré 0; à chaque niveau L de l'arbre nous avons les noyaux de degré D-L avec D la profondeur de l'arbre. Les lettres sur les branches correspondent aux littéraux qu'on élimine de l'expression initiale à chaque étape, on remarque que ces littéraux sont ordonnés de la même façon qu'on les lit suivant la largeur ou la profondeur de l'arbre.

Le "(a)" de la branche (b.c) veut dire qu'on est ramené à la branche (a) pour laquelle tous les noyaux de f/a ont été générés, donc en particulier ceux de f/(a.b.c).

II.2.2.2 Génération des monômes ou parties de monômes communs

La génération des monômes ou parties de monômes communs est fondée sur l'algorithme de génération de noyaux.

L'algorithme utilisé est le suivant :

```
Monôme_commun
{
      Fi = Fi.vi
      F = ∑ Fi
      Noyaux (F)
      C = {Cj conoyau/ aucune vi n'appartient à Cj}
      retourner (C)
}
```

$F_1 = a.b.c.d + d.e + h$
$F_2 = a.b.c.e + d.e + h$
on construit F de la façon suivante :
$F = a.b.c.d.v_1 + d.e.v_1 + h.v_1 + a.b.c.e.v_2 + d.e.v_2 + h.v_2$
Les noyaux et les conoyaux associés où v_1 et v_2 n'apparaissent pas sont :
$\{a.b.c.(d.v_1 + e.v_2), d.e.(v_1 + v_2), h.(v_1 + v_2)\}$
a.b.c est une partie de monômes commune à F_1 et F_2.
d.e et h sont deux conoyaux associés au même noyau $(v_1 + v_2)$ cela veut dire que (d.e + h) forme une somme de monômes communs.

II.2.3. 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. Cependant, 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 V. Chaque sommet v de V correspond à un diviseur d. On donne un poids à chaque sommet v qui correspond au gain associé au diviseur d et deux sommets v_1 et v_2 sont liés par une arrête si les candidats diviseurs correspondants sont incompatibles.
Trouver l'ensemble de tous les diviseurs de gain maximal revient exactement à résoudre le problème de la recherche du stable de poids maximal dans G. Ce problème est NP complet, 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 [Brayton87c]. 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. Par exemple, une expression qui comporte 100 monômes pour 10 variables peut facilement générer un million de rectangles. Dans ce cas, le choix de l'algorithme de synthèse peut dépendre de la complexité des expressions Booléennes et permet d'utiliser une méthode ou une autre, ce qui lui procure un bon compromis temps de calcul / qualité des résultats.

La deuxième méthode est un **algorithme glouton** "greedy". 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.

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.

II.2.3.1 Calcul de gain des candidats diviseurs

- Calcul de gain en littéraux pour les noyaux locaux :

Le noyau apparaît dans une seule fonction Booléenne, le gain est calculé comme suit:

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

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

- Calcul de gain en littéraux pour les noyaux globaux :

Le noyau K est un noyau de plusieurs fonctions Booléennes $(f_j)_{j=1..n}$. 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 :

$$Gain(K) = \sum_{j=1}^{n}\left(\sum_{i=1}^{m_j}((NbMon(K)-1).NbLit(C_j^i)) + (m_j-1).NbLit(K)\right) + (p-1).(NbLit(K)) - p$$

Où $(C_j^i)_{i=1..m_j}$ sont les conoyaux associés au noyau K dans la fonction f_j

- Calcul de gain en littéraux 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 associée à un monôme est la suivante :
Gain(m) = (NbOcc(m) - 1). NbLit(m) - NbOcc(m)

II.2.4. Division et substitution algébriques

La méthode de division algébrique par l'expression d'un noyau K permet, non seulement de mettre le conoyau C en facteur, mais de trouver le plus grand quotient H, de la division d'une fonction F par ce noyau K tel que $M_H \supseteq C$.

II.2.4.1. Principe de la division algébrique

Soit deux fonctions polynomiales F et D. Le principe de la division algébrique est la recherche du quotient H de la fonction D dans l'expression de F. F s'écrit alors : F = D.H + R, R est le reste tel que M_R est le sous-ensemble de monômes de M_F non impliqués dans le produit algébrique D.H.
L'implantation algorithmique de cette division algébrique a été introduite par Brayton [Brayton87b] :

```
Division_Algébrique(F,D)
    {
        U = restriction de F aux littéraux de D
        V = restriction de F aux littéraux qui ne sont pas dans D
                        /* u_j v_j  est le j^ème monôme de F          */
        V_i = { v_j ∈ V / u_j = d_i }
                        /* Recherche du quotient de la division de F  */
                        /* par chaque monôme d_i                      */
                        /* d_i est le i^ème monôme de D et V_i obtenu */
                        /* en divisant F par u_i                      */
        M_H = U M_Vi
                        /* H est le quotient de F par D               */
        M_R = M_F - M_D.H
                        /* R est le reste de la division de F par D   */
        retourner (H,R)
    }
```

Exemple :

Soit F une fonction de 5 monômes :

$F = \sum_{i=1}^{5} mi = a.b.e.f + c.d.e.f + a.b.g.h + c.d.g.h + a.b.k$ Soit D une fonction de 2 monômes : D = a.b + c.d

La restriction de F aux littéraux de D est :

U = a.b + c.d + a.b + c.d + a.b

La restriction de F aux littéraux qui ne sont pas dans D est :

$V = e.f + e.f + g.h + g.h + k$

$\Downarrow$

$V_{a.b} = \{e.f + g.h + k\}$

$V_{c.d} = \{e.f + g.h\}$

$M_H = M_{V_{a.b}} \cup M_{V_{c.d}} = \{e.f, g.h\}$

$M_R = a.b.k$

retourner(e.f + g.h, a.b.k)

L'expression de F, écrite sous sa forme factorisée, devient :

$$F = (a.b + c.d).(e.f + g.h) + a.b.k$$

Remarques :

• Si on appelle n_F le nombre de monômes de F et n_D le nombre de monômes de D, la complexité du produit algébrique est de l'ordre de $O(n_F * n_D)$.

• Si la fonction D n'est pas un diviseur algébrique de F, cela veut dire que :

- D contient au moins un littéral non contenu dans F
- D contient plus de monômes que F
- Pour chaque littéral de D, le nombre d'occurrences dans D est supérieur à celui dans F
- F est présente dans l'expression de D

Si l'une des conditions précédentes est satisfaite, il n'est pas nécessaire d'effectuer la procédure de division algébrique.

La substitution algébrique cherche à diviser non seulement par un diviseur D, mais également par son complément $\overline{D}$. L'algorithme de la substitution algébrique est le suivant :

```
Substitution _algébrique (F, D)
{
 (H, R) = Division_algébrique (F, D)
 Calcul du complement de D
 (H', R') = Division_algébrique (R,D)
 x = D
 /* Substitution dans F */
 F=H.x+H'.x+R'
}
```

L'expression polynomiale de F est transformée par la substitution algébrique de D de la façon suivante :

$$F = \frac{F}{D}.x + \frac{F}{\overline{D}}.\overline{x} + R'$$

avec $x = D$

Remarque :

Cette substitution fait appel à 2 divisions : k.O(n*m) (k est une constante) et un calcul de complément.

En ce qui concerne la substitution algébrique, le gain ne peut être calculé d'une façon exacte au stade de la génération des noyaux, car ce calcul n'est possible que si on effectue la division et la substitution algébriques, ce qui n'est pas envisageable pour tous les noyaux candidats. On se contente donc de faire la sélection des noyaux selon la formule de gain donnée plus haut, qui en fait constitue une borne inférieure de la fonction gain réelle.

Exemple:

Pour illustrer les algorithmes de division et de substitution algébriques, on considère une fonction F de 8 monômes :

$$F = \sum_{i=1}^{8} m_i = a.d + a.e + b.d + b.e + c.d + c.e + \bar{a}.\bar{b}.\bar{c}.f + g$$

L'algorithme de division algébrique par le noyau (a + b + c) donne le résultat suivant :

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

La substitution algébrique permet d'obtenir le résultat final suivant :

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

II.2.5. Filtrage des candidats diviseurs

Le filtrage des candidats diviseurs [Abouzeid92] et [Belrhiti93] est fait selon le critère de la taille du candidat diviseur. Ce critère permet d'éviter la création de sous-fonctions booléennes de taille inférieure à celle de la cellule de la cible technologique.

Ce 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 monômes et de variables, est supérieure à une valeur donnée k. 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 en plus d'avoir une sortance très élevée et conduisent à une saturation des ressources en ports d'entrées/sorties des macrocellules.

II.2.6. Algorithme général

La factorisation passe par deux étapes principales qui sont :

Étape 1

- La division par les conoyaux ou par les noyaux et leurs compléments.

Étape 2

- Recherche des monômes ou parties de monômes communs.

La question qui s'impose ici est la suivante:
Doit-on suivre la première étape jusqu'à épuisement des noyaux et seulement après, passer à la deuxième étape ?
La réponse par oui à cette question favorise d'abord la division algébrique par rapport à la recherche de monômes ou parties de monômes communs.
La solution adoptée qui a donné des résultats satisfaisants dans la pratique est de répéter les deux étapes alternativement. Cette discrétisation est faite suivant l'algorithme suivant :
Soit la suite décroissante d'entiers $S = \{s0, s1, ..., sn = 0\}$

```
début
S = s0
Tant qu'il existe des candidats faire
        /*Division par les noyaux et leurs compléments  */
        Substitution algébrique des noyaux candidats (gain des noyaux>= S)
        /*Recherche des monômes communs                 */
        Recherche de parties communes de gain >= S
        S <-- Succ ( S )
fin tant que
fin;
```

Il reste à trouver la suite S. Dans notre algorithme de factorisation nous avons déterminé d'une façon empirique la suite S après plusieurs essais, laquelle finalement correspond à l'expression suivante:

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

Exemples :

$f1 = a.b.c.d + a.b.e.f + h$

$f2 = a.b.c.d + c.d.k + j$

Les noyaux de l'ensemble des fonctions booléennes sont

$K1 = c.d + e.f$ gain (K1) = 2

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

Si on divise par K_1 et K_2 (possible car ils sont compatibles) l' ensemble des fonctions devient :

$f_1 = a.b.(c.d + e.f) + h$

$f_2 = c.d.(a.b + k) + j$

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

Cet exemple illustre l'incompatibilité qui peut survenir entre les deux étapes de la factorisation :

- La division par les conoyaux ou par les noyaux et leurs compléments.
- La recherche des monômes communs.

II.3. Représentation factorisée de fonctions booléennes

II.3.1. Arbre de factorisation

À une expression factorisée d'une fonction booléenne F est associée une arborescence de racine F dont les feuilles sont étiquetées par les variables de la fonction et dont les nœuds correspondent aux opérateurs booléens ET (+), OU (* ou .) et NOT (! ou -). L'orientation est supposée aller des feuilles vers la racine.

Exemple:

Soit une équation booléenne factorisée F:

F = a.b.c + (!a+!b).(!e+!f) + !c.!d.(i+j.k)+!g.(!i+!k)

L'arborescence correspondante est donnée dans la figure II.2.

• On appellera une telle représentation associée à une expression factorisée **arborescence de factorisation** ou plus brièvement **arbre de factorisation**.

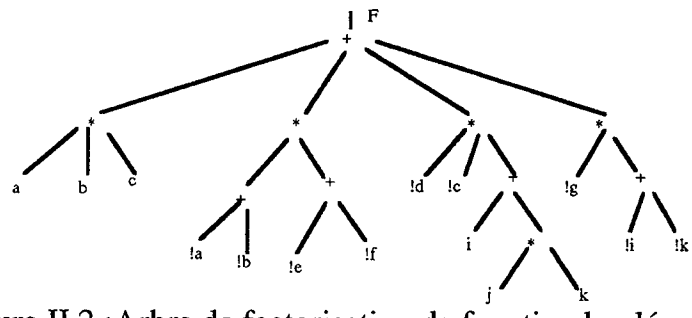


Figure II.2 :Arbre de factorisation de fonction booléenne

• **La profondeur** de cette arborescence est le nombre maximal d'opérateurs sur un chemin reliant la racine à une feuille.

• Les nœuds prédécesseurs d'un nœud de l'arbre constituent **le cône prédécesseur** de ce nœud. Le cône prédécesseur d'un nœud est montré dans la figure II.3.

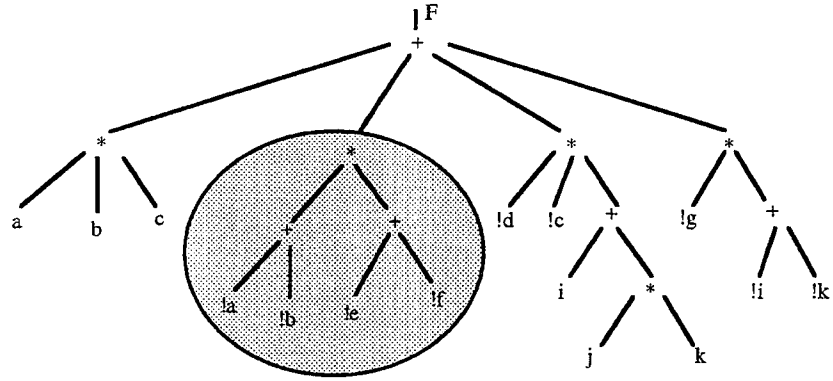


Figure II.3 : Cône prédécesseur d'un nœud

II.3.2. Arborescence hiérarchisée

Tout nœud de l'arborescence définit une sous-fonction booléenne incluse dans F dont la représentation est le cône prédécesseur de ce nœud.

Une **représentation hiérarchisée** consiste à remplacer ce cône par une nouvelle feuille appelée SF; la représentation de SF peut être considérée comme une arborescence séparée. On parle alors de **forêts d'arborescences**.

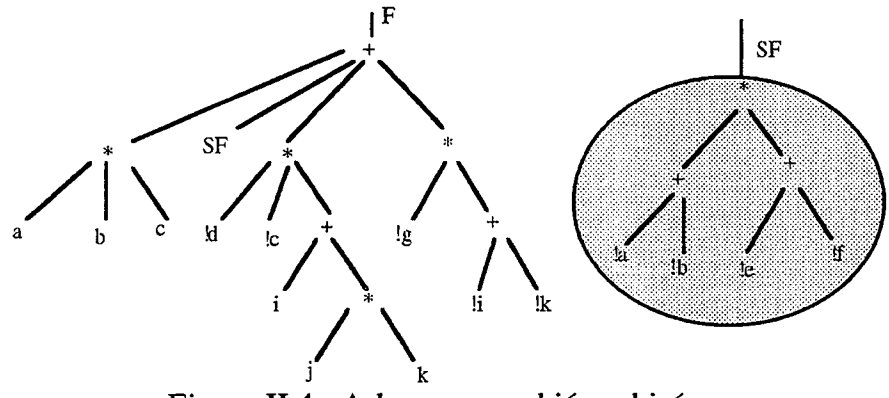


Figure II.4 : Arborescence hiérarchisée

S'il existe une seule feuille étiquetée SF dans la représentation hiérarchisée de la fonction booléenne, SF est dite **sous-fonction non-partagée**.

II.3.3. Sous-fonction partagée et graphe orienté hierarchique

Dans la représentation d'une fonction booléenne, une sous-arborescence peut apparaître plusieurs fois. Dans une représentation réduite, une telle sous-arborescence ne sera représentée qu'une seule fois.

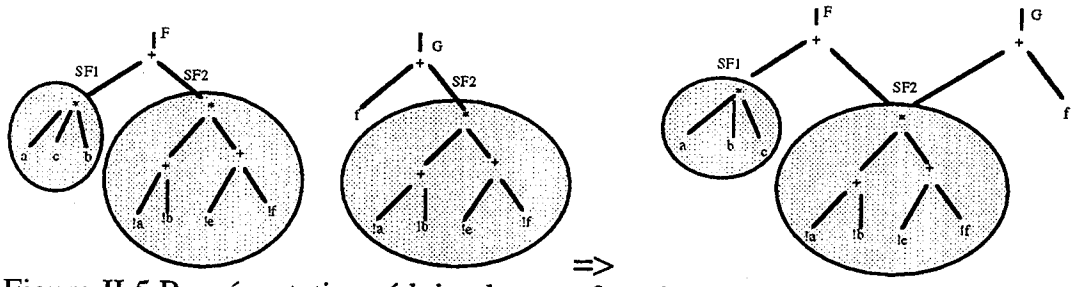


Figure II.5 Représentation réduite de sous-fonctions partagées/non-partagées

Comme précédemment, un tel graphe peut être également réduit en remplaçant la sous-arborescence par une feuille étiquetée SF.
Dans la figure II.5, SF_1 est une sous-fonction non-partagée. SF_2 est une sous-fonction partagée.

II.3.4. Graphe hiérarchique réduit

On appelle **graphe hiérarchique réduit**, le graphe composé de l'arborescence initiale de la fonction dans lequel toute sous-fonction partagée est représentée par une feuille et des arborescences séparées associées à toutes les sous-fonctions partagées.

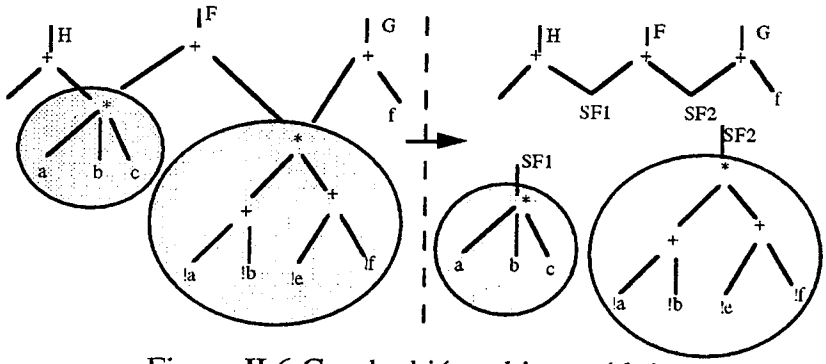


Figure II.6 Graphe hiérarchique réduit

II.3.4. Degré de hiérarchie

i) Le ***degré de hiérarchie*** est le nombre maximal de sous-fonctions rencontrées dans l'ensemble des chemins reliant la racine du graphe hiérarchique réduit et une feuille étiquetée par une variable.
ii) une sous-fonction SF1 est appelé ***père*** d'une autre SF2 si et seulement si l'arborescence de SF1 admet au moins une feuille étiquetée par la variable SF2.
iii) Le ***degré de partage*** de deux sous-fonctions SF1 et SF2 est le nombre d'étiquettes différentes de la sous-arborescence pointée par les deux arborescences: celles de SF1 et de SF2.

Exemple:
Dans la figure II.7 le degré de hiérarchie dans la décomposition de F1 est de 4 et celui de F3 est de 5.

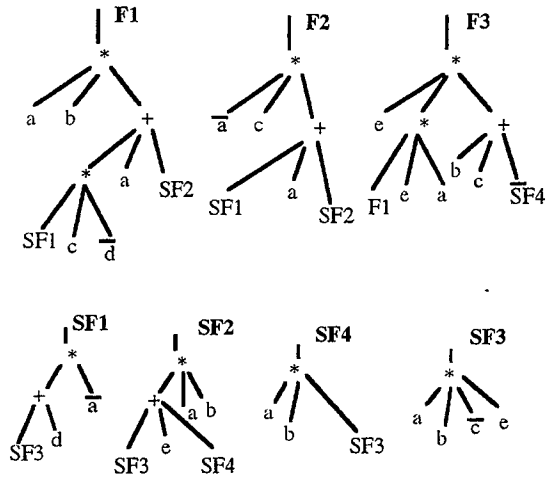


Figure II.7 Exemple du calcul du degré de hiérarchie du graphe hiérarchique réduit

II.4. Méthodes classiques de Réinjection

II.4.1. Réinjection élémentaire

Soit G un graphe hiérarchique réduit, et F la racine d'une arborescence de G. Soit SF une sous-fonction de F représentée par une feuille. On appelle **réinjection élémentaire** de SF dans F la substitution de la variable SF par son

arborescence associée dans celle de F. Une réinjection est dite totale de SF si celle-ci a été réinjectée dans tous ses pères; elle est partielle si elle concerne seulement une partie de ses pères.

Exemple:

La figure ci-dessous montre la réinjection élémentaire de SF_2 dans G. Cette réinjection est partielle; car la variable SF_2 figure toujours comme étiquette de F.

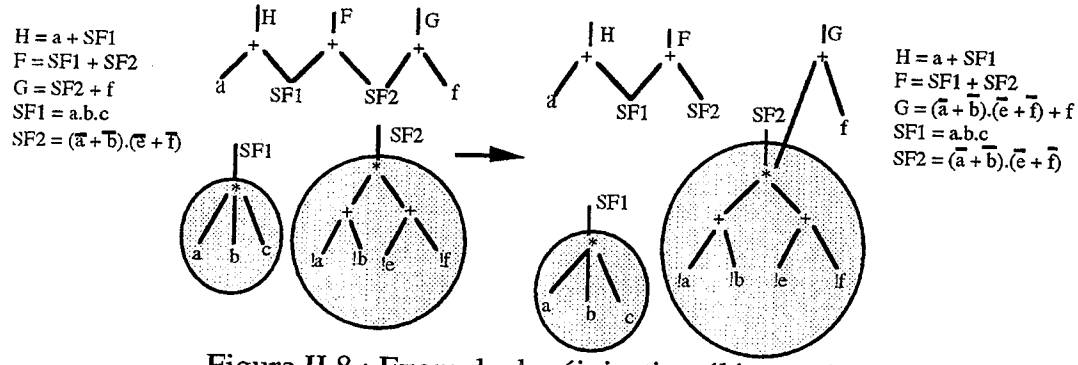


Figure II.8 : Exemple de réinjection élémentaire

II.4.2. Méthodes classiques de réinjection

La ***réinjection totale*** consiste à réinjecter toutes les sous-fonctions jusqu'à obtenir des équations en somme de monômes, laissant ainsi toute la liberté à la synthèse de mettre en place une factorisation adaptée, appelée par Brayton le processus créatif. Cette technique est très efficace pour les circuits de petite complexité et ne peut être envisagée pour les circuits à moyenne et grande complexité.

Soit G le graphe hiérarchique réduit représentant le réseau booléen. On exposera d'abord l'algorithme d'**étiquetage par degré d'hiérarchie**, qui sera souvent utilisé dans les algorithmes proposés de réinjection.

```
Étiquetage (G, q)
{
 Si G ∈ I alors q=0
                Sinon
                {
                 Soit Child(G) = {ensemble des noeuds fils de G}
                 {
                      Si (∀c ∈ Child(G), et c est marqué)
                      {
                            q = Max_{c∈Child(G)} (Marque (c)) + 1
                      }
                      Sinon
                      {
                       pourtout c ∈ Child(G) faire
                       {
                            Étiquetage (c, pc)
                       }
                       q = Max_{c∈Child(F)} (pc) +1
                      }
                 }
                }
}

Réinjection_totale (G)
{
        Étiquetage (G, q)
        Pour toutes les sous-fonctions SF
        {
                Réinjecter SF dans tous ses successeurs
                Si SF est une sortie locale
                        Supprimer le noeud de SF de G
        }
}
```

La réinjection totale non contrôlée peut amener à une explosion combinatoire dans le calcul des expressions en somme de monômes ou dans la décomposition technologique.

L'***élimination*** [Brayton87b] est fondée sur la notion de gain, considérée comme la façon la plus simple pour la réinjection partielle [Brayton87b]. L'élimination consiste à réinjecter toutes les sous-fonctions dont le gain en littéraux est inférieur à une valeur donnée. Un exemple de fonction gain est donné ci-dessous. Elle est essentiellement liée au nombre de littéraux et au degré de partage de sous-fonction. Le gain d'une sous-fonction Sf est calculé comme suit:

Cas 1 : Sf est une sortie locale:

$$Gain(Sf) = NbOcc(Sf) \times (NbLit(Sf) - 1) - NbLit(Sf)$$

Cas 2 : Sf est une sortie primaire:

$$\text{Gain}(Sf) = \text{NbOcc}(Sf) \times (\text{NbLit}(Sf) - 1)$$

Le gain en littéraux d'une sous-fonction Sf correspond à l'augmentation en littéraux occasionnée par la réinjection de Sf dans tous ses successeurs. Cette différence entre les deux cas provient du fait que si on réinjecte une sous-fonction locale dans tous ses successeurs, on peut éliminer la variable locale et son expression du réseau booléen, ce qui n'est pas le cas si la sous-fonction représente une sortie primaire.

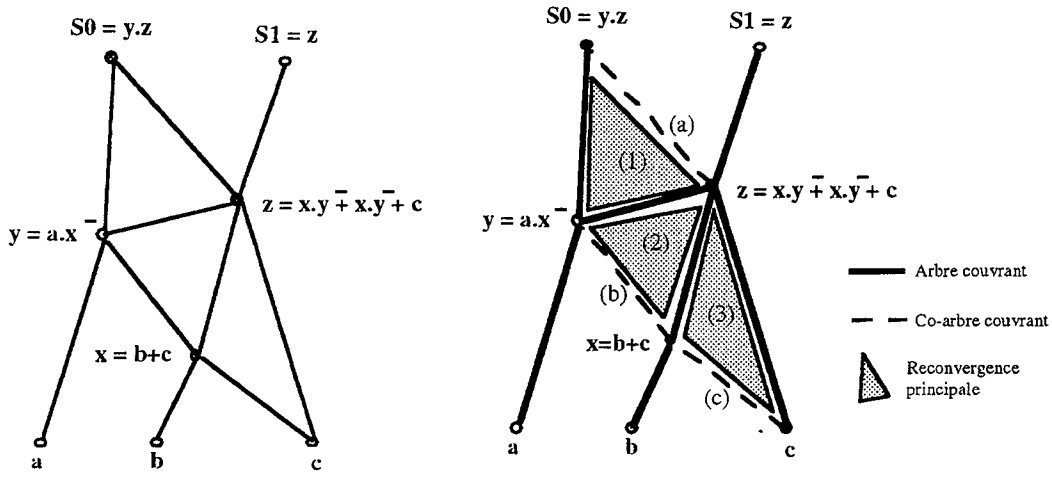
```
Elimination (G, Min_Gain)
{
Etiquetage (G, q)
Pour toutes les sous-fonctions Sf
{
   Si Sf est une sortie locale
    Gain (Sf) = NbOcc(Sf)*(NbLit(Sf)-1)-NbLit(Sf)
   Sinon
    Gain (Sf) = NbOcc(Sf)*(NbLit(Sf)-1)

  Si  (Gain (Sf) < Min_Gain)
  {
   Réinjecter Sf dans tous ses successeurs
   Si Sf est une sortie locale
     Supprimer le noeud de Sf de G
  }
}
}
```

Au lieu de la fonction gain en littéraux, on peut imaginer toute sorte de fonction coût, en général en fonction de la granularité de la cellule de base de la cible technologique. Le nombre de variables du support des sous-fonctions ou le nombre de monômes est le plus communément utilisé. Ce type de réinjection est plutôt appliqué pour éliminer les sous-fonctions de taille trop petite, produisant une sous-utilisation ou un taux de remplissage très faible de la cellule cible (le cas des FPGA à base de LUT ou les CPLD par exemple) lors de la phase de décomposition technologique. Elle pourrait également être appliquée pour éliminer toutes les sous-fonctions non partagées, en mettant l'argument Min_Gain à 0.

II.4.3. Méthodes fondées sur la reconvergence dans le circuit

II.4.3.1 Définitions préliminaires

On considère un graphe hiérarchique réduit G. On définit un ***arbre couvrant*** T de G comme étant un arbre contenant tous les sommets de G. Le ***co-arbre couvrant*** est défini comme étant l'ensemble des arrêtes de G n'appartenant pas à T. Une ***reconvergence principale*** est construite par le cycle engendré par une arrête du co-arbre couvrant et T. La fusion de reconvergences principales intersectant forment une ***reconvergence*** dans le circuit. Les sommets d'une reconvergence forment un ***graphe reconvergent.***



a- Graphe hiérarchique réduit G b- Les Reconvergences principales dans G

Figure II.9 : Notion de reconvergence dans le circuit

II.4.3.2 Principe de la méthode

La méthode de réinjection partielle fondée sur le concept de la reconvergence[Nakamura93] consiste à partitionner le graphe G en sous-graphes reconvergents. Chaque graphe reconvergent est réinjecté totalement, pour être ensuite minimisé et factorisé. L'idée d'exploiter la structure de reconvergence permet de tenir compte de l'intersection en terme de couverture booléenne. En conséquence, la réinjection des expressions des fonctions booléennes appartenant au même sous-graphe reconvergent permet d'obtenir un potentiel de minimisation logique.

II.4.3.3 Difficultés liées à l'application de la méthode

La première difficulté liée à l'application de cette méthode apparaît dans le cas où toutes les entrées primaires sont reliées par un chemin à toutes les sorties primaires. La partition du graphe est réduite à lui-même. Cette méthode conduit donc à la réinjection totale de tout le graphe. L'auteur a introduit des contraintes limitant la complexité d'une partition. Les contraintes de complexité concernent tout d'abord l'intersection en terme d'entrées/sorties dont dépendent les reconvergences. Ces contraintes s'exercent également sur le plus long chemin appelé ***longueur de reconvergence*** dans le graphe reconvergent.
La deuxième difficulté réside dans le nombre d'arbres couvrants. En effet, le nombre d'arbres couvrants dans un graphe G est très grand, il est de l'ordre de $O(n^{n-2})$ [Lawler76], où n est le nombre de sommets dans G. Le choix d'un arbre couvrant donné entraîne la génération d'un ensemble de reconvergence principale associé. Face à la complexité de choix, l'auteur a privilégié un arbre couvrant équilibré.

II.4.4. Conclusion sur les méthodes classiques

Si nous passons en revue les méthodes classiques de la réinjection qui précède la synthèse logique, nous distinguons trois approches principales:

- (A1) La réinjection totale laissant toute la liberté à la synthèse logique de reconstruire le partage.
- (A2) L'élimination guidée par une fonction coût prédite après réinjection.
- (A3) La réinjection structurelle guidée par la reconvergence dans le réseau booléen.

L'approche (A1) est en fait une totale resynthèse ignorant la structure de l'expression initiale. Elle s'avère très efficace dans le cas où initialement le partage de la logique est de faible rendement. En outre, sa faiblesse apparaît dans le cas des circuits à structure régulière où le partage est quasiment optimal. De plus, du point de vue complexité de calcul, les risques d'explosion combinatoire ne sont pas négligeables.

L'approche (A2) est plus progressive et moins brutale que la première, elle tient compte du partage et ne présente aucun risque de débordement en complexité. Sa faiblesse principale est sa fonction coût qui ne prend pas en compte une éventuelle simplification booléenne après réinjection.

L'approche (A3) tient compte indirectement d'éventuelles simplifications, ce qui se traduit par la reconvergence des signaux dans le réseau booléen. Elle tend dans certains cas à la réinjection totale. Plusieurs variantes ont proposé de contrôler les problèmes de complexité résultants, mais parfois au détriment de l'efficacité de l'approche.

II.5. Formulation et analyse des approches possibles

II.5.1. Formulation de la réinjection en un problème d'optimisation combinatoire

Le problème de la réinjection des fonctions booléennes peut être ramené à celui de l'optimisation combinatoire [Burlet92] suivant :
Étant donné un graphe hiérarchique réduit G. Quel est l'ensemble de réinjections élémentaires qui conduit à ce que la complexité des expressions booléennes résultantes, en terme d'une fonction coût, soit minimale? Plus formellement, si on note $(f_i)_{1 \le i \le n}$ l'ensemble des expressions booléennes de G, C() la fonction coût exprimant la complexité, la fonction objective $\sum_{1 \le j \le n}(C(f_j))$ doit être minimale.
Avant d'exposer l'approche proposée, nous exposons l'algorithme qui atteint la solution exacte et des heuristiques pour obtenir une solution optimale approchée.

II.5.2. Recherche de la solution exacte

Le but est de trouver toutes les combinaisons possibles de réinjections élémentaires avec leurs gains en terme d'une fonction objective donnée. Nous construisons la matrice de couverture où chaque ligne correspond à un noeud du graphe hiérarchique réduit G, et chaque colonne à une combinaison possible de réinjection élémentaire. Si une combinaison j contient un noeud i de G, on met un 1 dans la case (i, j) de la matrice. Il s'agit donc de résoudre le problème de couverture minimale sur cette matrice. Ce problème est NP-Complet [Garey79].

II.5.3. Recherche de la solution optimale approchée

Nous présentons deux heuristiques qui permettent de donner une solution approchée : gloutonne, et une variante de l'approche connue sous le nom de "FFD" (First Fit Decreasing) [Gondran85].

La première identifie les réinjections élémentaires possibles, et associe un poids en fonction du gain apporté à chacune d'elle. Cette approche consiste à sélectionner la réinjection élémentaire ayant le poids maximal et répéter la procédure jusqu'à ce qu'il n'y ait plus de réinjection élémentaire ayant un poids positif. La fonction poids doit tenir compte de la simplification booléenne après chaque réinjection élémentaire. En plus, la mise à jour des poids nécessite de refaire le calcul des gains pour tous les noeuds incidents au noeud résultat de la réinjection élémentaire sélectionnée. Le processus de calcul et de mise à jour des poids devient donc trop long pour être appliqué.

La seconde heuristique est en général utilisée pour résoudre le problème du remplissage des boites, connu sous le nom de problème de BIN PACKING [Gondran85]. Cette méthode consiste à trier les noeuds dans un ordre décroissant par rapport à leur fonction coût. On commence par mettre le noeud N_{max} de coût maximal dans une première boîte créée, on fixe une capacité maximale $M[N_{max}]$ de cette boîte en fonction du coût du noeud N_{max}. Tant que la fonction coût de la boite ne dépasse pas la capacité fixée on continue à réinjecter, sinon on crée une nouvelle boîte. La difficulté d'appliquer cette méthode réside dans le choix de la capacité en fonction du coût. Une solution possible consiste à prendre le coût du noeud comme capacité maximale, malheureusement cette solution a l'inconvénient de restreindre le nombre de réinjections élémentaires possibles.

II.6. Méthode proposée de réinjection : Recherche par amélioration itérative

L'approche proposée s'est fixée comme objectif de minimiser la complexité des expressions booléennes. Elle est fondée sur l'amélioration itérative (une sorte de recuit simulé) d'une fonction coût prédite (noté **FCP** dans la suite de l'exposé) dont l'expression est liée à la granularité technologique. Le critère de sélection de sous-fonction à réinjecter tient compte de son partage de la logique et de sa complexité. Toutes les sous-fonctions ne sont pas traitées simultanément. En échange, une stratégie de regroupement "Clustering" est proposée. L'itération sur cette procédure d'optimisation conduit à la convergence vers une solution optimale approchée.

II.6.1. La fonction coût prédite proposée (FCP)

La complexité d'un circuit réalisé sur des réseaux programmables (FPGA, CPLD) est estimée à l'aide du nombre de cellules nécessaires pour implanter l'ensemble des expressions booléennes. Ces cellules ont une capacité d'absorption de la logique qui se traduit par le nombre de monômes et de variables. Notre fonction coût prédictive est en fait le minimum théorique du nombre de cellules nécessaire pour implanter une expression booléenne. Elle est une borne inférieure de cette complexité, et s'écrit à l'aide de la formule suivante:

$$\textbf{Coût (f)} = \textbf{Max}\left(\left\lceil \frac{\textbf{NbVar (f)}}{\textbf{C}_{\textbf{NbVar}}\textbf{ (Cellule)}} \right\rceil, \left\lceil \frac{\textbf{NbMonôme (f)}}{\textbf{C}_{\textbf{NbMonôme}}\textbf{ (Cellule)}} \right\rceil\right)$$

NbVar (f)	: Nombre de variables du support de f.
NbMonôme (f)	: Nombre de monômes de l'expression de f.
$\mathbf{C_{NbVar}}$**(Cellule)**	: Capacité en nombre de variables de la cellule cible.
$\mathbf{C_{NbMonômes}}$**(Cellule)**	: Capacité en nombre de variables de la cellule cible.

Définition de la granularité d'une cellule

On définit la granularité d'une cellule "Cell" noté **Gr[Cell]** par le couple ($\mathbf{C_{NbVar}}$, $\mathbf{C_{NbMonôme}}$), où $\mathbf{C_{NbVar}}$ est la capacité en nombre de variables de la cellule de la technologie cible, et $\mathbf{C_{NbMonôme}}$ est la capacité en nombre de monômes de la cellule de la technologie cible.

La méthode de réinjection proposée fait appel à plusieurs passes avant de converger vers une solution optimisée. Une passe comprend un parcours de tous les noeuds du graphe hiérarchique réduit pour examiner d'éventuelles réductions de complexité locale en terme de la FCP lors d'une réinjection élémentaire. Au cours d'une passe, l'algorithme ne traite pas tous les noeuds du réseau simultanément mais identifie un sous-ensemble de candidats à la réinjection.

Les étapes d'une passe sont les suivantes:

- **Identification des noeuds candidats à la réinjection.**
- **Stratégie de regroupement "Clustering" des candidats à la réinjection.**
- **Sélection des candidats à la réinjection.**

II.6.2. Identification des noeuds candidats à la réinjection

L'ordre de l'exploration du réseau est du bas vers le haut. Il suit l'ordre croissant du degré hiérarchique du graphe réduit. Il est donc nécessaire de mettre à jour l'étiquetage hiérarchique à chaque passe. Un noeud visité, peut être réinjecté partiellement ou totalement. Si aucune des réinjections de ce noeud n'a eu lieu, le noeud est marqué "visité". Les noeuds candidats doivent satisfaire les deux conditions suivantes:

- N'être pas marqués "visité".
- Avoir comme fils uniquement des noeuds marqués "visité" ou de degré hiérarchique 1 (entrées primaires).

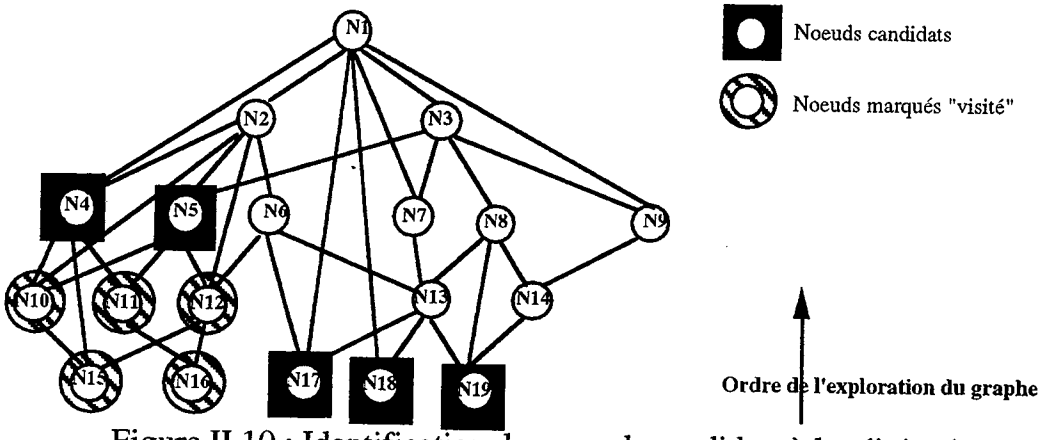


Figure II.10 : Identification des noeuds candidats à la réinjection

N4, **N5**, **N17**, **N19** et **N18** sont **candidats à la réinjection**. Les noeuds N10, N11, N12, N15 et N16 sont marqués **"visité"**. **N4** et **N5** ont respectivement comme fils N10, N15, N11 et N11, N12 qui sont tous marqués "visité". **N17**, **N18**, et **N19** ont comme noeuds fils uniquement des entrées primaires.

II.6.3. Stratégie de regroupement des candidats à la réinjection

L'ordre du déroulement du processus de réinjection dans un graphe hiérarchique réduit est extrêmement important. En effet, une exploration aléatoire peut ramener à des résultats imprévisibles.

Le regroupement est un processus qui précède la réinjection et qui concerne les noeuds candidats ou les noeuds pères candidats à une réinjection. L'objectif du regroupement proposé est de pallier le problème de l'ordre du déroulement du processus de réinjection.

Comme nous l'avons formulé au début de ce paragraphe, le but de notre réinjection est de minimiser la FCP du graphe hiérarchique réduit; pour ce faire, la réinjection vise les deux objectifs suivants:
- Réduire la complexité des noeuds.
- Diminuer le nombre de noeuds sans augmenter la FCP globale du graphe hiérarchique. L'élimination d'un noeud se fait par réinjection totale dans tous ses noeuds pères.

Considérons un ensemble de noeuds candidats composé du noeud père et de ses fils. Notre approche passe par 2 étapes imbriquées de regroupement:
La première étape de regroupement consiste tout d'abord à associer à chaque noeud candidat les pères ayant le plus petit degré hiérarchique. Les candidats "père" d'une réinjection sont donnés dans le tableau qui reprend l'exemple de la figure II.10 :

Noeuds candidats	Noeuds pères candidats
N4	N2
N5	N2, N3
N17	N13
N19	N13, N14
N18	N13

La première étape de regroupement est ensuite formée autour des noeuds pères. Chaque regroupement est bijectivement associé à un noeud père. Les regroupements formés sont donc:
(**N2**, N4, N5); (**N3**, N5); (**N13**, N17, N18, N19); (**N14**, N17)

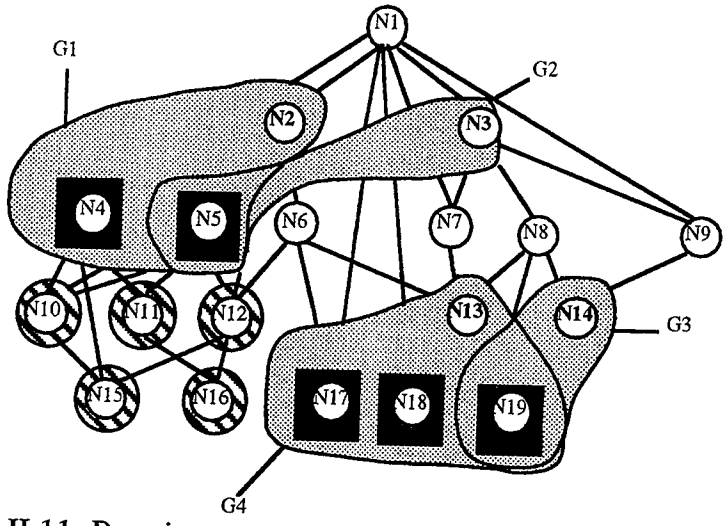


Figure II.11 :Premier regroupement des candidats à la réinjection

Le deuxième regroupement est un filtrage des ensembles générés par le premier. Il consiste à mettre ensemble les noeuds fils ayant une intersection de support entre eux ou avec le noeud père.

Le deuxième regroupement a pour but de réduire la complexité du noeud père tout en diminuant la sortance des noeuds fils dans le graphe, sachant que lorsque la valeur de la sortance d'une sortie locale atteint 0, le noeud est éliminé du graphe hiérarchique réduit.

Soit le groupe $G_3 = (N_{13}, N_{17}, N_{18}, N_{19})$; on suppose les hypothèses suivantes:

$$\begin{cases} Supp(N_{17}) \cap (Supp(N_{18}) \cup Supp(N_{19}) \cup Supp(N_{13})) = \phi & (1) \\ Supp(N_{18}) \cap Supp(N_{13}) \neq \phi & (2) \\ Supp(N_{19}) \cap Supp(N_{13}) = \phi & (3) \\ Supp(N_{18}) \cap Supp(N_{19}) \neq \phi & (4) \end{cases}$$

On appelle ***noeud isolé*** celui qui n'a pas d'intersection de support avec les autres noeuds d'un même groupe. D'après l'hypothèse (1) N_{17} est un noeud isolé. Le reste des noeuds d'un même groupe forme des sous-regroupements de noeuds à support intersectant . Le seul sous-regroupement de noeuds à support intersectant est (N_{13}, N_{18}, N_{19}) d'après (2), (3) et (4). En général, il peut y avoir plusieurs sous-regroupements et des noeuds isolés dans un groupe.

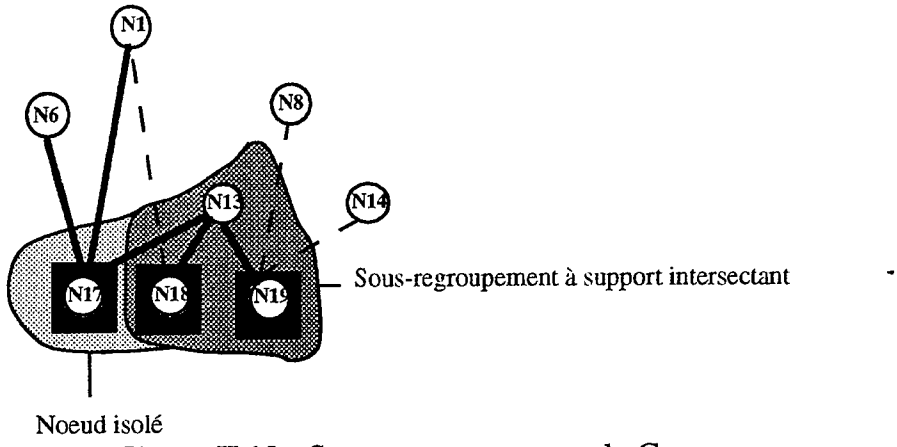


Figure II.12 : Sous-regroupements de G3

Les noeuds isolés sont destinés à être réinjectés totalement et éliminés du graphe, tandis que les noeuds fils du sous-regroupement à support intersectant sont destinés à être réinjectés dans le noeud père, afin de diminuer sa complexité, et en même temps diminuer leurs sortances.

II.6.4. Sélection des candidats à la réinjection

II.6.4.1 Définition de la fonction coût locale

La fonction coût locale est une fonction de coût estimée au niveau d'un groupement donné. Nous différencions deux types de groupement :

- Noeud isolé avec ses noeuds pères.
- Noeuds à support intersectant.

Le **gain** associé à une réinjection au niveau d'un groupement donné est **la valeur de la différence entre la fonction coût locale avant et celle après réinjection et minimisation logique**. La sélection des candidats à la réinjection nécessite la stricte positivité de la valeur du gain associé.

II.6.4.2 Noeuds isolés

La fonction coût locale avant la réinjection est calculée de la façon suivante :

$$C_{avant} = C_N + \sum_{i=1}^{m} C_{Pi}$$

N :	Noeud isolé candidat à la réinjection.
Pi, i = {1,...,m} :	Ensemble des noeuds pères de N.
CN :	FCP du noeud N.
CPi :	FCP du noeud père Pi.

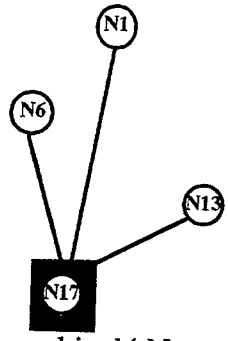


Figure II.13 : Noeud isolé N17 avant réinjection

Pour l'exemple de la figure II.13, la complexité initiale est de la forme:

$C_{avant} = C_{N17} + C_{N1} + C_{N13} + C_{N6}$

Le calcul de la fonction coût locale après réinjection et minimisation s'effectue selon la formule suivante :

Si N représente une sortie locale

$$C_{après} = \sum_{i=1}^{m} C_{Pi}$$

Si N représente une sortie primaire

$$C_{après} = \sum_{i=1}^{m} C_{Pi} + C_N$$

C_{Pi} : FCP du noeud père Pi après réinjection et minimisation logique.
C_N : FCP du noeud N.

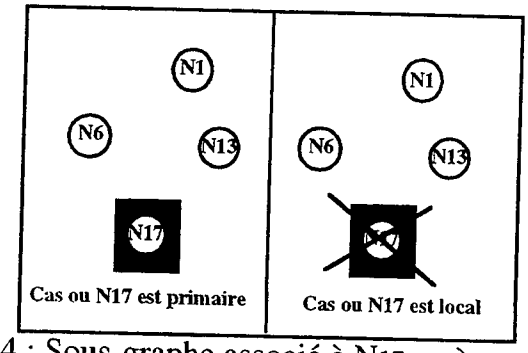


Figure II.14 : Sous-graphe associé à N17 après sa réinjection

II.6.4.3 Sous-regroupement de noeuds à support intersectant

Soit E = {N1, N2, ..., Nm} l'ensemble constituant le sous-regroupement à support intersectant candidat à la réinjection.

Le calcul de la fonction coût locale avant réinjection se fait selon la formule suivante:

$$C_{avant} = C_P + \sum_{i=1}^{m} \lceil \alpha_i . C_{Ni} \rceil$$

C_P : FCP du noeud père P

C_{Ni} : FCP du noeud fils N_i

$\lceil \alpha_i . C_{Ni} \rceil$: Terme associé à la FCP et la sortance du noeud fils N_i

Si N_i est une sortie primaire $\alpha_i = 0$

Si N_i est une sortie locale

$$\begin{cases} \alpha_i = \dfrac{1}{(\text{Sortance } (N_i))} \text{ si } (\text{Sortance } (N_i)) \geq 1 \\ \text{Sinon le coefficient } \alpha_i \text{ est nul} \end{cases}$$

α_i est inversement proportionnel à la sortance du noeud Ni

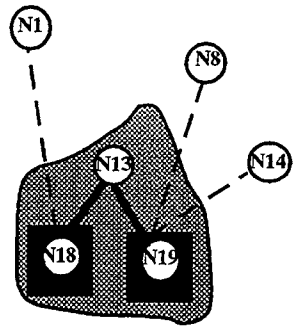


Figure II.15 : Sous-regroupement (N13, N18, N19) avant réinjection

Pour l'exemple de la figure II.15 la fonction coût locale avant réinjection est de la forme:

$$C_{avant} = C_{N13} + \left\lceil \frac{C_{N19}}{3} \right\rceil + \left\lceil \frac{C_{N18}}{2} \right\rceil$$

La formule de la fonction coût locale après réinjection dans P et sa minimisation logique est la suivante:

$$C_{après} = C_P + \sum_{1 \le i \le m} \beta_i . C_{Ni}$$

C_{Ni} : FCP du noeud fils Ni

C_P : FCP du noeud père P

β_i : Coefficient nul si :

Ni est local et sortance(Ni) ≤ 1

égal à 1 sinon

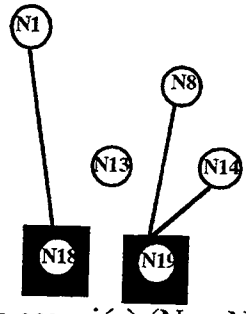


Figure II.16 : Sous-graphe associé à (N13, N18, N19) après réinjection

Dans le cas de l'exemple de la figure II.16 la fonction coût locale après réinjection et minimisation est:

$C_{après} = C_{N13} + C_{N18} + C_{N19}$

Dans le cas où le gain associé à cette réinjection est négatif ou nul, le processus de cette réinjection est annulé et les noeuds Ni seront examinés, un par un, de la même façon que les noeuds isolés (voir a-).

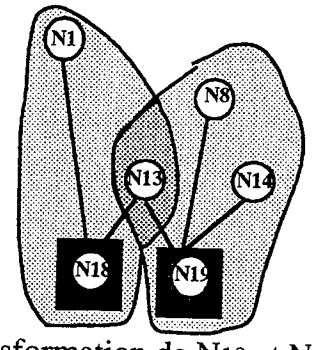


Figure II.17 : Transformation de N18 et N19 en noeuds isolés

II.6.5. Algorithme général: Réinjection et minimisation concurrentes

Le principe de l'approche proposée est fondé sur l'amélioration itérative de la FCP du graphe hiérarchique réduit, en itérant à chaque fois qu'une nouvelle passe apporte une amélioration supplémentaire. Avant d'amorcer une nouvelle passe les marquages "visité" des noeuds du graphe sont effacés et l'étiquetage hiérarchique est mis à jour. L'algorithme général appliqué à un graphe hiérarchique réduit est le suivant:

```
Réinjection (G)
{
    Étiquetage_hiérarchique (G)
    CTotale = Σ_{N∈G} CN                    /* CTotale est la FCP de G          */
    Tant que (CTotale diminue) faire
    {
     Identification_candidat (G)
     Gi = Premier_regroupement (G)     /* Gi sous-graphe de G; i = {1, ..., p} */
     Pour i ∈ {1, ..., p} faire
     {
          Gi,j = Deuxième_regroupement (G)
                                       /* Gi,j sous-graphe de Gi; j = {1, ..., q} */
          Pour j ∈ {1, ..., q} faire
          {
           Si Gi,j = noeud_isolé alors
           {
                Réinjection_noeud_isolé (Gi,j)
           }
           Si Gi,j = graphe_support_intersectant alors
           {
                Réinjection_graphe_support_intersectant (Gi,j)
           }
          }
     }
    }
    Effacer_marquage (G)
    Mise_à_jour_étiquetage_hiérarchique (G)
}
```

II.7. Expertise des résultats

II.7.1. Amélioration en terme de fonction coût prédite (FCP) par la méthode proposée de réinjection

Les tests sont appliqués sur un ensemble représentatif d'exemples qui proviennent directement d'une description VHDL. Ces tests sont destinés à montrer la capacité de notre approche à éliminer les redondances booléennes qui proviennent d'une description de haut niveau (VHDL) et l'optimiser en terme de la fonction coût prédite qui est une borne inférieure du nombre de cellules sur une technologie donnée. Les tests sont faits pour deux granularités différentes (∞,3) et (4,4), le nombre de passes est limité à 2. Pour Gr[Cell] = (4,4) on évaluera ensuite l'amélioration en terme de la FCP si on augmente le nombre de passes à 4. Cette évaluation permettra d'analyser la convergence de méthode proposée.

Name	IPC	IML	IINodes	NbPass	FPC	FML	FINodes	Cpu	MaxMem	G.FCP	G.noeuds
alpha1pa	340	6	194	1	337	6	113	6,3	477532	1	42
crc_ck1	255	12	125	1	260	9	72	4,46	422420	-2	42
jps	1097	6	263	1	1091	4	83	152	1699184	1	68
pnpisa	1113	6	322	1	817	4	63	113	3135752	27	80
typetop	261	12	79	1	118	3	13	7,72	414992	55	84
bus_ctrl	557	10	243	1	418	5	74	16,7	754952	25	70
crossbar	2438	4	1237	1	1236	2	9	140	4001164	49	99
k2	1407	2	182	0	947	1	0	346	1227248	33	100
switch	841	7	221	1	1006	3	55	293	2029036	-20	75
upc_comp	619	10	366	1	180	5	15	10,2	864040	71	96
centilli	653	13	98	1	684	9	77	19,5	1208824	-5	21
drinknik	104	17	57	1	69	6	18	1,43	190368	34	68
lab2a	21	4	8	1	18	2	4	0,18	76356	14	50
teiler16	111	6	35	1	90	2	3	1,06	307432	19	91
wgt8	198	2	7	0	205	2	7	99,8	1078860	-4	0
heinz	65	4	12	1	40	2	1	0,39	212964	38	92
p3_quant	139	5	64	1	129	3	27	7,74	383724	7	58
teletext	262	8	79	1	252	7	28	11,2	762996	4	65
countz	169	22	109	1	89	5	21	2,72	265276	47	81
ifsm	58	4	21	1	36	2	2	0,61	141808	38	90
peak	151	3	8	1	146	2	3	1,12	427332	3	63
topsch	319	13	90	1	227	6	11	3,58	592328	29	88
Moyenne										**21**	**69**

Tableau II.1 : Résultats pour Gr[Cell] = (1,∞)
le nombre de passes est limité à 2

Légende

IPC : Valeur initiale du FCP
IML : Degré hiérarchique initiale
IINodes : Nombre de noeuds dans le graphe hiérarchique réduit initiale
NbPass + 1 : Nombre de passes de la réinjection
FPC : Valeur finale du FCP après réinjection
FML : Degré hiérarchique final après réinjection
FINodes : Nombre de noeuds après réinjection
Cpu : Temps de calcul
MaxMem : Espace mémoire
G.FCP : % de gain de la FCP
G.noeuds : % de gain en nombre de noeuds du graphe hiérarchique réduit

Gr[Cell] = (1,∞) signifie que la cellule est saturée avec un monôme et ne dépend pas du nombre de variables, ce cas correspond exactement au critère classique de la minimisation logique. Dans le tableau II.1, nous remarquons une amélioration importante en terme du nombre de monômes. Cela signifie que **la technique de réinjection proposée peut être appliquée en minimiseur logique qui tient**

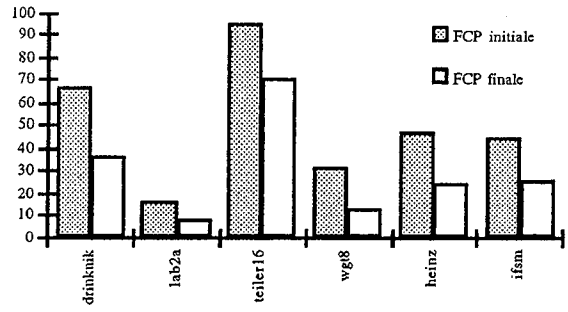
compte du partage de la logique dans le graphe hiérarchique réduit, ce qui n'est pas possible avec un minimiseur logique classique.

Name	IPC	IML	IINodes	NbPass	FPC	FML	FINodes	Cpu	MaxMem	G.FCP	G.n
alpha1pa	202	6	194	1	125	4	0	0,33	193468	38	100
crc_ck1	156	12	125	1	86	7	17	403	9881760	45	86
jps	748	6	263	1	706	4	192	213	1377432	6	27
pnpisa	1173	6	322	1	618	4	57	138	3184000	47	82
typetop	158	12	79	1	104	7	17	15,7	414992	34	78
bus_ctrl	421	10	243	1	276	5	69	24,9	944636	34	72
crossbar	2269	4	1237	1	889	2	9	157	4007448	61	99
k2	1033	2	182	1	925	2	178	8477	1471384	10	2
switch	773	7	221	1	509	4	82	376	1967352	34	63
upc_comp	366	10	366	1	144	3	32	20,7	866164	61	91
centilli	600	13	98	1	552	7	57	34,7	1291024	8	42
drinknik	67	17	57	1	36	4	9	8,84	302376	46	84
lab2a	16	4	8	1	8	1	0	0,11	78348	50	100
teiler16	95	6	35	1	71	2	11	1,44	316792	25	69
wgt8	31	2	7	0	12	1	0	606	5106164	61	100
com21	1024	7	276	1	758	4	32	42,6	2715892	26	88
heinz	47	4	12	1	24	2	1	0,4	213916	49	92
p3_quant	191	5	64	1	100	3	15	90,7	449496	48	77
teletext	207	8	79	1	164	4	24	45,3	1133516	21	70
countz	137	22	109	1	39	5	12	13,3	424628	72	89
ifsm	45	4	21	1	25	1	0	0,55	132304	44	100
peak	122	3	8	0	116	1	0	0,89	430296	5	100
topsch	238	13	90	1	189	5	18	3,76	576860	21	80
Moyenne										**38**	**81**

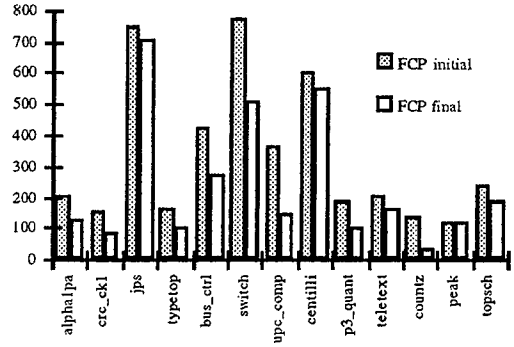
Tableau II.2 : Résultats pour Gr[Cell] = (∞,3)
le nombre de passes est limité à 2

Légende

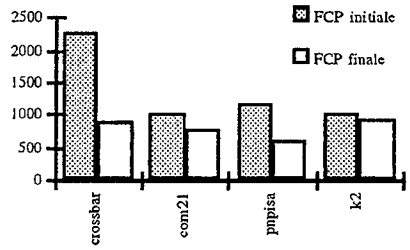
IPC : Valeur initiale du FCP
IML : Degré hiérarchique initiale
IINodes : Nombre de noeuds dans le graphe hiérarchique réduit initiale
NbPass + 1 : Nombre de passes de la réinjection
FPC : Valeur finale du FCP après réinjection
FML : Degré hiérarchique final après réinjection
FINodes : Nombre de noeuds après réinjection
Cpu : Temps de calcul
MaxMem : Espace mémoire
G.FCP : % de gain de la FCP
G.n : % de gain en nombre de noeuds du graphe hiérarchique réduit



Graphe II.1 : Amélioration de la FCP pour les exemples de petite complexité (**Gain moyen = 46%**)



Graphe II.2 : Amélioration de la FCP pour les exemples de moyenne complexité (**Gain moyen = 33%**)



Graphe II.3 : Amélioration de la FCP pour les exemples de grande complexité (**Gain moyen = 36%**)

Les graphes II.1, II.2, II.3 présentent le gain obtenu par la réinjection proposée avec une granularité de (∞, 3). Le gain est calculé par rapport au design initial en terme de la FCP. Il est de 46% pour les exemples de petite complexité (<100), 33% pour les exemples de moyenne complexité (100<X<1000) et de 36% pour les exemples de grande complexité (>1000).

Name	IPC	IML	IINodes	NbPass	FPC	FML	FINodes	Cpu	MaxMem	G.FCP	G.n
alpha1pa	202	6	194	1	124	5	73	38,8	773404	39	62
crc_ckl	154	12	125	1	114	9	43	17,1	620412	26	66
jps	647	6	263	1	599	4	160	224	1E+06	7	39
pnpisa	982	6	322	1	562	4	63	111	3E+06	43	80
typetop	160	12	79	1	83	3	12	21,3	446608	48	85
bus_ctrl	385	10	243	1	242	5	56	21,6	782048	37	77
crossbar	1914	4	1237	1	682	2	10	144	4E+06	64	99
k2	793	2	182	1	288	1	0	7212	1E+06	64	100
switch	604	7	221	1	467	4	97	264	2E+06	23	56
upc_comp	358	10	366	1	108	3	13	13,3	902408	70	96
centilli	537	13	98	1	488	6	45	24,2	1E+06	9	54
drinknik	60	17	57	1	30	5	11	3,59	226200	50	81
lab2a	14	4	8	1	5	1	0	0,14	77484	64	100
teiler16	89	6	35	1	56	2	3	1,36	313276	37	91
wgt8	55	2	7	0	55	2	7	98	1E+06	0	0
com21	918	7	276	1	719	3	48	41,5	3E+06	22	83
heinz	46	4	12	1	22	2	1	0,4	213916	52	92
p3_quant	157	5	64	1	55	3	7	125	886812	65	89
teletext	180	8	79	1	140	4	26	15,1	872152	22	67
countz	112	22	109	1	33	5	15	4,94	293340	71	86
ifsm	39	4	21	1	22	2	1	0,63	141476	44	95
peak	86	3	8	1	83	2	5	0,92	425484	3	38
topsch	184	13	90	1	138	4	19	2,75	581856	25	79
Moyenne										**40**	**78**

Tableau II.3 : Résultats pour Gr[Cell] = (4,4)
le nombre de passes est limité à 2

Légende

IPC : Valeur initiale du FCP
IML : Degré hiérarchique initiale
IINodes : Nombre de noeuds dans le graphe hiérarchique réduit initiale
NbPass + 1 : Nombre de passes de la réinjection
FPC : Valeur finale du FCP après réinjection
FML : Degré hiérarchique final après réinjection
FINodes : Nombre de noeuds après réinjection
Cpu : Temps de calcul
MaxMem : Espace mémoire
G.FCP : % de gain de la FCP
G.n : % de gain du nombre de noeuds du graphe hiérarchique réduit

II.7.2. Étude de la convergence de la méthode de réinjection proposée

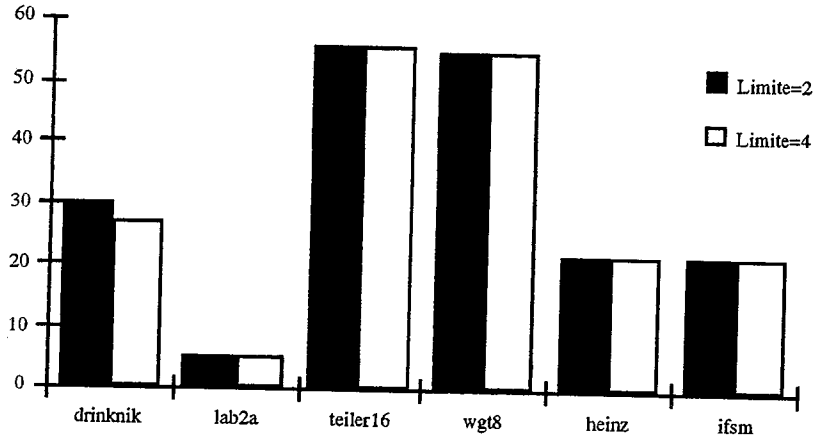
Cette section a pour but d'étudier la convergence de la méthode proposée. Nous avons pris comme granularité Gr[Cell] = (4,4), tenant ainsi compte à la fois du nombre de monômes et de variables.

Name	IPC	IML	IINodes	NbPass	FPC	FML	FINodes	Cpu	MaxMem	G.FCP	G.n
alpha1pa	202	6	194	2	124	5	73	88,25	792328	39	62
crc_ck1	154	12	125	2	114	9	43	33,59	631656	26	66
jps	647	6	263	3	594	4	155	545,59	1426116	8	41
pnpisa	982	6	322	3	556	4	58	223,74	3144176	43	82
typetop	160	12	79	3	77	3	8	58,54	504144	52	90
bus_ctrl	385	10	243	3	232	5	43	47,32	813292	40	82
crossbar	1914	4	1237	2	682	2	10	156,79	4004528	64	99
k2	793	2	182	1	288	1	0	8042,6	1345220	64	100
switch	604	7	221	2	467	4	97	362,73	2038768	23	56
upc_comp	358	10	366	2	108	3	13	17	903444	70	96
centilli	537	13	98	2	488	6	45	36,66	1302476	9	54
drinknik	60	17	57	3	27	2	5	6,04	226948	55	91
lab2a	14	4	8	1	5	1	0	0,15	77800	64	100
teiler16	89	6	35	2	56	2	3	1,93	313600	37	91
wgt8	55	2	7	0	55	2	7	113,85	1079200	0	0
com21	918	7	276	3	651	3	12	71,54	2850796	29	96
heinz	46	4	12	2	22	2	1	0,52	214160	52	92
p3_quant	157	5	64	3	43	3	4	222,39	886808	73	94
teletext	180	8	79	3	137	4	22	39,23	908704	24	72
countz	112	22	109	3	27	4	10	11,86	314496	76	91
ifsm	39	4	21	2	22	2	1	0,78	141808	44	95
peak	86	3	8	1	83	2	5	1,2	425432	3	38
topsch	184	13	90	2	138	4	19	5,43	582668	25	79
Moyenne										**42**	**80**

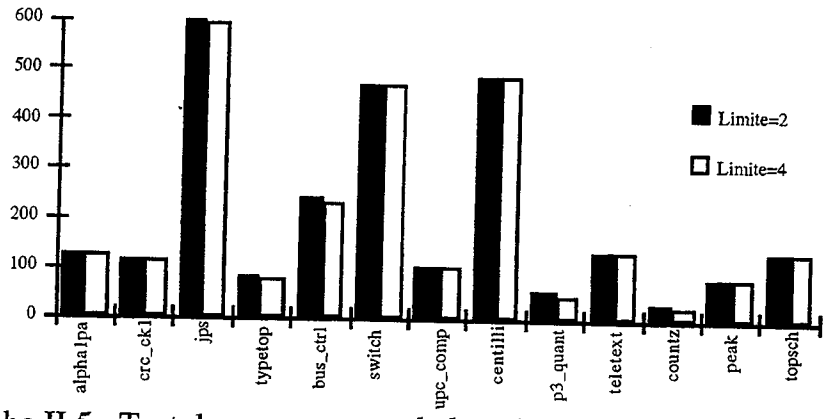
Tableau II.4 : Résultats pour Gr[Cell] = (4,4)
le nombre de passes est limité à 4

Légende

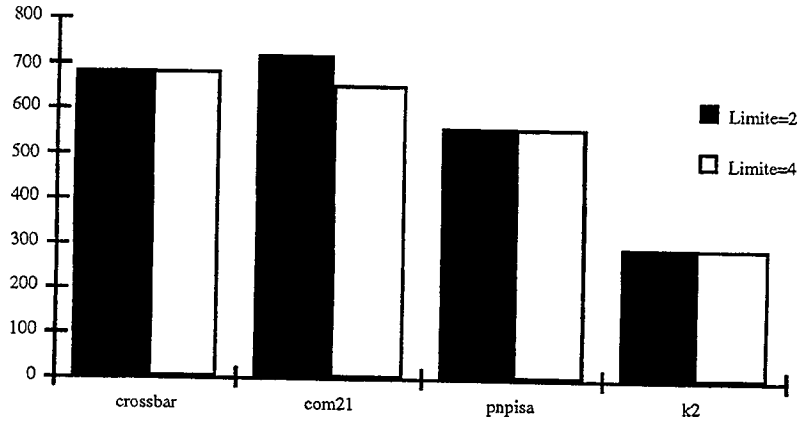
IPC : Valeur initiale du FCP
IML : Degré hiérarchique initiale
IINodes : Nombre de noeuds dans le graphe hiérarchique réduit initiale
NbPass + 1 : Nombre de passes de la réinjection
FPC : Valeur finale du FCP après réinjection
FML : Degré hiérarchique final après réinjection
FINodes : Nombre de noeuds après réinjection
Cpu : Temps de calcul
MaxMem : Espace mémoire
G.FCP : % de gain de la FCP
G.n : % de gain en nombre de noeuds du graphe hiérarchique réduit



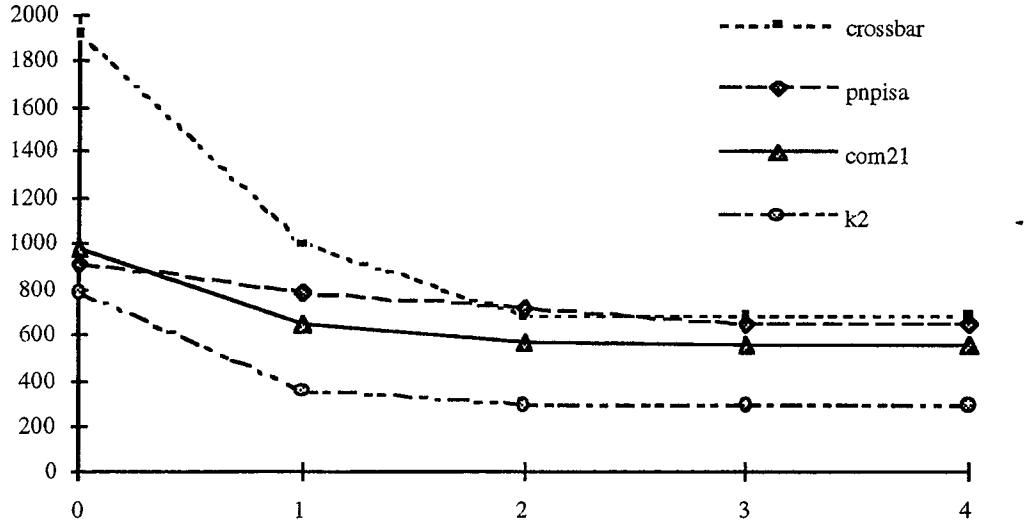
Graphe II.4 : Test de convergence de la méthode pour les exemples de petite complexité



Graphe II.5 : Test de convergence de la méthode pour les exemples de moyenne complexité



Graphe II.6.a : Test de convergence de la méthode pour les exemples de grande complexité



Graphe II.6.b : Test de convergence de la méthode pour les exemples de grande complexité

Les résultats sur les graphes II.4, II.5 et II.6 (a et b) montrent que l'augmentation de la limite du nombre de passes de 2 à 4 offre une amélioration générale des résultats. Cependant, le gain reste faible ou nul, il gravite autour de 5% dans les meilleurs des cas. On situe donc la convergence de la méthode entre 2 et 4 passes, pour la suite des expériences on prend la valeur 2. Ce jeu de test a permis de mettre en évidence la rapidité de la convergence de la méthode proposée en terme de nombre de passes.

II.7.3. Impact du filtrage des candidats diviseurs sur la réinjection

Dans cette section, les tests sont faits sur un jeu d'exemples qui n'a pas de sous-fonctions (Gros contrôleurs). En effet, toutes les sorties primaires dépendent directement et seulement des entrées primaires. Dans ce cas, la factorisation est appelée pour créer le partage de logique et diminuer la taille des expressions des fonctions booléennes. La réinjection dans ce cas corrige l'effet de la factorisation dans le but de diminuer la valeur de la FCP des expressions booléennes. Nous avons testé l'impact du filtrage des candidats diviseurs sur ce processus d'optimisation. Nous avons choisi la granularité de la cellule Gr[Cell] = (4,4).

Name	IPC	IML	IINodes	NbPass	FPC	FML	FINodes	Cpu	MaxMem	G.FCP	G.n
elmos1	16	2	3	1	15	2	2	0,18	152516	6	33
ctrl1	32	3	6	1	29	2	2	0,23	259528	9	67
lvq	143	4	42	1	112	2	3	10,05	1708588	22	93
scf	189	5	55	1	158	3	9	45,55	4513796	16	84
imec1	292	5	94	1	201	2	11	90,26	5443668	31	88
imec6	268	5	77	1	243	3	28	19,05	5719516	9	64
heyti2	415	5	143	1	283	3	10	376,29	6744788	32	93
imec8	431	5	134	1	282	2	4	402,98	7166192	35	97
hyeti1	599	5	211	1	408	4	13	746,04	9591256	32	94
ps_ocr	486	6	166	1	400	3	25	859,22	11075288	18	85
imec4	712	6	256	1	428	3	8	730,62	11438420	40	97
imec7	1076	6	381	1	708	4	24	2848,5	19415956	34	94
seqbis	849	7	303	1	648	4	38	4158	41303928	24	87
Moyenne										**24**	**83**

Tableau II.5 : Résultats pour Gr[Cell] = (4,4)
avec filtrage des candidats diviseurs

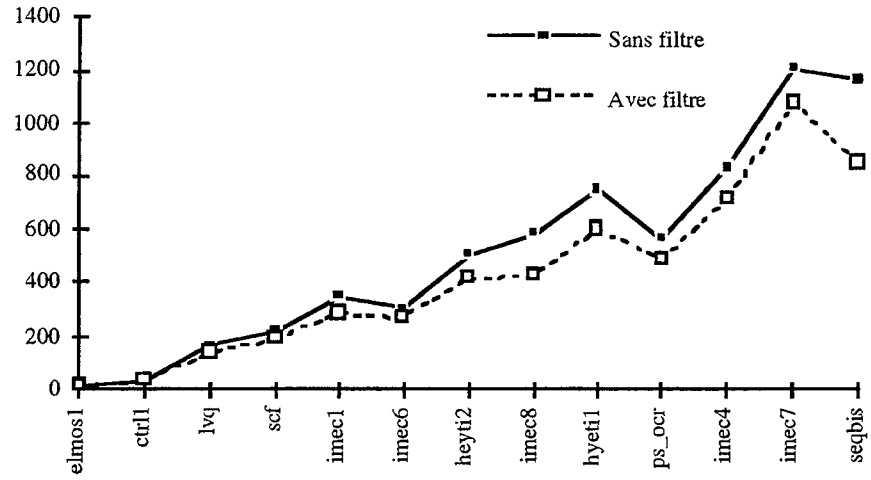
Name	IPC	IML	IINodes	NbPass	FPC	FML	FINodes	Cpu	MaxMem	G.FCP	G.n
elmos1	16	2	3	1	15	2	2	0,19	139728	6	33
ctrl1	32	3	6	1	29	2	2	0,24	252484	9	67
lvq	166	5	51	1	112	2	1	20,82	1708528	33	98
scf	221	5	71	1	165	3	20	85,24	4513796	25	72
imec1	346	6	129	1	197	3	5	167,5	5444316	43	96
imec6	306	7	110	1	247	4	35	41,06	5719772	19	68
heyti2	505	7	185	1	289	3	5	828,32	6745180	43	97
imec8	582	6	191	1	288	1	0	784,35	7165896	51	100
hyeti1	753	8	285	1	429	3	25	1835,02	9591256	43	91
ps_ocr	568	7	204	1	411	4	46	1878,74	11075288	28	77
imec4	826	10	306	1	520	5	62	1933,32	11438420	37	80
imec7	1203	7	451	1	851	5	103	6986,06	19415316	29	77
seqbis	1163	7	407	1	705	3	32	14756,94	41303928	39	92
Moyenne										**31**	**81**

Tableau II.6 : Résultats pour Gr[Cell] = (4,4)
sans filtrage des candidats diviseurs

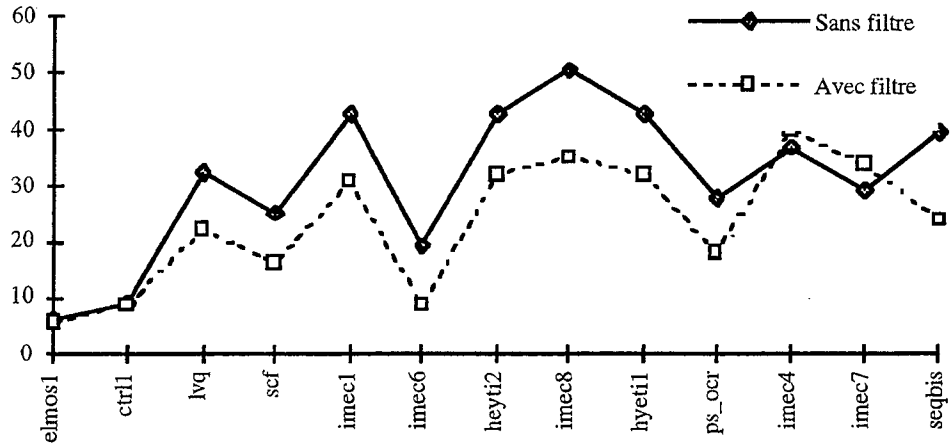
Légende

IPC : Valeur initiale du FCP
IML : Degré hiérarchique initiale
IINodes : Nombre de noeuds dans le graphe hiérarchique réduit initiale
NbPass + 1 : Nombre de passes de la réinjection
FPC : Valeur finale du FCP après réinjection
FML : Degré hiérarchique final après réinjection
FINodes : Nombre de noeuds après réinjection
Cpu : Temps de calcul

MaxMem : Espace mémoire
G.FCP : % de gain de la FCP
G.n : % de gain en nombre de noeuds du graphe hiérarchique réduit

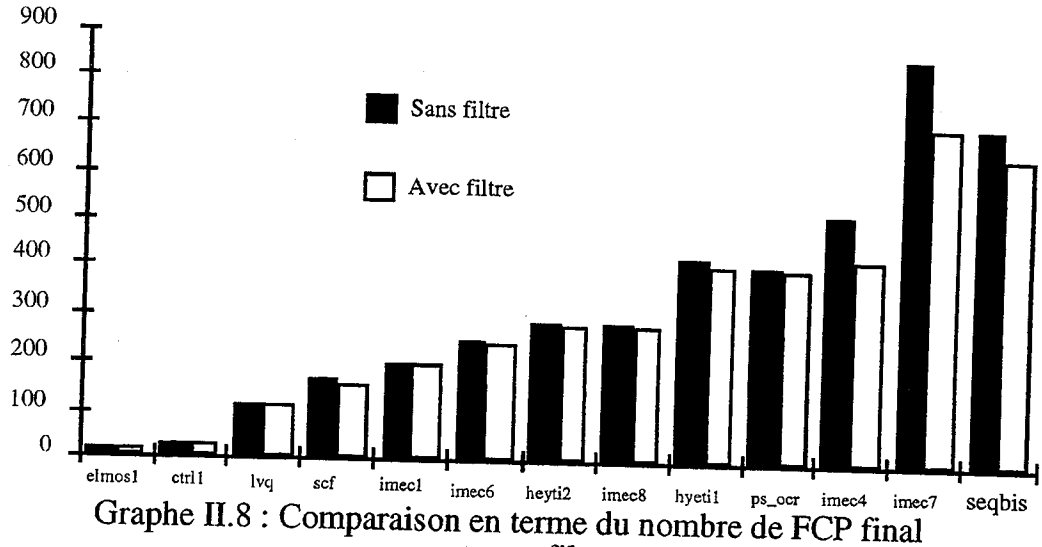


Graphe II.7.a : Comparaison entre la FCP initiale avec et sans filtrage



Graphe II.7.b : Gains apportés par rapport à la solution initiale avec et sans filtrage

La FCP initiale est celle calculée après la factorisation et juste avant la réinjection. Le filtrage des candidats diviseurs a permis d'avoir une FCP initiale inférieure à celle obtenue sans filtrage. Le graphe II.7.b montre le gain réalisé par notre approche de réinjection sur une FCP initiale avec ou sans filtrage. On remarque que le gain réalisé sur une expression booléenne initiale résultat du filtrage est inférieur en général (sauf pour le cas de l'exemple "imec4") au gain sans filtrage. Ce résultat prouve la capacité de notre approche de réinjection à corriger l'effet de la factorisation.



Graphe II.8 : Comparaison en terme du nombre de FCP final avec et sans filtrage

Le résultat final obtenu montre que l'avance du filtrage en terme de FCP a été raccourcie mais pas complètement rattrapée. Ce résultat confirme la nécessité de l'étape de filtrage des candidats diviseurs pendant la factorisation, d'autant plus que la consommation mémoire et le temps de calcul sont bien réduits.

II.8. Conclusion

L'approche proposée de la réinjection est très efficace en terme de la FCP (fonction de coût prédite). Une amélioration intéressante du design initial est obtenue pour plusieurs granularités. En outre, la plupart des solutions optimisées sont atteintes rapidement, au bout de 2 passes.

Les résultats expérimentaux montrent également la capacité de l'approche proposée à corriger l'effet de la factorisation.

Finalement, il s'avère nécessaire de mettre en place une phase de filtrage des candidats diviseurs pendant la factorisation; en plus de résultats meilleurs le filtrage permet de réduire le temps de calcul et l'espace mémoire.

Chapitre III. Application : Synthèse sur réseaux programmables

III.1. Les familles des réseaux programmables PLD/CPLD

III.1.1. Introduction aux réseaux programmables

Les réseaux programmables font partie de la famille des circuits à la demande (ASIC: Application Specific Integrated Circuits) et de la sous-famille des circuits " semi-custom ", au même titre que les réseaux prédiffusés (Gate Arrays).

Ils présentent par rapport à ces derniers de nombreux avantages. L'investissement initial comprenant l'achat du composant et le dispositif nécessaire à la programmation est relativement faible et cela leur confère un coût très avantageux pour de petites séries. Le développement d'une application est simple et s'effectue en un temps très réduit, de quelques heures à quelques jours. Leur programmation est très aisée, elle s'effectue directement chez le concepteur, ce qui permet de s'affranchir des problèmes de fonderie.

Récemment leur densité d'intégration et leur fréquence de fonctionnement sans cesse croissante les a placés en concurrents directs des réseaux prédiffusés. Dès leur apparition, les réseaux programmables comptaient environ 500 portes, ce qui leur a vite permis d'absorber puis de dépasser les composants réalisés en logic " MSI-SSI ".

Par la suite, le passage à des techniques nouvelles de programmation (EEPROM, antifusible PLICE/ViaLink, CMOS,...) les a conduits à des densités capables d'intégrer entre 1000 et 50.000 portes, à des temps de traversée inférieurs à 5 ns. Ils se trouvent désormais en concurrence avec 80 % des réalisations en circuits intégrés classiques, en particulier celles qui requièrent moins de 20.000 portes et fonctionnent à des fréquences de moins de 30 MHZ [Van96].
Ces progrès permettent de réaliser, à l'aide des réseaux programmables, des applications de complexité croissante, de plus en plus rapides, tout en conservant

une grande souplesse, en particulier la possibilité de reprogrammer un composant, donc de réaliser un nouveau circuit, sans remettre en question le routage d'une carte.

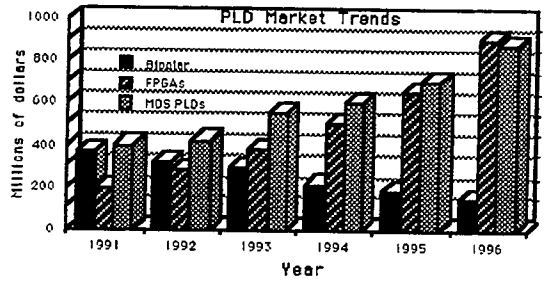


Figure III.1 : Croissance du marché des réseaux programmables

La figure III.1 montre la croissance du marché des réseaux programmables.
On peut distinguer deux grandes familles de réseaux programmables suivant la primitive booléenne sous-jacente à leur organisation topologique:
- **Les réseaux à forte granularité** organisés topologiquement en deux plans " ET " et " OU ", implantant des fonctions booléennes écrites sous forme de somme de monômes. Ils sont appelés: les PLD "Programmables Logic Devices", et les CPLD "Complex Programmables Logic Devices".
- **Les réseaux de cellules**, offrant comme primitives des cellules proches des portes de base des " Gate Array ". Ils sont appelés FPGA "Field Programmable Gate Array".

Nous nous intéressons particulièrement aux réseaux à forte granularité pour lesquels nous proposons un flot de synthèse fondé sur les méthodes d'optimisation booléenne exposées dans les chapitres précédents (minimisation symbolique, calcul de De Morgan, factorisation, réinjection) à partir d'une description VHDL, ou d'une façon générale d'un ensemble d'équations booléennes.

III.1.2. La famille des PLD, CPLD

Cette famille de réseaux programmables est caractérisée par deux plans logiques. Le premier, appelé "ET" ou étage "Produit", réalise un "ET logique" entre les variables d'entrée pour former des monômes. Le second, appelé étage "OU" ou étage "Somme", réalise un "OU logique" entre les monômes pour former des fonctions booléennes. Soulignons que toute variable est présentée à la fois sous sa forme normale (a) et sous sa forme complémentée ($\bar{a}$). À la sortie

des deux plans, nous obtenons des fonctions booléennes sous forme de somme de monômes.

Cette implantation des fonctions booléennes, sous forme de somme de monômes, nécessite la minimisation logique pour chaque fonction, qui doit être globale si la structure du réseau autorise le partage des monômes.
La représentation la plus classique des réseaux à deux plans est illustrée par la figure III.2

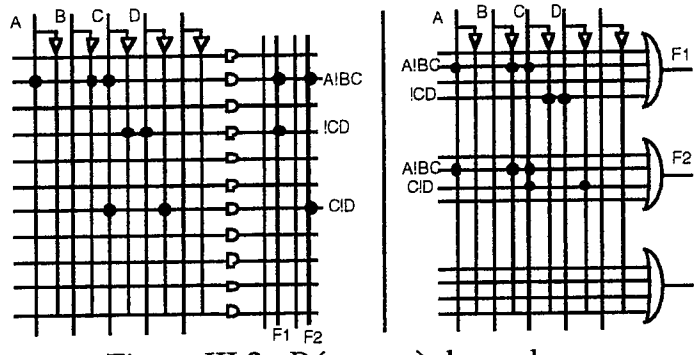


Figure III.2 : Réseaux à deux plans

La programmation est réalisée à l'intersection des lignes verticales et horizontales de chaque plan, soit à l'aide de fusibles que l'on brûle dans le cas des réseaux non-reprogrammables, soit à base de cellules " EPROM ", une par croisement, que l'on charge pour assurer le contact dans le cas des réseaux reprogrammables.
Dans la figure III.2, une connexion est caractérisée par un point " • ", placé à l'intersection des lignes connectées. Dans le plan " ET ", cela signifie que la variable associée à la ligne verticale est affectée au monôme correspondant à la ligne horizontale, et dans le plan "OU", que le monôme associé à la ligne horizontale est affecté à la sortie correspondant à la ligne verticale. On trouve parfois une croix "X" à la place du point. Elle signifie que l'intersection considérée peut être programmée par le concepteur, par opposition à celles qui ont déjà été programmées par le fabriquant.
La possibilité ou l'impossibilité, pour deux fonctions différentes, d'utiliser le même monôme, nous amène à distinguer:

- **Les fonctions ne partageant pas les monômes.**
- **Les fonctions partageant les monômes.**

III.1.2.1 Fonctions ne partageant pas les monômes

Les réseaux programmables à deux plans, sans partage de monômes, sont organisés autour d'un bloc qui réalise les deux plans de logique et le traitement éventuel de la fonction booléenne avant sa sortie. On distingue les réseaux monoblocs et les réseaux multiblocs.

a) *Réseaux monoblocs ne partageant pas les monômes*

Un des boîtiers les plus connus de ce type est PAL22V10.

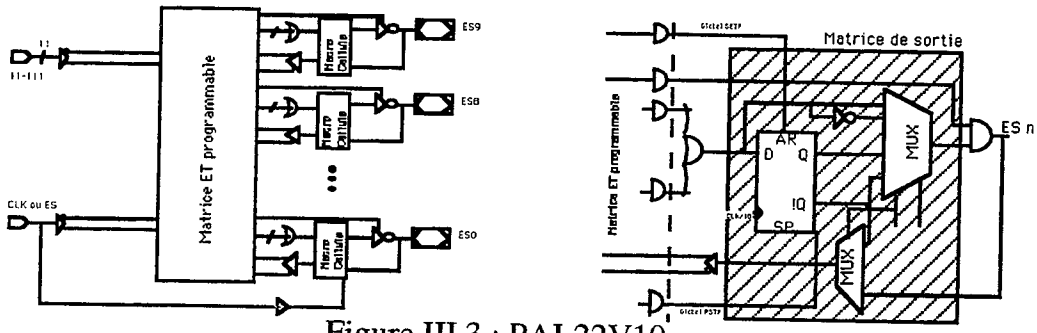


Figure III.3 : PAL22V10

Tous les monômes peuvent utiliser toutes les variables d'entrée. Chaque sortie a un nombre fixé de monômes qui lui sont propres. Ils ne peuvent être utilisés par aucune autre sortie. Une cellule de sortie ou macrocellule représente la sortie de la fonction booléenne générée à l'étage " OU ".

b) Réseaux multiblocs ne partageant pas les monômes

La structure de ces réseaux est constituée par un ensemble de blocs de même nature que celui des réseaux monoblocs, et par un canal de connexion assurant la distribution des entrées et la liaison entre les blocs. Cette structure est présentée dans la figure III.4

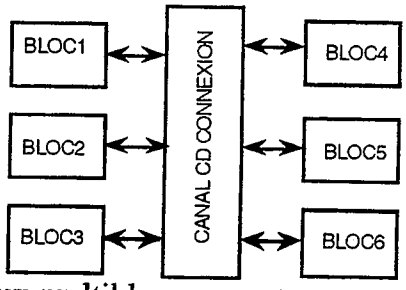


Figure III.4 : Réseaux multiblocs ne partageant pas des monômes

Chaque bloc a une structure très proche de celle des réseaux monoblocs, mais il se singularise par deux aspects. Le premier a trait à la taille du plan "ET".

Elle est identique pour tous les blocs et chaque entrée est dupliquée pour réaliser sa forme normale et sa forme complémentée. Le second concerne les monômes qui ne sont toujours attribués qu'une seule fois.
Chaque sortie dispose d'une possibilité supplémentaire de rebouclage. Cela permet d'utiliser la sortie comme une entrée, si la macrocellule associée à cette sortie génère un signal interne.
Le canal de connexion, pour sa part, assure la distribution des signaux d'entrée aux blocs:

- Soit directement de l'extérieur, à partir des broches dédiées aux entrées ou des broches de sortie utilisées comme entrées.
- Soit de l'intérieur, à partir du rebouclage d'une sortie de macrocellule.

Cette dernière possibilité autorise le rebouclage d'un signal dans un même bloc ou dans un autre bloc à travers le canal.

III.1.2.1 Fonctions partageant les monômes

Les réseaux à deux plans partageant certains monômes et possédant des monômes propres sont appelés monômes privés. Ils sont organisés autour d'un bloc qui réalise les deux plans de logique et le traitement de la fonction booléenne avant sa sortie. On distingue les réseaux monoblocs et les réseaux multiblocs.

L'organisation en blocs de ce type de réseaux est très proche de celle des réseaux qui ne partagent pas les monômes, et la distinction entre monobloc et multibloc est faite pour les même raisons. En effet, le réseau multibloc permet de gérer simultanément des monômes propres à une fonction et des monômes partageables entre toutes les fonctions d'un même bloc.

a) Réseaux monoblocs partageant les monômes

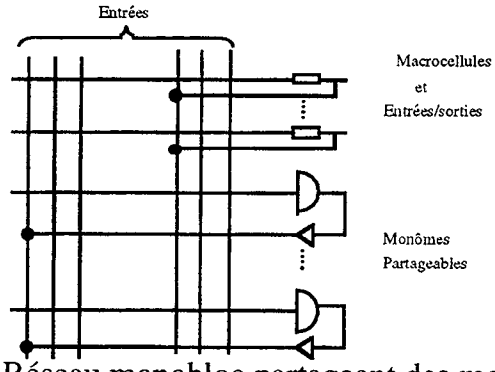


Figure III.5 Réseau monobloc partageant des monômes

On distingue un plan "ET" programmable, ainsi qu'un plan "OU" formé de deux types d'éléments, des macrocellules pouvant réaliser des fonctions de n monômes et des monômes partageables. Ces derniers rebouclent le plan "ET" pour former des variables internes qui peuvent alors être utilisées comme variables d'entrées par toutes les macrocellules.

b) Réseaux multiblocs partageant les monômes

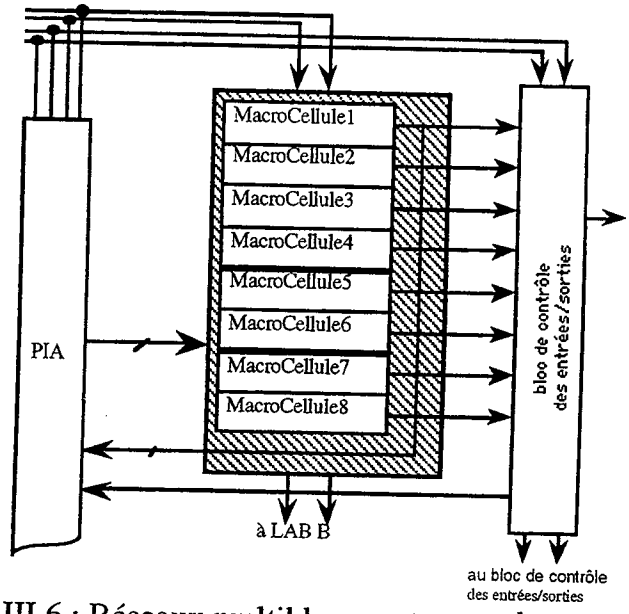


Figure III.6 : Réseaux multiblocs partageant les monômes

La structure de ces réseaux est constituée d'un ensemble de blocs et d'un canal de connexion assurant la distribution des entrées et la liaison entre blocs. Cette structure multibloc est illustrée dans la figure III.6; les réseaux multiblocs ne partagent pas les monômes entre blocs.

Exemple:

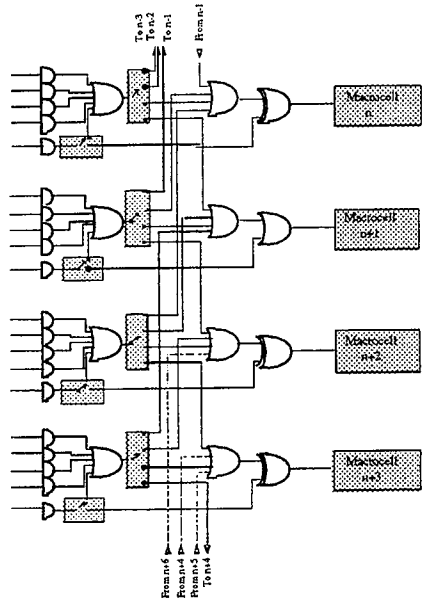


Figure III.7 : Un bloc de macrocellules

III.2. Module de synthèse combinatoire sur CPLD

Dans ce paragraphe, nous proposons un module de synthèse combinatoire sur la technologie CPLD. L'objectif de ce module est d'optimiser les équations booléennes, provenant d'une "netlist" de réseaux de portes, ou du résultat de codage de contrôleur [De Micheli84] [Duff91], ou de la sortie d'un compilateur d'une description de haut niveau (VHDL, Verilog) [Cébélieu95].
Après la présentation de ce module, nous commentons les résultats expérimentaux obtenus sur deux cibles technologiques CPLD: La famille MACH de la société AMD [AMD95], et la cellule XC9500 de la société XILINX [Xilinx96]. Le critère d'optimisation est le nombre de macrocellules.

III.2.1. Granularité d'une macrocellule

Pour déterminer si une sous-fonction est mappable sur une macrocellule, les caractéristiques physiques d'une macrocellule sont décrites par ce qu'on appelle le calibre "template" ou la granularité. Cette granularité est caractérisée par les paramètres suivants:
- **k1** : Le nombre maximal de variables d'entrées accepté par le bloc logique de la macrocellule (26 pour les MACH230 d'AMD, 36 pour les XC9500 de Xilinx).

- **k2** : Le nombre maximal de monômes qu'on peut implanter dans une macrocellule (4 pour les MACH230 [AMD95] et 5 pour les XC9500 [Xilinx96]). Ce nombre peut être augmenté "logic allocator" à 16 pour les MACH230 [AMD95] et à 20 pour les XC9500 [Xilinx96].

III.2.2. Présentation générale du module de synthèse logique

Cette optimisation consiste à recalibrer les expressions booléennes pour une implantation sur la technologie CPLD. L'objectif est donc de minimiser le nombre de macrocellules après décomposition technologique. Ce module comprend la minimisation des expressions booléennes par la technique symbolique implantée et exposée au premier chapitre, permettant le choix entre la forme normale et la forme complémentée (à l'aide du calcul de DeMorgan). L'extraction des sous-fonctions partagées est effectuée par des techniques de factorisation algébrique. L'optimisation surfacique et la correction de la granularité se font à l'aide de la méthode de réinjection proposée dans le cadre de cette thèse.

Cette optimisation supporte:
i) Les expressions des valeurs indéfinies "don't care".
ii) La préservation de certains signaux requis, de telle façon que le signal reste identifiable à la netlist finale résultat de ce module.
iii) Les signaux ou les modules à ne pas optimiser. Les modules les plus couramment préservés sont les fonctions arithmétiques (+, -, *), ou relationnelles (>, <, = , ...etc.).

Ce module comprend trois sous-modules:

i) **Le réducteur de complexité**
ii) **La correction de la granularité**
iii) **La décomposition en macrocellules**

III.2.3. Réducteur de complexité

Cette première étape consiste à trouver la base irrédondante optimale pour les fonctions et les sous-fonctions, en terme de nombre de monômes. Pour chaque expression polynomiale de fonctions et sous-fonctions, le ROBDD associé est construit. L'ordre initial des variables est trouvé à l'aide d'une heuristique de recherche du meilleur ordre; cette heuristique est fondée sur l'assignement

dynamique [Minato90] des variables. Le ROBDD obtenu est réduit par la méthode d'échange de variables par fenêtre glissante [Besson96]. À partir du ROBDD amélioré nous cherchons les monômes canoniques et les monômes premiers. Le résultat est leur représentation implicite sous forme de MBDD ([Minato93] et [Coudert93a]). Cette représentation compacte permet la manipulation d'un nombre exponentiel de monômes avec un nombre polynomial de noeuds.

Le calcul du noyau cyclique est effectué avec des processus de réductions fondés sur l'essentialité et la dominance. L'application de ces règles produit les monômes essentiels. Si le noyau cyclique est vide, la base irrédondante minimale est obtenue par les monômes premiers (appelés essentiels) qui couvrent les monômes essentiels. Sinon, elle est obtenue par l'union des monômes premiers et la solution du noyau cyclique. La résolution du noyau cyclique se fait par l'approche exacte de sélection et séparation "Branch & Bound", ou - si la taille du noyau cyclique est de grande complexité - une solution approchée par l'heuristique fondée sur l'extension des monômes à couvrir.

La sélection de la forme normale ou complémentée choisit celle qui sera la plus adaptée pour les étapes suivantes. La technique de minimisation est utilisée pour effectuer la complémentation symbolique de De Morgan. Elle consiste à introduire un inverseur à la racine du ROBDD qui est propagé jusqu'à ses feuilles.

III.2.4. Correction de la granularité

Ce sous-module comprend la factorisation algébrique qui génère des sous-fonctions partagées réduisant la complexité en terme de nombre de fonctions coûts (CF Chapitre II). Elle contient deux étapes majeures: la génération des candidats diviseurs et la substitution algébrique. Les candidats diviseurs peuvent être des noyaux locaux ou globaux et des monômes communs. La substitution algébrique est la division algébrique par le noyau et son complément.
Pour effectuer la sélection des candidats diviseurs, une approche gloutonne est appliquée, fondée sur la fonction coût en nombre de littéraux. Une suite de seuils de gains permet d'alterner le choix des deux types de diviseurs (le noyau et les monômes). Le filtrage des candidats diviseurs est testé, il élimine les candidats diviseurs ayant une taille (en terme de fonction coût) strictement inférieure à 1.

Une phase qui corrige (les équations booléennes déjà synthétisées, par exemple après factorisation) ou optimise (les équations booléennes proviennent directement d'une description de concepteur) la granularité des sous-fonctions consiste à faire appel à une méthode originale de réinjection partielle qu'on a proposée dans le cadre de cette thèse. Cette phase recalibre la taille des expressions booléennes pour saturer les macrocellules. Cette approche tient compte du partage de la logique et de la complexité des noeuds. Elle s'effectue en plusieurs passes et finit par converger vers une solution optimisée. Chaque passe comprend un parcours du graphe pour détecter des réductions locales de la fonction coût liée à la granularité technologique, et ceci par réinjection élémentaire. Au cours de ce parcours, les noeuds ne sont pas examinés simultanément, une stratégie de regroupement "clustering" est appliquée.

III.2.5. Décomposition en macrocellules [ASYL95]

Ce sous-module est la seule phase technologie dépendante. Il est d'autant plus rapide que les fonctions booléennes sont calibrées. La décomposition des fonctions qui ne sont pas mappables sur une macrocellule est de deux sortes:
- Si le nombre de variables est plus grand que k_1, la fonction est décomposée en sous-fonctions, où le nombre de variables de chacune est inférieur à k.
- Si le nombre de monômes est plus grand que k_2, la fonction est décomposée en sous-fonctions, où le nombre de monômes est inférieur à k_2.

III.2.6. Le flot général

Le flot général décrivant le module d'optimisation combinatoire est donné dans la figure suivante:

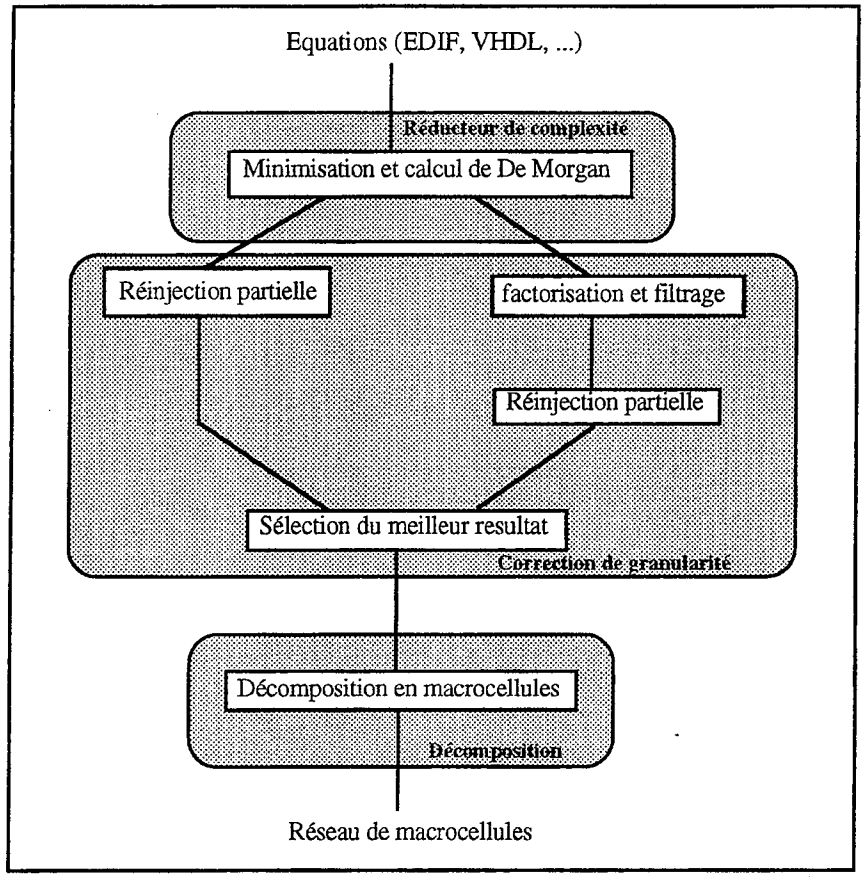


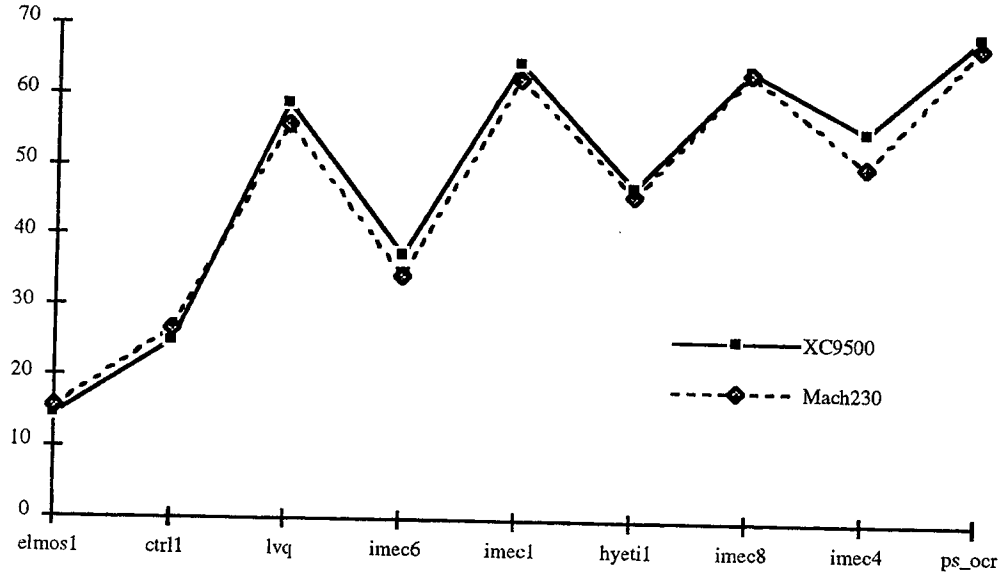
Figure III.8 : Flot de synthèse du module d'optimisation sur CPLD

III.3. Résultats expérimentaux

III.3.1. Exemples de contrôleurs

	MACH230 (AMD)			XC9500 (XILINX)		
Name	MC_Init	MC_Opt	%Gain	MC_Init	MC_Opt	%Gain
elmos1	14	12	14	13	11	15
ctrl1	32	24	25	30	22	27
lvq	197	81	59	163	72	56
imec6	469	293	38	383	251	34
imec1	542	191	65	431	162	62
hyeti1	672	354	47	540	292	46
imec8	799	287	64	636	232	64
imec4	906	405	55	722	357	51
ps_ocr	1098	338	69	880	284	68

Tableau III.1 Résultats en terme de macrocellules après décomposition sur XC9500 et MACH230



Graphe III.1 : Courbe de gain en macrocellules après décomposition par rapport à la description initiale

Les résultats sont obtenus sur un jeu de tests "Gros contrôleurs", Pour la cellule XC9500, on a appliqué la réinjection pour la valeur de la granularité Gr[XC9500] = (5, ∞) (5 monômes et 36 variables) et la granularité de la cellule

MACH230 est de Gr[MACH230] = (4, ∞) (4 monômes et 26 variables). Le gain moyen pour la cellule MACH230 est de **48%**, tandis que pour XC9500 est de **47%**

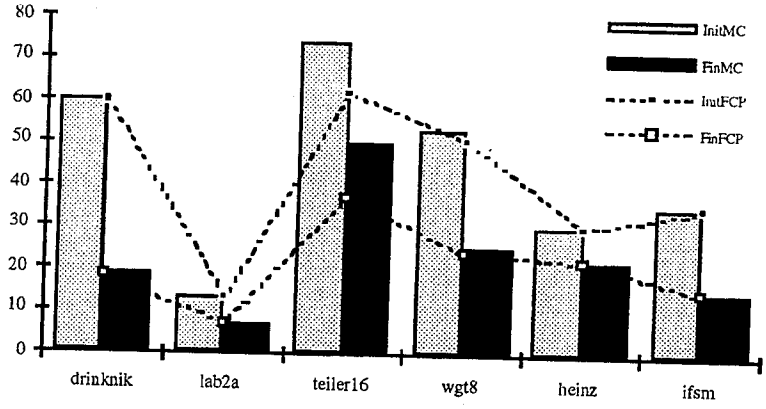
III.3.1. Exemples à partir d'une description VHDL

Name	InitFCP	InitMC	InitSf	FinFCP	FinMC	FinSf	G_MC	G_FCP
alpha1pa	202	202	194	124	138	73	32	39
heinz	30	30	12	22	22	1	27	27
switch	448	657	221	385	437	44	33	14
ifsm	35	35	21	15	15	1	57	57
teiler16	62	74	35	37	50	2	32	40
centilli	459	491	98	418	440	37	10	9
jps	524	672	263	280	312	80	54	47
teletext	150	172	79	110	137	19	20	27
com21	796	847	276	558	604	26	29	30
k2	505	939	182	256	336	0	64	49
countz	111	112	109	21	24	5	79	81
lab2a	13	13	8	7	7	0	46	46
typetop	109	129	79	40	54	5	58	63
crc_ck1	140	145	125	101	122	49	16	28
p3_quant	78	118	64	37	40	21	66	53
upc_comp	341	343	366	52	54	13	84	85
crossbar	1461	1508	1237	444	551	4	63	70
peak	85	85	8	77	77	0	-9	9
wgt8	51	53	7	24	25	12	53	53
drinknik	60	60	8	18	19	5	68	70
pnpisa	692	1036	322	403	567	44	45	42
bus_ctrl	348	360	243	175	210	37	42	50
topsch	182	184	90	101	108	8	41	45
Moyenne							45	45

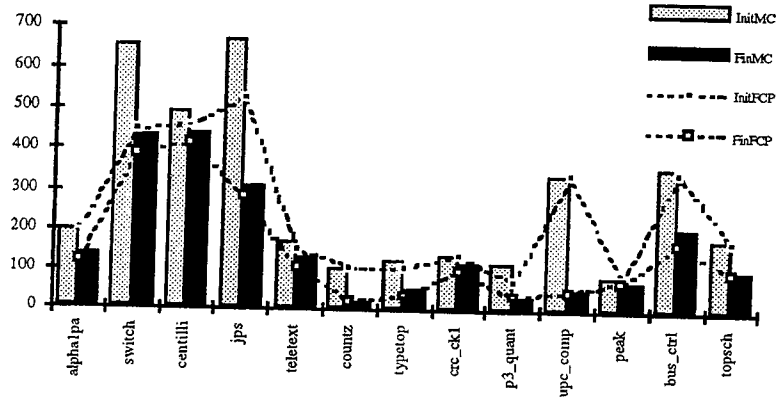
Tableau III.2 Résultats sur les exemples VHDL en terme de macrocellules après décomposition sur MACH230

Légende

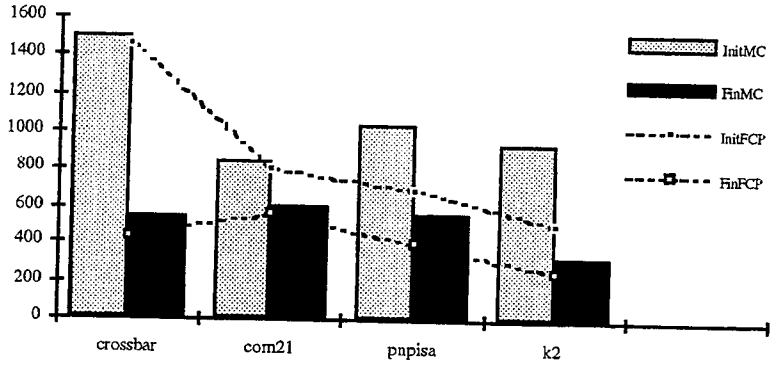
InitFPC : Valeur initiale du FCP
InitMC : Nombre initial de macrocellules après décomposition
InitSf : Nombre initial de sous-fonctions
FinFCP : Valeur finale du FCP
FinMC : Nombre final de macrocellules après décomposition
FinSf : Nombre final de sous-fonctions
G_FCP : Gain en % de la FCP
G_MC : Gain en % du nombre de macrocellules après décomposition



Graphe III.2 : Amélioration de la FCP et du nombre de macrocellules sur MACH230 pour les petits exemples



Graphe III.3 : Amélioration de la FCP et du nombre de macrocellules sur MACH230 pour les exemples de moyenne complexité



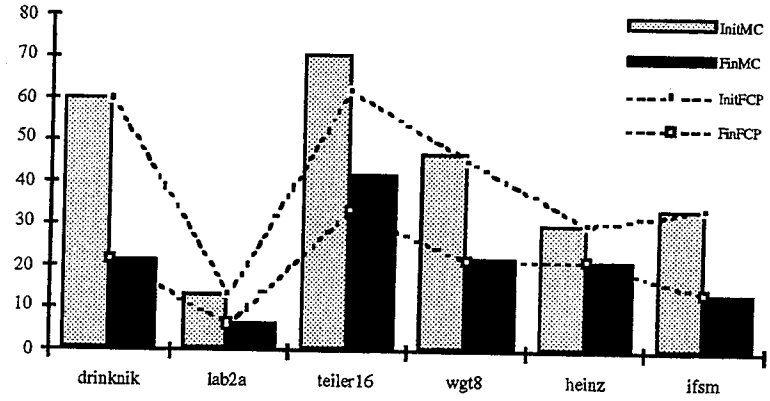
Graphe III.4 : Amélioration de la FCP et du nombre de macrocellules sur MACH230 pour les exemples de grande complexité

Name	InitFCP	InitMC	InitSf	FinFCP	FinMC	FinSf	G_MC	G_FCP
alpha1pa	202	202	194	123	132	67	35	39
heinz	30	30	12	21	21	0	30	30
switch	445	573	221	281	296	14	48	37
bus_ctrl	341	343	243	161	167	42	51	53
ifsm	34	34	21	14	14	0	59	59
teiler16	62	71	35	33	42	2	41	47
centilli	459	472	98	404	418	29	11	12
jps	480	556	263	256	276	80	50	47
teletext	149	161	79	107	123	21	24	28
com21	789	802	276	544	553	13	31	31
k2	446	756	182	211	255	0	66	53
topsch	179	180	90	103	104	5	42	42
countz	111	112	109	14	17	0	85	87
lab2a	13	13	8	6	6	0	54	54
typetop	103	105	79	35	40	5	62	66
crc_ck1	139	145	125	92	104	50	28	34
p3_quant	75	101	64	37	39	23	61	51
upc_comp	339	340	366	47	48	11	86	86
crossbar	1449	1476	1237	420	468	4	68	71
peak	85	85	8	77	77	0	9	9
wgt8	45	47	7	21	22	13	53	53
drinknik	60	60	57	21	21	5	65	65
pnpisa	681	908	322	392	498	56	45	42
							48,09	47,67

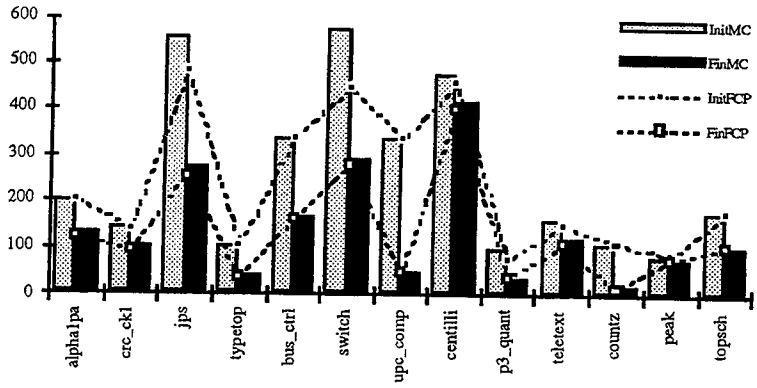
Tableau III.3 : Résultats sur les exemples VHDL en terme de macrocellules après décomposition sur XC9500

Légende

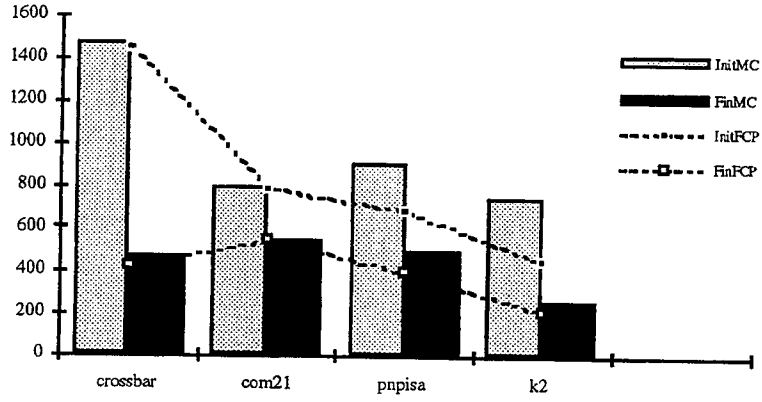
InitFPC : Valeur initiale du FCP
InitMC : Nombre initial de macrocellules après décomposition
InitSf : Nombre initial de sous fonctions
FinFCP : Valeur finale du FCP
FinMC : Nombre final de macrocellules après décomposition
FinSf : Nombre final de sous fonctions
G_FCP : Gain en % de la FCP
G_MC : Gain en % du nombre de macrocellules après décomposition



Graphe III.5 : Amélioration de la FCP et du nombre de macrocellules sur XC9500 pour les exemples de faible complexité



Graphe III.6 : Amélioration de la FCP et du nombre de macrocellules sur XC9500 pour les exemples de moyenne complexité



Graphe III.7 : Amélioration de la FCP et du nombre de macrocellules sur XC9500 pour les exemples de grande complexité

III.4. Conclusion

Le module d'optimisation proposé est très efficace sur la technologie CPLD. Le résultat final après décomposition technologique est corrélé avec la fonction coût prédite dans le graphe hiérarchique réduit. Ce module a permis de "corriger" la granularité, d'une part des expressions booléennes obtenues par la factorisation, d'autre part des équations obtenues par une description de haut niveau de type VHDL

Chapitre IV. Génération de macro-blocs

IV.1. Introduction

Les macro-blocs peuvent être classifiés en macros dures "hard macro", macros molles "soft macro" et en générateurs paramètrés.

Les macros dures sont des blocs à structure figée du point de vue physique ou topologique.
Les macros molles sont des blocs où la structure interne est fixée mais le placement et le routage restent flexibles; et ne seront effectués que lors du routage global du circuit; dans ce cas le bloc sera mêlé dans la netlist physique globale pour le placement et le routage global. Les macros molles sont généralement fournies par le fondeur pour des tailles fixes en nombre de bits (ex. 4, 8, 16 bits pour l'additionneur, multiplieur, comparateur etc).

Les générateurs paramétrés sont destinés à générer durant la synthèse des macros molles pour toutes les tailles possibles. LPM est la bibliothèque des générateurs paramétrés et un ensemble de modules qui constitue le standard de base.

Si l'usage des bibliothèques de base (cellules standards) est banal, l'extension à des bibliothèques de macroblocs ([Khalil95] et [Cébélieu95]) et à des générateurs de macro-blocs s'est accélérée. Afin de réduire le temps de synthèse des circuits complexes tout en obtenant les meilleurs résultats, on ne va utiliser que des parties déjà synthétisées et optimisées suivant les critères habituels (surface, vitesse). Ces parties pré-synthétisées appelées macros, sont stockées dans une bibliothèque et insérées dans les circuits suivant les besoins. Par exemple, des additionneurs ou des multiplieurs décrits au niveau de la saisie comportementale en langage de haut niveau (VHDL) seront directement pris dans la bibliothèque sans même repasser par une phase de synthèse logique (figure IV.1).

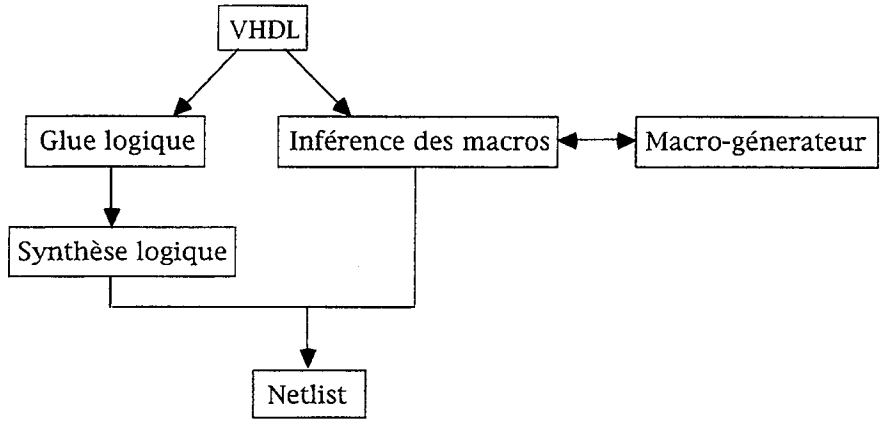


Figure IV.1: Flot de conception

Le macro-générateur paramétré a pour but de fournir une macro optimisée sur une technologie donnée au cours de la synthèse. Il va utiliser plusieurs paramètres. Tout d'abord le type de macro désiré, ensuite la taille voulue des macros en terme de nombre de bits, ainsi que les critères d'optimisation tels que surface ou vitesse. La tendance actuelle nous conduit à traiter les macros suivant des contraintes fixées par l'utilisateur et à trouver des compromis "trade-off" entre les critères habituels. Ces nouveaux besoins impliquent une complexification des macro-générateurs.

Nous nous attacherons ici à la macro-génération des additionneurs, qui constituent un élément essentiel de la macro-génération en général. En effet, beaucoup d'autres macros telles que les soustracteurs, les incrémenteurs, les compteurs, les multiplieurs sont dérivées de l'additionneur.

La littérature est riche en publications traitant de la synthèse des additionneurs. Sklansky proposa un algorithme [Sklansky60] dont le principe est de générer deux ensembles de sorties et de retenues, un ensemble au cas où la retenue est 1 et un second au cas ou la retenue est 0. Kelliher [Kelliher82] proposa un additionneur de même structure mais avec un ensemble d'équations différentes. Wei et Thomson [Wei85] modifièrent l'algorithme pour traiter le problème de la sortance qui croît exponentiellement dans l'additionneur de Sklansky. Brent et Kung [Brent82] proposèrent une structure qui limite la sortance mais qui est deux fois plus profonde que l'additionneur de Sklansky. L'algorithme de Kogge et Stone [Kogge93] limite la sortance tout en optimisant la profondeur de l'additionneur. Enfin, Han et Carlson [Han87] combinèrent les structures de Brent et Kung et de Kogge et Stone.

IV.2. Équations de l'additionneur

Le problème majeur de l'additionneur réside non pas dans le résultat bit à bit, mais dans la gestion et l'accélération du calcul de la retenue entrante. La retenue est par définition exprimée par des équations récurrentes. Celle-ci peut être exprimée et définie de différentes façons. Brent et Kung ([Brent82], [Koren93], [Guyot95], [Belrhiti95]) introduisirent en 1982 l'opérateur arithmétique Δ, destiné à modéliser son comportement quant à sa propagation et sa génération en fonction des variables d'entrée/sortie.

IV.2.1. Génération et Propagation

Considérons la somme de deux nombres $a=(a_0,a_1,...,a_{n-1})$ et $b=(b_0,b_1,...,b_{n-1})$ de tailles n. Les bits seront numérotés de 0 à n-1.

IV.2.1.1 Au niveau d'un bit

À chaque position i, la retenue suivante peut être:

- Générée, si l'égalité $g_i=a_i.b_i=1$ est vérifiée.
- Propagée, si l'égalité $p_i=a_i\oplus b_i=1$ est vérifiée.
- Annulée, si l'égalité $\overline{a}_i.\overline{b}_i=1$ est vérifiée.

Deux parmi ces 3 données seulement sont nécessaires pour décrire l'évolution de la retenue. On retient (p,g) pour la suite de ce chapitre.

IV.2.1.2 Au niveau d'une tranche de bits:

La génération de la retenue au niveau d'une tranche de bits (notée G_i^j) entre les bits i et j signifie qu'une retenue a été générée pour l'ensemble de la tranche, et propagée jusqu'à la position i.

Les équations récurrentes sont donc condensées pour calculer les termes de génération et de propagation de la retenue pour une tranche de bits:

$G_i^j=G_i^k+P_i^k.G_{k-1}^j$ avec $n-1\geq i\geq k\geq j\geq 0$.
$P_i^j=P_i^k.P_{k-1}^j$ avec $n-1\geq i\geq k\geq j\geq 0$.

Quant à la sortie, le vecteur résultat de l'addition, noté :
$(Out\_0,Out\_1,...,Out\_(n-1))$, il sera donné par les formules suivantes :

$Out\_0 = a_0 + b_0$

$Out\_i = G_i^0 \oplus a_i \oplus b_i$ pour $n-1 \geq i \geq 1$.

IV.2.2 Définition de La Cellule Δ

On notera:

- $PG_i^j = (P_i^j, G_i^j) = (P_i^k . P_{k-1}^j, G_i^k + P_i^k . G_{k-1}^j)$
- Δ est l'opérateur tel que : $PG_i^k \Delta PG_{k-1}^j = (P_i^k . P_{k-1}^j, G_i^k + P_i^k . G_{k-1}^j)$

Δ est un opérateur qui admet quatre entrées et deux sorties. Il est idempotent, associatif, mais non commutatif.

IV.2.2.1 Définition d'une Δ-tranche

Considérons une tranche de bits entre la position i et j (i < j), supposons que le calcul de la retenue est réalisé pour tous les bits entre i et j. On appelle cette série une Δ-tranche et on la note Δ[i,j].

Δ[i, j] = {Δ(i, i), Δ(i, i+1), Δ(i, i+2), ..., Δ(i, j)}

IV.2.2.2 Définition d'un module de concaténation

Considérons 2 Δtranches Δ[i, j] avec i < j; et Δ[j+1, k] avec j < k. On appellera module de concaténation associé à Δ[i, j] et Δ[j+1, k] qui sera noté:

Δ[i, j]&Δ[j+1, k]

le module ayant comme entrées les 2 Δtranches et réalisant en sortie un ensemble de : Δ[i, k].

Le graphe associé à ce module contient trois noeuds : le noeud de la concaténation Δ(i, j, k) représenté par un triplet (les deux bornes séparées par l'indice du bit de concaténation), les noeuds opérandes (les deux Δtranches: Δ[i,j] et Δ[j+1, k]) et le noeud sommet de la Δtranche:

Δ[i,k] = Δ[i, j]&Δ[j+1, k]

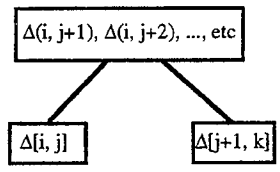


Figure IV.2 : Noeud du graphe
réalisant un module de concaténation

On appelle j **le bit de coupe du module de concaténation** associé à Δ(i, j) et Δ(j+1, k).

IV.3. Arbre de Coupe

Un arbre de coupe est un arbre binaire dont les noeuds et les feuilles sont des Δtranches. Un noeud $\Delta[i,j]$ a pour fils les deux Δtranches $\Delta[i,k]$ et $\Delta[k+1,j]$ avec $i \leq k \leq j-1$ si la Δtranche $\Delta[i,j]$ est calculée grâce au module de concaténation des Δtranches $\Delta[i,k]$ et $\Delta[k+1,j]$.
Les Δtranches $\Delta[i,j]$ associées aux feuilles correspondent au calcul en série des valeurs $\Delta(i, i) \ldots \Delta(i,j)$.

Exemple:
Les fonctions associées aux cellules Δdans les intervalles de bit $I_1 = [0, 2]$, $I_2=[3, 4]$, $I_3 = [5, 6]$, $I_4 = [7, 8]$, $I_5 = [9, 11]$, $I_6 = [12, 14]$, $I_7 = [15, 16]$ sont calculés en série (voir la figure IV.3). Les intervalles I_1 et I_2 sont calculés en parallèle puis composés à l'aide du module de concaténation C_1 ($C_1 = I_1 \Delta I_2$); $C_2 = I_3 \Delta I_4$; $C_3 = I_6 \Delta I_7$; $C_4 = C_1 \Delta C_2$; $C_5 = I_5 \Delta C_4$; $C_6 = C_5 \Delta C_3$. Ces opérations de sérialisation/parallélisation sont représentées dans le graphe ci-dessous.

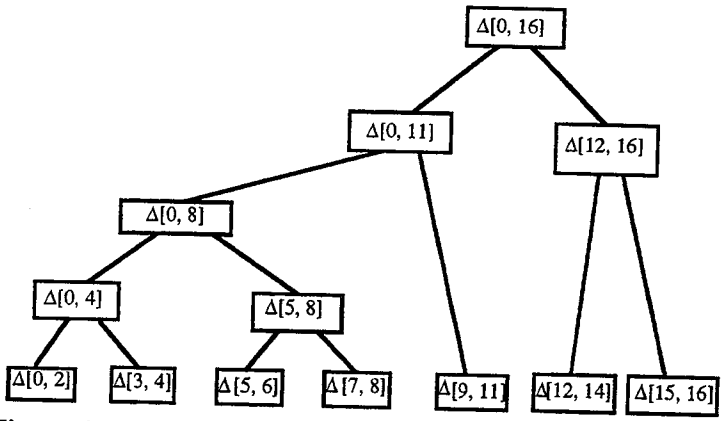


Figure IV.3 : Représentation d'un additionneur 16 bits

Ce graphe est appelé **arbre de coupe**.

On pourra ne représenter que les bits de coupe dans l'arbre, dépourvu de ses feuilles [Belrhiti95]. Il est aisé de reconstituer les Δtranches associées aux feuilles.

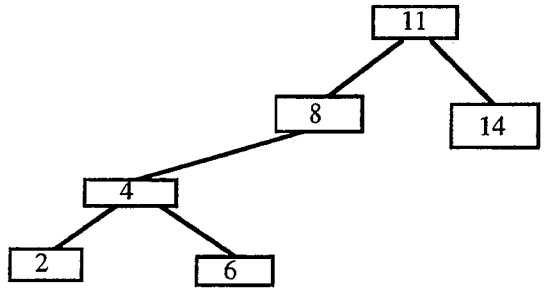


Figure IV.4: Représentation minimale d'un additionneur 16 bits

Notation associée:
Les arbres sont codés sur un modèle LISP par énumération des points de coupes parenthèsés hiérarchiquement. Pour un additionneur de taille n, dont le premier bit de coupe est k, on code l'arbre de coupe comme suit:

(k,(A1)(A2))

où A1 représente le codage de l'arbre de coupe des bits inférieurs à k et A2 le codage de l'arbre de coupe des bits supérieurs à k.
Pour l'additionneur de 16 bits, le codage est:

(11(8(4(2)(6)))14)

IV.4. Additionneurs classiques

Nous allons voir ici les additionneurs traditionnellement utilisés.
Le "carry ripple" est le plus simple et le plus petit (sa taille en terme de cellules Δ est de $n-1$) consiste à calculer chaque bit en fonction de la retenue issue du bit précédent. C'est l'additionneur le plus profond (en terme de cellules Δ) mais de faible sortance maximale. Son graphe de coupe est réduit à un seul module.

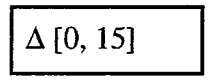


Figure IV.5 : Arbre de coupe
de l'additionneur 16 bits en "carry-ripple"

L'additionneur à deux niveaux de type "carry-select" est dérivé de l'additionneur précédent en réalisant une parallélisation par tranche des calculs de la retenue. Le graphe de coupe suivant représente un additionneur 32 bits avec le code LISP **(23(16(10(5(1)))))**.

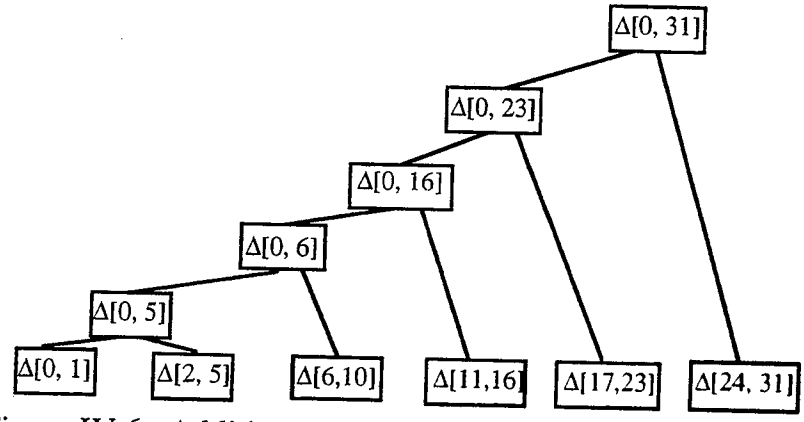


Figure IV.6 : Additionneur 32 bits en architecture carry-select

L'additionneur de Sklansky [Sklansky60] est construit récursivement en parallélisant successivement les calculs par une division symétrique des blocs. C'est l'additionneur le moins profond en terme de cellules Δ, mais sa sortance croît exponentiellement le long du chemin critique. Cette architecture peut donner de mauvais résultats si cet additionneur est réalisé sur une technologie sensible à la sortance (ou à la connectique). Il demeure néanmoins l'additionneur le moins profond. Son code en LISP associé est:
(16(24(20(18)(22))(28(26)(30)))(8(4(2)(6))(12(10)(14))))

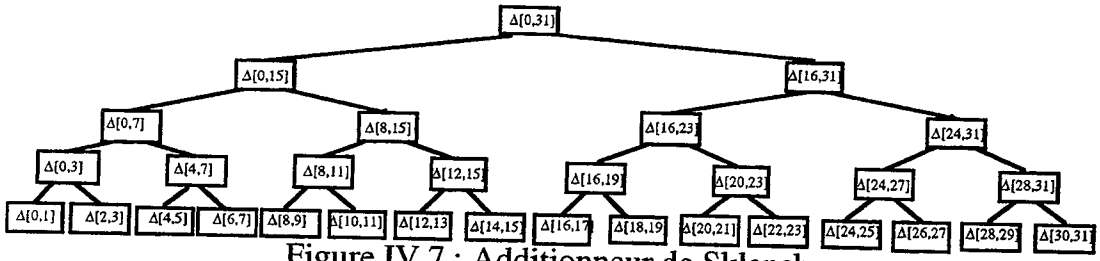


Figure IV.7 : Additionneur de Sklansky

L'additionneur de Brent et Kung [Brent82] est fondé sur la même structure que l'additionneur de Sklansky, mais la sortance est réduite grâce à une sérialisation locale des calculs. Cette sérialisation provoque une augmentation locale de la profondeur de l'additionneur.

IV.5. Classification des additionneurs

Un additionneur est dit ***construit par inclusion*** si et seulement si il peut être décrit par un arbre de Δtranches. En d'autres termes, cela signifie que lorsque les deux Δtranches Δ[i,j] et Δ[j+1,k] sont concaténées, tous les calculs des Δ(i,l), $i \leq l \leq k$ sont réalisés.

L'additionneur de Sklansky est construit par inclusion, mais l'additionneur de Brent&Kung ne l'est pas.

On s'attachera ici a l'étude exclusive des additionneurs construits par inclusion. Pour chaque additionneur, on définit sa ***caractéristique*** le vecteur constitué des trois éléments (**surface, profondeur et sortance maximale**).Toutes ces valeurs sont aisément calculées à partir des arbres de coupes [Belrhiti95] et données en terme des cellules Δ.
Les informations données au niveau des cellules Δ ne sont pas suffisantes pour une bonne estimation des caractéristiques de l'additionneur final, quelle que soit la technologie-cible choisie. Par exemple, l'additionneur de Sklansky est le moins profond en terme de cellules, mais ne donne pas forcément l'additionneur le plus rapide. Néanmoins la caractéristique pourra être utilisée comme filtre pour éliminer des architectures dont les performances apparaissent comme médiocres.

Type d'additionneur	Nombre de cellules Δ	Profondeur en cellules Δ	sortance maximale
Ripple	$n-1$	$n-1$	1
Carry-select	$\lceil 2.n-\sqrt{2.n} \rceil$	$\lceil \sqrt{2.n} \rceil$	$\lceil \sqrt{2.n} \rceil$
Von Neumann	$\left\lceil \frac{n}{2}.\log_2(n) \right\rceil$	$\log_2(n)$	$\left\lceil \frac{n}{2} \right\rceil$

Tableau IV.1 : Additionneurs classiques avec leurs caractéristiques

Dans le cas de l'additionneur 32 bits, nous obtenons les valeurs numériques suivantes:

Type d'additionneur	Nombre de cellules Δ	Profondeur en cellules Δ	Sortance maximale
Ripple	31	31	1
Carry-select	56	8	8
Von Neumann	80	5	16

Tableau IV.1 : Additionneur 32 bits classiques avec leurs caractéristiques

IV.6. Exploration de l'espace des solutions

A partir d'un arbre de coupe initial, on peut explorer l'espace des solutions par les trois opérations élémentaires suivantes:

- **Création d'un nouveau bit de coupe (1)**
- **Élimination d'un bit de coupe (2)**
- **Translation d'un sous-arbre de coupe (3)**

On Notera M(A) l'application aléatoire de l'une de ces transformations sur l'additionneur A.
La translation d'un sous-arbre de coupe est réalisée par translation de tous les bits de coupe contenus dans ce sous-arbre.

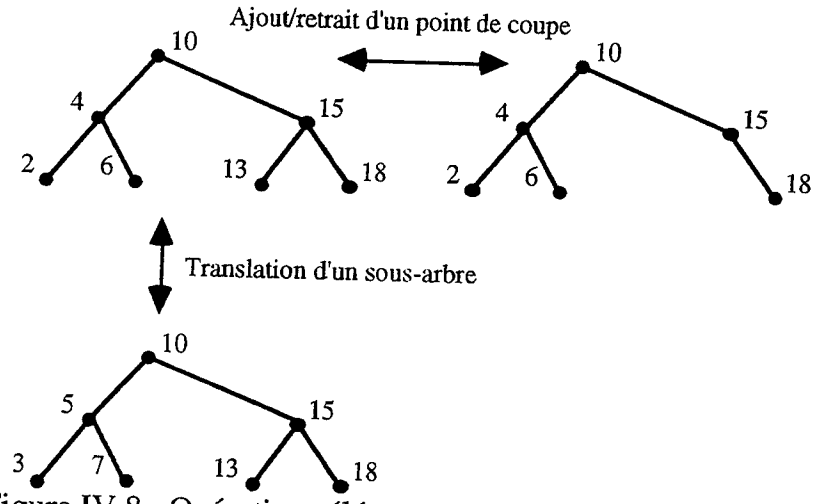


Figure IV.8 : Opérations élémentaires sur un arbre de coupe

IV.6.1. Nuage d'additionneurs

Le schéma suivant présente un ensemble d'additionneurs de l'espace des additionneurs construits par inclusion. L'abscisse représente le délai de l'additionneur et l'ordonnée sa surface. On considère ici un additionneur de taille 19 technologiquement décomposé sur la bibliothèque de cellules standards vsc370.

Ce nuage d'additionneurs est obtenu par un algorithme aléatoire qui génère un ensemble d'arbres de Δtranches en appliquant les trois opérations élémentaires décrites ci-dessus.

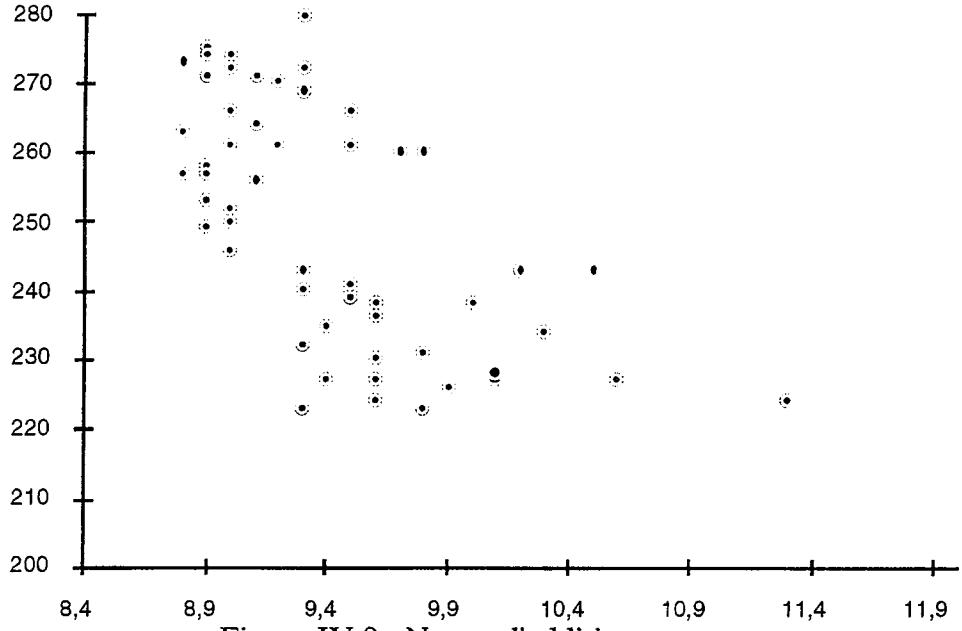


Figure IV.9 : Nuage d'additionneurs.

IV.6.2. Filtrage de l'espace de solutions

Pour un additionneur A, on notera C(A) = (surf(A), prof(A), sort(A)) sa caractéristique, où surf(A), prof(A) et sort(A) désignent respectivement la surface, la profondeur et la sortance maximale de l'additionneur A.

IV.6.2.1 Définition d'une relation d'ordre

La relation d'ordre partielle << est définie sur les additionneurs par:
$A_1 << A_2$ si et seulement si les trois conditions suivantes sont vérifiées:

- $surf(A_2) \leq surf(A_1)$
- $prof(A_2) \leq prof(A_1)$
- $sort(A_2) \leq sort(A_1)$

Un additionneur A_1 est dit moins performant qu'un additionneur A_2 si et seulement si : $A_1 << A_2$.

la caractéristique de l'architecture (surface, profondeur, sortance maximale) peut être utilisée comme un filtre permettant d'éliminer les structures dont les performances semblent médiocres.
Le filtre permet d'éliminer les solutions trivialement inintéressantes.

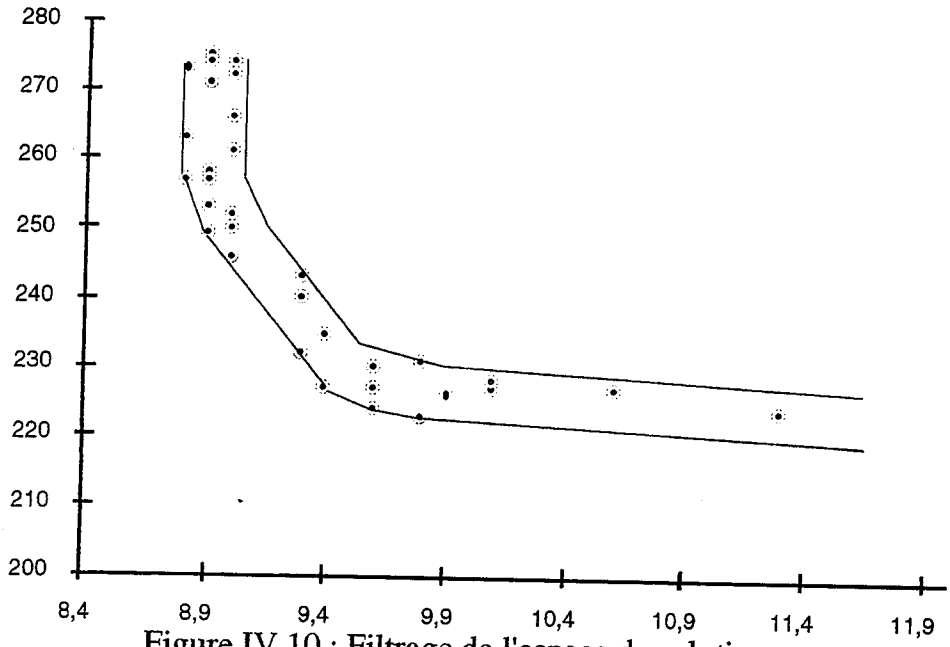


Figure IV.10 : Filtrage de l'espace de solutions

Le filtre permet de sélectionner une zone de solutions acceptables délimitée par les deux courbes décrites dans la figure ci-dessus.

IV.6.3. Optimisation par dérivation d'une solution acceptable

À partir d'une solution acceptable A, on dérive l'arbre de coupe par l'application de la fonction M(A). La relation d'ordre partielle << permet l'élimination des additionneurs dont les caractéristiques C(A) sont plus mauvaises que la caractéristique de l'additionneur de départ.
L'algorithme suivant est appliqué récursivement jusqu'à l'obtention d'une solution jugée meilleure. La qualité de la solution n'est pas seulement estimée par la caractéristique de l'additionneur, mais aussi après une phase de décomposition technologique. L'algorithme est basé sur une méthode Tabou [Gondran85] [Finke91] qui permet de ne pas revenir sur des solutions déjà testées. Sa complexité est exponentielle, mais l'usage du filtre dont le coût est linéaire et très faible rend utilisable l'algorithme.

Algorithme d'optimisation:

Soit A_1 une solution acceptable
modification aléatoire de A_1 : $A_2 = M(A_1)$
Si $A_2 << A_1$
    éliminer A_2
Sinon
    décomposition technologique de $A_2 \rightarrow A_{2_{dec}}$
    Si $A_{2_{dec}}$ est meilleur que $A_{1_{dec}}$ alors
        $A_1 \leftarrow A_2$
        $A_{1_{dec}} \leftarrow A_{2_{dec}}$
    Sinon
        éliminer A_2

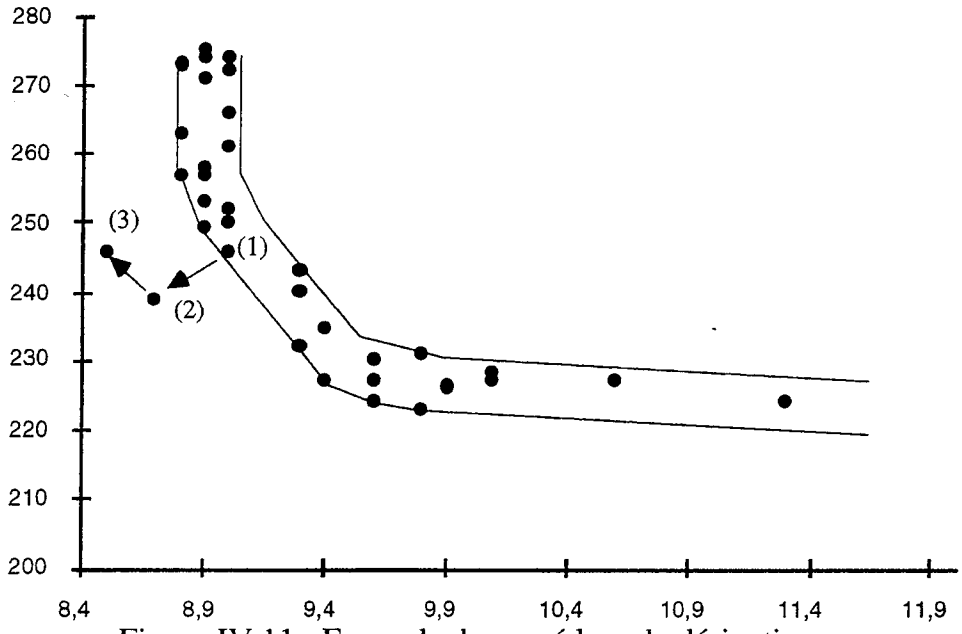


Figure IV.11 : Exemple de procédure de dérivation

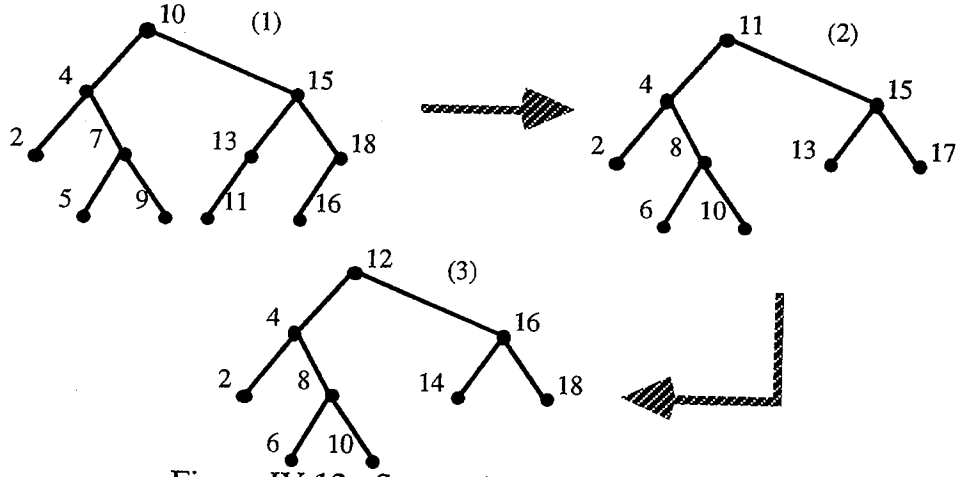


Figure IV.12 : Succession d'arbres de coupe

Ce schéma décrit une succession d'arbres obtenue par application de l'algorithme d'optimisation qui correspondent aux points marqués sur la figure IV.12. Pour les trois arbres, nous obtenons respectivement les trois caractéristiques suivantes:

$C(A_1)=(41, 5, 9)$, $C(A_2)=(40, 5, 8)$, et $C(A_3)=(40, 5, 8)$.

Les additionneurs correspondants possèdent les caractéristiques suivantes:

Additionneurs	Surface μ^2	délai (ns)
A1	246	8.9
A2	240	8.7
A3	246	8.5

Tableau IV.3 : Performance des additionneurs associée aux arbres de coupe

IV.6.4. Outil pour l'exploration de l'espace des solutions

Un outil général a été développé pour réaliser l'exploration de l'espace des solutions ainsi que l'optimisation par dérivation d'une solution acceptable. Les entrées de l'additionneur sont d'une part la taille et d'autre part un arbre de coupe ou un ensemble d'arbres de coupe décrits à l'aide du code LISP.

L'outil comprend deux phases majeures:

- La génération d'un additionneur ou d'un ensemble d'additionneurs à partir d'un arbre de coupe ou d'un ensemble d'arbres de coupe. Les additionneurs résultants sont synthétisés et estimés.

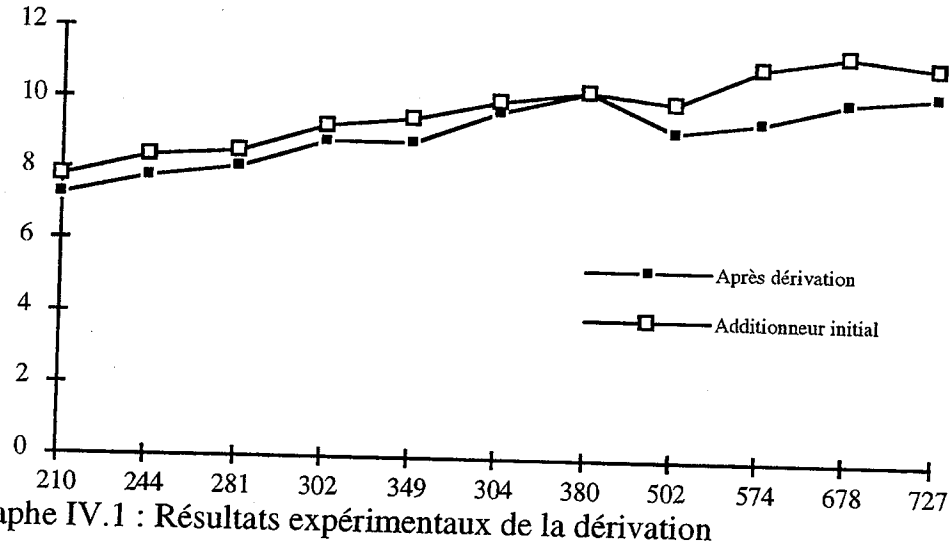
- Le processus de dérivation : Le point de départ peut être sélectionné automatiquement pour l'optimisation du délai ou choisi par l'utilisateur.

IV.7. Résultats expérimentaux

Le tableau suivant présente un ensemble d'additionneurs obtenus à l'aide de l'algorithme décrit précédemment. Les additionneurs sont technologiquement décomposés sur la bibliothèque de Cellules Standards vsc370. L'algorithme utilisé est basé sur la programmation dynamique et décrit dans [Touati90] et [Sakouti93]. La colonne "additionneur initial" donne le résultat en terme de surface (exprimée en μ^2) et le délai (exprimé en nano-secondes) de l'additionneur utilisé comme point de départ de la procédure de dérivation; la seconde colonne donne les caractéristiques de l'additionneur résultant de la dérivation .

Taille de l'additionneur	Additionneur initial		Additionneur dérivé.		Gain (%)	
	surface	délai	surface	délai	surface	délai
14	198	7,8	210	7,3	-6	6,4
16	258	8,4	244	7,8	5,4	7,1
18	281	8,5	281	8,1	0	4,7
20	283	9,3	302	8,8	-6,7	5,4
22	348	9,5	349	8,8	-0,3	7,4
24	310	10	304	9,7	0,2	3
26	381	10,3	-	-	-	-
28	514	10	502	9,2	2,3	8
30	567	11	574	9,5	-1,2	13,6
32	639	11,4	678	10,1	-6,1	11,4
34	788	11,1	727	10,3	7,7	7,2

Tableau IV.4 : Résultats expérimentaux de la dérivation



Graphe IV.1 : Résultats expérimentaux de la dérivation

Dans le cas de l'additionneur de taille 26, on remarque que la procédure de dérivation n'a pas trouvé de meilleur résultat. Néanmoins, la procédure permet sur cet ensemble d'additionneurs un gain moyen de 7,5% en terme de délai pour une surface moyenne équivalente.

IV.8. Conclusion

Nous avons proposé une méthode originale de l'exploration de l'espace des solutions fondée sur le filtrage des solutions générées. Pour une solution donnée, nous avons proposé une méthode de dérivation fondée sur l'approche Tabou pour améliorer d'avantage les performances. Des améliorations de notre méthode sont envisageables pour raffiner l'exploration de l'espace de solutions pour éviter le nombre de fois ou on est ramené à faire la décomposition technologique pour sélectionner une solution. Notre approche peut être efficacement appliquée sur la macro-génération sous contraintes temporelles.

Conclusion générale

Cette thèse avait plusieurs objectifs. Le premier était d'évaluer l'impact des outils de la "troisième génération" d'écriture de bases irrédondantes de fonctions booléennes à savoir les minimiseurs dits symboliques. Ces minimiseurs utilisent principalement des structures de données différentes. Pour ceci un tel outil a été implanté. Les conclusions sont celles attendues. Si les résultats en qualité (nombre de monômes et nombre de littéraux) ne sont pas sensiblement supérieurs à ceux de la génération précédente (heuristiques fondées sur une représentation classique), la complexité des exemples que l'on peut traiter est infiniment supérieure. De plus cette représentation seule permet de faire un choix efficace entre une fonction et son complément. Ces résultats sont extrêmement importants vu l'évolution technologique car les nouvelles techniques d'intégration intègrent des systèmes entiers et requièrent précisément une nouvelle génération d'outils. Cette thèse aura eu l'intérêt d'apporter une expertise sur ce point.

La deuxième contribution a consisté à étudier les problèmes de granularité des expressions factorisées. Là-aussi il s'agit d'une réflexion de "troisième génération". En effet les factorisations de la "deuxième génération" ont poussé des objectifs de minimisation de compte de littéraux. L'étude faite dans cette thèse concerne, la "correction" de cette granularité par des techniques de réinjection qui est communément trop fine et doit s'adapter à la technologie cible. Cette technique permet aussi de corriger la granularité des équations obtenues à la sortie de la compilation d'une description de haut niveau (VHDL, Verilog) car elle allie optimisation et réinjection. Il nous semble que cette thèse aura apporté une contribution importante sur ce problème de correction de granularité et que des expériences pratiques auront été concluantes.

Enfin une deuxième contribution, rappelle que la logique aléatoire ne constitue qu'une faible partie d'un circuit et que l'essentiel du circuit est composé de macroblocs. Une dernière contribution a donc montré que les méthodes de synthèse actuelle permettent d'explorer les compromis surface/temps de façon efficace et ceci est important pour les outils de synthèse sophistiqués fondés sur les contraintes temporelles [Hiol96].

En conclusion nous pensons que cette thèse aura contribué à faire une expertise détaillée et significative, puis à fournir d'importantes contributions aux outils de la troisième génération de synthèse logique. Il est primordial de constater que comme par le passé, ces outils évoluent poussés par l'évolution technologique et qu'il est surprenant de constater que des apports très significatifs sont accomplis à chaque fois.

Références bibliographiques

[Abouzeid92] P. Abouzeid, "Méthodes de Factorisation Algébrique Dédiées aux Circuits Intégrés Complexes", Thèse de Doctorat, INPG, Grenoble, France, 1993.

[Akers78] S.B. Akers, "Binary Decision Diagrams", IEEE Trans on Computers, Vol. 27, pp. 509-516, 1978.

[AMD95] AMD -Data Book- 1995.

[ASYL95] ASYL+ User's Manual - IST - 1995

[Bartee62] T.C. Bartee, I.L. Lebow, I.S. Reed, Theory and Design of Digital Machines (McGraw-Hill, New York, 1962)

[Belrhiti93] M. Belrhiti, G. Saucier, P. Abouzeid, "Dedicated Factorization Techniques for Complex Logic Blocks", International Workshop on Logic Synthesis (IWLS '93), May 23-26th 1993, Granlibakken Conference Center, Iahoe City, CA., USA.

[Belrhiti95] M. Belrhiti, G. Bosco, A. Guyot, "Efficient Generators of Adders", IWLS Iahoe City CA USA, May 1995.

[Besson93-a] T. Besson, H. Bouzouzou, I. Floricica, R. Roane et G. Saucier. "Input Order for ROBDDs Based on Kernel Analysis", in: Proc of EDAC, February 1993.

[Besson93-b] T. Besson, H. Bouzouzou, G. Saucier. "Ordering for ROBDDs Based on Sharing Analysis", in: Proc of Sashimi 1993, Japan, p.355.

[Besson96] T. Besson, "Optimisation des BDD et Applications", Thèse de Doctorat de l'INPG, Décembre 1996.

[Billon87] J.P Billon, "Perfect Normal Forms For Discrete Programs", BULL Research Report, No 87019, June 1987.

[Bollig94] B.Bollig, P.Savicky and I.Wegener, "On the improvement of variable orderings for OBDDs", in: Proc IFIP Workshop on Logic and Architecture Synthesis, December 1994, p71.

[Boole1854] G. Boole, "An Investigation of the Laws of Thought" (Walton, London, 1854) (reédité par Dover Books, New York, 1954)

[Brayton82] R. Brayton, C. T. McMullen. "The Decomposition and Factorization of Boolean Expressions", ISCAS Proceeding pp 49-54, Mai 1982.

[Brayton84] R.K. Brayton, G.D. Hachtel, C.T. McMullen, A.L. Sangiovanni-Vincentelli, Logic Minimization Algorithms for VLSI Synthesis (Kluwer Academic Publishers, Dordrecht, 1984)

[Brayton87a] R. Brayton, "Factoring Logic Functions",IBM J. Research, vol 31, pp 187-198, Mars 1987.

[Brayton87b] R. Brayton, R. Rudell, A. Wang, and A. Sangiovanni-Vincentelli "MIS: a multiple-level logic optimization system", In IEEE Transactions on CAD, pp 1062-1081, Novembre 1987.

[Brayton87c] R. Brayton, R. Rudell, A. Wang, and A. Sangiovanni-Vincentelli, "Multi-level logic optimization and the rectangular covering problem", pp 66-69, ICCAD 1987, Santa Clara, California, USA.

[Brayton93] R.K. Brayton, P.C. McGeer, J. Sanghavi, A.L. Sangiovanni-Vincentelli, "A New Exact Minimizer for Two-level Logic Synthesis, in: Logic Synthesis and Optimization", T. Sasao (Ed.) (Kluwer Academic Publishers, Dordrecht, 1993) pp. 1-31.

[Brent82] R. Brent, H. Kung, "A regular layout for parrallel adders", IEEE Transaction on Computers, 1982.

[Bryant86] R.E. Bryant, "Graph-based Algorithms for Boolean Functions Manipulation", IEEE Transaction on Computers, Vol. C-35, No. 8, pp. 677-692, August 1986

[Bryant88] R.E. Bryant, "On Complexity of VLSI Implementations and Graph Representations of Boolean Functions with Application to Integer Multiplication", Carnegie Mellon University Research Report, September 1988.

[Bryant92] R.E. Bryant, "Symbolic Boolean Manipulations with Ordered Binary Decision Diagrams", ACM Computing Surveys, Vol. 24, No. 3, pp. 293-318, September 1992.

[Burlet92] M. Burlet, "cours de DEA Recherche Opérationnelle sur les concepts fondamentaux de l'optimisation combinatoire", Université Joseph Fourrier - Grenoble 1- 1991-92

[Calazans91] N. Calazans, R. Jacobi, Q. Zhang and C. Trullemans. "Improving Binary Decision Diagrams Through Incremental Reduction and Improved Heuristics", in: Proc of Custem Integrated Circuits Conference, 1991.

[Calazans92] N. Calazans, Q. Zhang, R. Jacobi, B. Yernaux and A.M. Trullemans. "Advanced Ordering and Manipulation Techniques for Binary Decision Diagrams", in: Proc of European Design Automation Conference, 1992.

[Cébélieu95] M.C. Cébélieu, "Utilisation de macroblocs en synthèse VHDL", Thèse de Docteur INPG, Spécialité Microélectronique, Décembre 1995.

[Coudert92-a] O. Coudert, J. Madre, "Implicit and Incremental Computation of Primes and Essential Primes of Boolean Functions", 29th Design Automation Conference, Anaheim, CA, USA, pp. 36-39, June 1992.

[Coudert92-b] O. Coudert, J. Madre, "A New Method to Conpute Prime and Essential Prime Implicants of Boolean Functions", Proc. Brown/MIT Conference on Advanced Research in VLSI and Parallel Systems, Cambridge, MA, USA, March 1992.

[Coudert93-a] O. Coudert, J.C. Madre, "A New Graph Based Prime Computation Technique", in Logic Synthesis and Optimization, T. Sasao (Ed.) (Kluwer Academic Publishers, Dordrecht, 1993) pp. 33-57.

[Coudert93-b] O. Coudert, J.C. Madre, H. Fraisse, "A New Viewpoint on Two-Level Logic Minimization", Proc. 30th Design Automation Conference, Dallas, TX, pp. 625-630, June 1993.

[Coudert94] O. Coudert, "Two-level logic minimization: an overview", INTEGRATION, the VLSI journal 17 (1994) pp. 97-140.

[De Micheli84] G. De Micheli, "KISS : a Program for Optimal State Assignement of Finite State Machines", ICCAD 84, Santa Clara, November 1984.

[Duff91] C. Duff, "Codage d'automates et théorie des cubes intersectants", Thèse de Docteur INPG, Spécialité: Microélectronique, Mars 91.

[Felt93] E. Felt, G.York, "Dynamic BDD Variable Reordering", in: Proc EDAC 1993.

[Finke91] G. Finke, "cours de DEA Recherche Opérationnelle sur l'optimisation combinatoire appliquée" , Université Joseph Fourrier - Grenoble 1- 1991-92

[Friedman87] S. J. Friedman, D. J. Supowit. "Finding the optimal Variable Ordering for Binary Decision Diagrams". in: Poc of 24th Design Automation Conference, 1987.

[Friedman90] S. J. Friedman, K.J. Supowit, "Finding the optimal Variable Ordering for Binary Decision Diagrams", IEEE Transactions on computer, Vol. C-39, No. 5, pp. 710-713, May 1990.

[Fujita91] M. Fujita, Y. Matsunaga and T. Kakuda. "On Variable Ordering of Binary Decision Diagrams for the Application of Multi-level Logic Synthesis", in: Pro of European Design Automation Conference, 1991

[Fujiwara85] H. Fujiwara, "Logic testing and design for testability", MIT Press Series in Computer Systems. The MIT Press, Cambridge, Massachusetts 1985.

[Garey79] M. Garey and D.S. Johnson, Computers and Intractability: A Guide to the theory of NP Completeness (Freeman, San Francisco, 1979)

[Gondran85] M. Gondran, M. Minoux, "Graphes et Algorithmes", Editions Eyrolles 1985.

[Han87] T. Han, D. Carlson, "Fast area-efficient VLSI adders", Symposium on Computer Arithmetic, 1987.

[Hiol96] J.P. Hiol, "Optimisation Temporelle des Réseaux Logiques", Thèse de Doctorat, INPG, Grenoble, France, 1996

[Hong74] S.J. Hong, R.G. Cain, D.L.Ostapko, "MINI: A Heuristic Approach for Logic Minimization", IBM Journal R&D, pp. 443-458, 1974.

[Hong91] S.J. Hong, S. Muroga, "Absolute Minimization of Completely Specified Switching Functions", IEEE Transactions on Computers, Vol. 40, pp. 53-65, 1991.

[Ishiura91] N. Ishiura, H. Sawada, S.Yajima, "Minimization of Binary Decision Diagrams Based on Exchanges of Variables", in: Proc of IEEE International Conference on Computer Aided Design, Nov. 1991

[Jacobi91] R. Jacobi, N. Calzans and C. Trullemans, "Incremental Reduction of Binary Decision Diagrams", in: Proc of International Symposium on Circuits and Systems, Singapore, June 1991.

[Jacobi93] R. Jacobi. "A Study of the application of Binary Decision Diagrams on Multilevel Logic Synthesis". Doctoral Thesis, Université Catholique de Louvain, Belgium. Dec. 1993

[Jeong92] S. W. Jeong, B. Plessier, G. D. Hachtel, F. Somenzi. "Variable Ordering for Binary Decision Diagrams". in: Proc of European Design Automation Conference, 1992.

[Karnaugh53] M. Karnaugh, "The Map Method for Synthesis of Combinational Logic Circuits", AIEE Transactions on Communications and Electronics, Vol. 9, pp. 593-599, 1953

[Kelliher82] T. Kelliher, R. Owens, M. Irwin, T. Hwang, "A fast addition algorithm discovered by a program", IEEE Transaction on Computers, 1982.

[Khalil95] O Khalil, "Utilisation des bibliothèques de macro-blocs dans la synthèse de haut niveau et dans la migration technologique", Thèse de Docteur INPG, Spécialité : Microélectronique, 1995.

[Kogge93] P. Kogge, H. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations", IEEE Transaction on Computers, 1993.

[Koren93] I. Koren, "Computer Arithmetic Algorithms", Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[Kuntzmann68] J. Kuntzmann - "Algèbre de Boole", Edition Dunod, Paris, 1968.

[Lawler76] E.L. Lawler, "Combinatorial Optimization : Networks and Matroids", Holt, Rinehart and Winston, New York, 1976.

[Lee59] C. Y. Lee. "Representation of Switching Circuit by Binary Decision Diagrams". in: BSTJ, No. 38, July 1959, pp. 985-999.

[Loui82] M.C. Loui, G. Bilardi, "The Correctness of Tison's Method for Generating Prime Implicants", Report R-952, UILU-ENG 82-2218, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, 1982.

[Madre88] J.C Madre and J.P Billon: "Proving Circuit Correctness using Formal Comparison Between Expected and Extracted Behaviour", in: Pro. 25th ACM/IEEE Design Automation Conference, Anaheim, CA, USA, July 1988

[Malik88] S. Malik, A. R. Wang, R. K. Brayton and A. Sangiovanni Vincentilli, "Logic Verification using Binary Decision Diagrams in a Logic Synthesis Environment", in: Proc of IEEE International Conference on Computer Aided Design, 1988

[McCluskey59] E.L. Jr. McCluskey, "Minimization of Boolean Functions", Bell System Technical Journal, Vol. 35, pp. 1417-1444, April 1959.

[McMullen86] C. McMullen, J. Shearer, "Prime Implicants, Minimum Covers, and the Complexity of Logic Simplification", IEEE Transactions on Computers, Vol. C-35, pp. 761-762, Agust 1986.

[Mercer92] M.R. Mercer, R. Kapur, and D.E. Ros, "Functional Approaches to Generating Orderings for efficient Symbolic representations", Pro. of the 29th Design Automation Conference, June 1992

[Minato90] S. Minato, N. Ishiura, S. Yajima. "Shared Binary Decision Diagrams with Attributed Edges for Efficient Boolean Function Manipulation", in: Proc of 27th Design Automation Conference, 1990.

[Minato93] S. Minato, "Zero-Supressed BDDs for Set Manipulation in Combinatorial Problems", Proc. 30th Design Automation Conference, Dallas, TX, pp. 272-277, June 1993.

[Nakamura93] Y. Nakamura, K. Wakabayashi, T. Fujita, " A Partitioning Method For Area Optimization By Tree Analysis", Logic Synthesis and Optimization, edited by Tsutomu Sasao in: Kluwer Academic Publishers pp. 127-143

[Quine52] W.V.O. Quine, "The problem of Simplifying Truth Functions", American Mathematics Monthly, Vol. 59, pp. 521-531, 1952.

[Quine53] W.V.O. Quine, "Two Theorems about Truth Functions", Bol. Society of Mathematics Mexicana, Vol. 10, pp. 64-70, 1953

[Quine59] W.V.O. Quine, "On Cores and Prime Implicants of Truth Functions", American Mathematics Monthly, Vol. 66, pp. 755-760, 1959.

[Rudell86] R. L. Rudell, "Multiple-Valued Logic Minimization for PLA Synthesis", Research Report, UCB M86/65, 1986.

[Rudell87] R. L. Rudell, A. L. Sangiovanni-Vincentelli, "Multiple-Valued Minimization for PLA Optimization", IEEE Transactions on CAD, Vol 6, No. 5, pp. 727-750, September 1987.

[Rudell89] R. L. Rudell, "Logic Synthesis for VLSI Design", PhD Thesis, UCB/ERL M89/49, 1989.

[Rudell93] R.L. Rudell, "Dynamic Variable Ordering for Binary Decision Diagrams", Proc. Int. Conference on Computer Aided Design, Santa Clara, USA, pp. 42-47, November 1993.

[Sakouti93] K. Sakouti, "Synthèse et Optimisation Temporelle de Réseaux Booléens", Thèse de Doctorat, INPG, Grenoble, France, 1993

[Shannon49] C.E. Shannon, "The Synthesis of Two-terminal Switching Function", Bell System Technical Journal, Vol. 28, No. 1, pp. 59-98, 1949.

[Sicard88] P. Sicard, "Nouvelles Méthodes de Synthèse Logique", Thèse de Doctorat, INPG, Grenoble, France 1988.

[Sklansky60] J. Sklansky, "Conditionnal sum addition logic", IRE Transaction on Electronic Computers, 1960.

[Slage69] J.R. Slage, C.L. Chang, R.C.T. Lee, "Completeness Theorems for Semantics Resolution in Consequence Finding", Proc. Int. Join Conference on Artificiel Intelligence, pp. 281-285, 1969.

[Slage70] J.R. Slage, C.L. Chang, R.C.T. Lee, "A New Algorithm for Generating Prime Implicants", IEEE Transactions on Computers, Vol. C-19, No. 4, pp. 304-310, 1970.

[Swamy93] G.M. Swamy, P. McGeer, R.K. Brayton, "A Fully Quine-McCluskey Procedure using BDD's", Report #UCB/ERL M92/127, November 1992. Also in: Proc. Int. Workshop on Logic Synthesis, Lake Tahoe, CA, May 1993.

[Tison67] P. Tison, "Generalized Consensus Theory and Application to the Minimization of Boolean Functions", IEEE Transactions on Electronic Computers, Vol. EC-16, No. 4, pp. 446-456, 1967.

[Touati90] H.J. Touati, "Performance-Oriented Technology Mapping", PhD Thesis, No. UCB/ERL M90/109, 1990.

[Wei85] B. Wei, C. Thomson, Y. Chen, "Time-Optimal design of a CMOS adder", Asilomar Conference on Circuits, Systems and Computers, 1985.

[Xilinx96] XILINX -Data Book- 1996.

[Yang91] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide", Microelectronics Center of North Carolina, January 1991.

Annexe

FINAL COM'L: -10/15/20 IND: -18/24

MACH230-10/15/20

High-Density EE CMOS Programmable Logic

Advanced Micro Devices

DISTINCTIVE CHARACTERISTICS

- 84 Pins
- 128 Macrocells
- 10 ns t_{PD} Commercial
  18 ns t_{PD} Industrial
- 100 MHz f_{CNT}
- 70 Inputs
- 64 Outputs
- 128 Flip-flops; 4 clock choices
- 8 "PAL26V16" blocks with buried macrocells
- Pin-compatible with MACH130, MACH131, MACH231, and MACH435

GENERAL DESCRIPTION

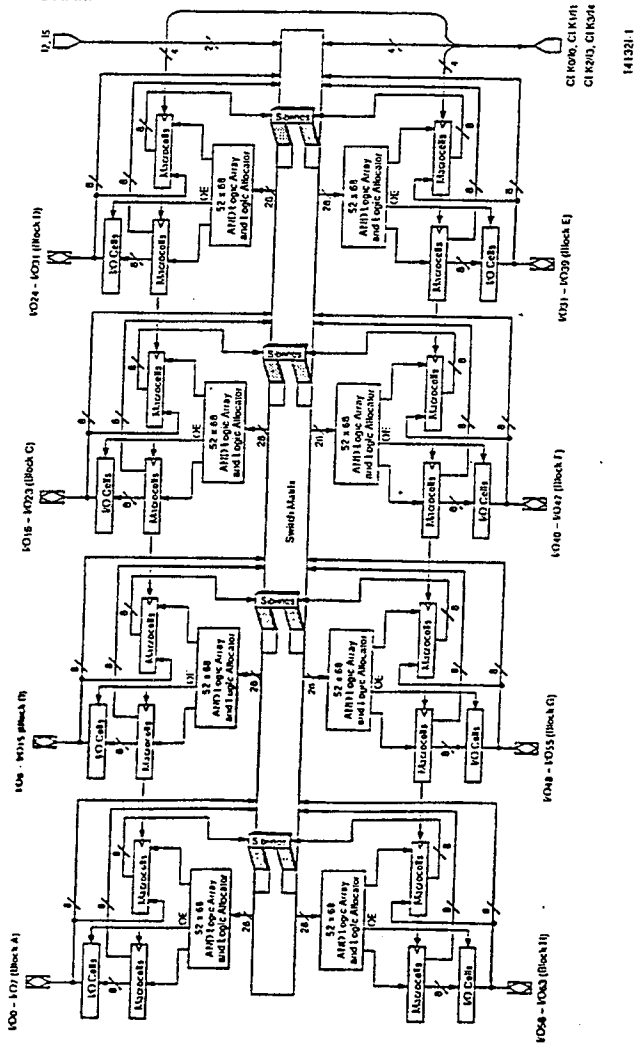
The MACH230 is a member of AMD's high-performance EE CMOS MACH 2 device family. This device has approximately twelve times the logic macrocell capability of the popular PAL22V10 without loss of speed.

The MACH230 consists of eight PAL blocks interconnected by a programmable switch matrix. The switch matrix connects the PAL blocks to each other and to all input pins, providing a high degree of connectivity between the fully-connected PAL blocks. This allows designs to be placed and routed efficiently.

The MACH230 has two kinds of macrocell: output and buried. The output macrocell provides registered, latched, or combinatorial outputs with programmable polarity. If a registered configuration is chosen, the register can be configured as D-type or T-type to help reduce the number of product terms. The register type decision can be made by the designer or by the software. All output macrocells can be connected to an I/O cell. If a buried macrocell is desired, the internal feedback path from the macrocell can be used, which frees up the I/O pin for use as an input.

The MACH230 has dedicated buried macrocells which, in addition to the capabilities of the output macrocell, also provide input registers for use in synchronizing signals and reducing setup time requirements.

BLOCK DIAGRAM



FUNCTIONAL DESCRIPTION

The MACH230 consists of eight PAL blocks connected by a switch matrix. There are 64 I/O pins and 2 dedicated input pins feeding the switch matrix. These signals are distributed to the four PAL blocks for efficient design implementation. There are 4 clock pins that can also be used as dedicated inputs.

The PAL Blocks

Each PAL block in the MACH230 (Figure 1) contains a 64-product-term logic array, a logic allocator, 8 output macrocells, 8 buried macrocells, and 8 I/O cells. The switch matrix feeds each PAL block with 26 inputs. This makes the PAL block look effectively like an independent "PAL26V16" with 8 buried macrocells.

In addition to the logic product terms, two output enable product terms, an asynchronous reset product term, and an asynchronous preset product term are provided. One of the two output enable product terms can be chosen within each I/O cell in the PAL block. All flip-flops within the PAL block are initialized together.

The Switch Matrix

The MACH230 switch matrix is fed by the inputs and feedback signals from the PAL blocks. Each PAL block provides 16 internal feedback signals and 8 I/O feedback signals. The switch matrix distributes these signals back to the PAL blocks in an efficient manner that also provides for high performance. The design software automatically configures the switch matrix when fitting a design into the device.

The MACH230 places a restriction on buried macrocell feedback only. Buried macrocell feedback from one block can be used as an input only to that block or its "sibling" block. Sibling blocks are illustrated in the block diagram and in Table 1. Output macrocell feedback is not restricted.

Table 1. Sibling Blocks

PAL Block	Sibling Block
A	H
B	G
C	F
D	E
E	D
F	C
G	B
H	A

The Product-Term Array

The MACH230 product-term array consists of 64 product terms for logic use, and 4 special-purpose product terms. Two of the special-purpose product terms provide programmable output enable, one provides asynchronous reset, and one provides asynchronous preset.

The Logic Allocator

The logic allocator in the MACH230 takes the 64 logic product terms and allocates them to the 16 macrocells as needed. Each macrocell can be driven by up to 16 product terms. The design software automatically configures the logic allocator when fitting the design into the device.

Table 2 illustrates which product term clusters are available to each macrocell within a PAL block. Refer to Figure 1 for cluster and macrocell numbers.

Table 2. Logic Allocation

Macrocell		Available Clusters
Output	Buried	
M_0	M_1	C_0, C_1, C_2 C_0, C_1, C_2, C_3
M_2	M_3	C_1, C_2, C_3, C_4 C_2, C_3, C_4, C_5
M_4	M_5	C_3, C_4, C_5, C_6 C_4, C_5, C_6, C_7
M_6	M_7	C_5, C_6, C_7, C_8 C_6, C_7, C_8, C_9
M_8	M_9	C_7, C_8, C_9, C_{10} C_8, C_9, C_{10}, C_{11}
M_{10}	M_{11}	$C_9, C_{10}, C_{11}, C_{12}$ $C_{10}, C_{11}, C_{12}, C_{13}$
M_{12}	M_{13}	$C_{11}, C_{12}, C_{13}, C_{14}$ $C_{12}, C_{13}, C_{14}, C_{15}$
M_{14}	M_{15}	C_{13}, C_{14}, C_{15} C_{14}, C_{15}

The Macrocell

The MACH230 has two types of macrocell: output and buried. The output macrocells can be configured as either registered, latched, or combinatorial, with programmable polarity. The macrocell provides internal feedback whether configured with or without the flip-flop. The registers can be configured as D-type or T-type, allowing for product-term optimization.

The flip-flops can individually select one of four clock/gate pins, which are also available as data inputs. The registers are clocked on the LOW-to-HIGH transition of the clock signal. The latch holds its data when the gate input is HIGH, and is transparent when the gate input is LOW. The flip-flops can also be asynchronously initialized with the common asynchronous reset and preset product terms.

The buried macrocells are the same as the output macrocells if they are used for generating logic. In that case, the only thing that distinguishes them from the output macrocells is the fact that there is no I/O cell connection, and the signal is only used internally. The buried macrocell can also be configured as an input register or latch.

The I/O Cell

The I/O cell in the MACH230 consists of a three-state output buffer. The three-state buffer can be configured in one of three ways: always enabled, always disabled, or controlled by a product term. If product term control is chosen, one of two product terms may be used to provide the control. The two product terms that are available are common to all I/O cells in a PAL block.

These choices make it possible to use the macrocell as an output, an input, a bidirectional pin, or a three-state output for use in driving a bus.

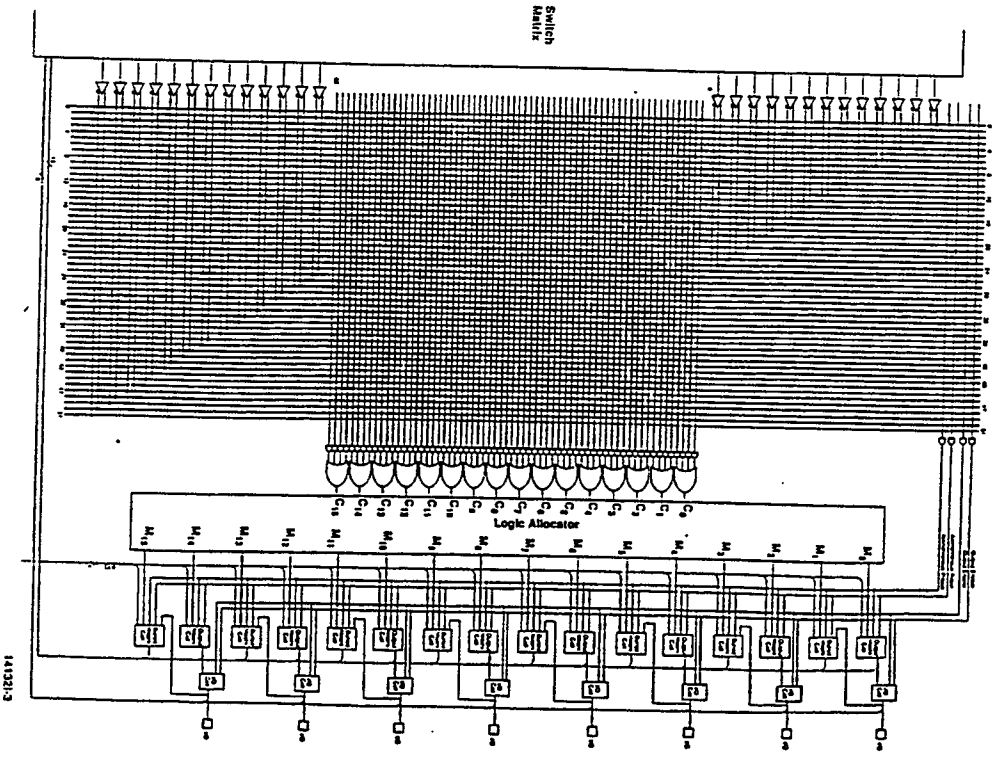


Figure 1. MACH230 PAL Block

XILINX®

XC9500 In-System Programmable CPLD Family

June 1, 1996 (Version 1.0)

Preliminary Product Information

Features

- High-performance
  - 5 ns pin-to-pin logic delays on all pins
  - f_{CNT} to 125 MHz
- Large density range
  - 36 to 576 macrocells with 800 to12,800 usable gates
- 5 V in-system programmable
  - Endurance of 10,000 program/erase cycles
  - Program/erase over full voltage and temperature range
- Enhanced pin-locking architecture
- Flexible 36V18 Function Block
  - 90 product terms drive any or all of 18 macrocells within Function Block
  - Global and product term clocks, output enables, set and reset signals
- Extensive IEEE Std 1149.1 boundary-scan (JTAG) support
- Programmable power reduction mode in each macrocell
- Slew rate control on individual outputs
- User programmable ground pin capability
- Extended pattern security features for design protection
- High-drive 24 mA outputs with 3.3 V or 5 V I/O capability
- PCI compliant (-5, -7, -10 speed grades)
- Advanced 0.6µm CMOS 5V FastFLASH technology

Description

The XC9500 CPLD family provides advanced in-system programming and test capabilities for high performance, general purpose logic integration. All devices are in-system programmable for a minimum of 10,000 program/erase cycles. Extensive IEEE 1149.1 (JTAG) boundary-scan support is also included on all family members.

As shown in Table 1, the nine devices of the XC9500 family range in logic density from 800 to over 12,800 usable gates with 36 to 576 registers, respectively. Multiple package options and associated I/O capacity are shown in Table 2. The XC9500 family is fully pin-compatible allowing easy design migration across multiple density options in a given package footprint.

The XC9500 architectural features address the requirements of in-system programmability. Enhanced pin-locking capability avoids costly board rework. An expanded JTAG instruction set allows version control of programming patterns and in-system debugging. In-system programming throughout the full device operating range and a minimum of 10,000 program/erase cycles provide worry-free reconfigurations and system field upgrades.

Advanced system features include output slew rate control and user-programmable ground pins to help reduce system noise. I/Os may be configured for 3.3 V or 5 V operation. All outputs provide 24 mA drive.

Architecture Description

Each XC9500 device is a subsystem consisting of multiple Function Blocks (FBs) and I/O Blocks (IOBs) fully interconnected by the FastCONNECT switch matrix. The IOB provides buffering for device inputs and outputs. Each FB provides programmable logic capability with 36 inputs and 18 outputs. The FastCONNECT switch matrix connects all FB outputs and input signals to the FB inputs. For each FB, 12 to 18 outputs (depending on package pin-count) and associated output enable signals drive directly to the IOBs. See Figure 1.

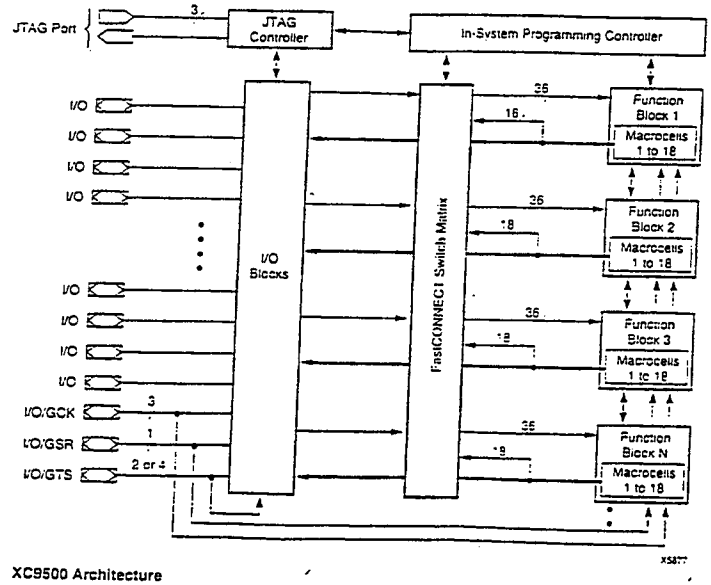


Figure 1: XC9500 Architecture

Table 1: XC9500 Device Family

	XC9536	XC9572	XC95108	XC95144	XC95180	XC95216	XC95288	XC95432	XC95576
Macrocells	36	72	108	144	180	216	288	432	576
Usable Gates	800	1,600	2,400	3,200	4,000	4,800	6,400	9600	12,800
Registers	36	72	108	144	180	216	288	432	576
t_{PD} (ns)	5	7.5	7.5	7.5	10	10	10	10	12
t_{SU} (ns)	4.5	5.5	5.5	5.5	6.5	6.5	6.5	6.5	9.5
t_{CO} (ns)	4.5	5.5	5.5	5.5	6.5	6.5	6.5	6.5	9.5
f_{CNT} (MHz)	125	125	125	125	111	111	111	111	100
f_{SYSTEM} (MHz)	100	83	83	83	67	67	67	67	67
	Preliminary								

Note: f_{CNT} = Operating frequency for 16-bit counters
f_{SYSTEM} = Internal operating frequency for general purpose system designs spanning multiple FBs.

Table 2: Available Packages and Device I/O Pins

	XC9536	XC9572	XC95108	XC95144	XC95180	XC95216	XC95288	XC95432	XC95576
44-Pin PLCC	34								
44-Pin VQFP	34								
84-Pin PLCC		69	69						
100-Pin PQFP		72	81	81					
100-Pin TQFP		72	81						
160-Pin PQFP			108	133	133	133			
208-Pin HQFP					168	168	168		
304-Pin HQFP							192	232	232

Note: Does not include the dedicated JTAG pins.

Function Block

Each Function Block, as shown in Figure 2, is comprised of 18 independent macrocells, each capable of implementing a combinatorial or registered function. The FB also receives global clock, output enable, and set/reset signals. The FB generates 18 outputs that drive the FastCONNECT switch matrix. These 18 outputs and their corresponding output enable signals also drive the IOB.

Logic within the FB is implemented using a sum-of-products representation. Thirty-six inputs provide 72 true and complement signals into the programmable AND-array to form 90 product terms. Any number of these product terms, up to the 90 available, can be allocated to each macrocell by the product term allocator.

Each FB supports local feedback paths that allow any number of FB outputs to drive into its own programmable AND-array without going outside the FB.

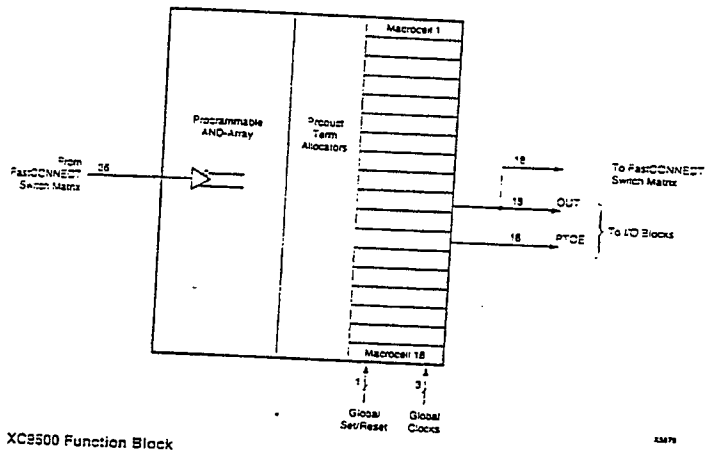


Figure 2: XC9500 Function Block

Macrocell

Each XC9500 macrocell may be individually configured for a combinatorial or registered function. The macrocell and associated FB logic is shown in Figure 3.

Five direct product terms from the AND-array are available for use as primary data inputs (to the OR and XOR gates) to implement combinatorial functions, or as control inputs including clock, set/reset, and output enable. The product term allocator associated with each macrocell selects how the five direct terms are used.

The macrocell register can be configured as a D-type or T-type flip-flop, or it may be bypassed for combinatorial operation. Each register supports both asynchronous set and reset operations. During power-up, all user registers are initialized to the user-defined preload state (default to 0 if unspecified).

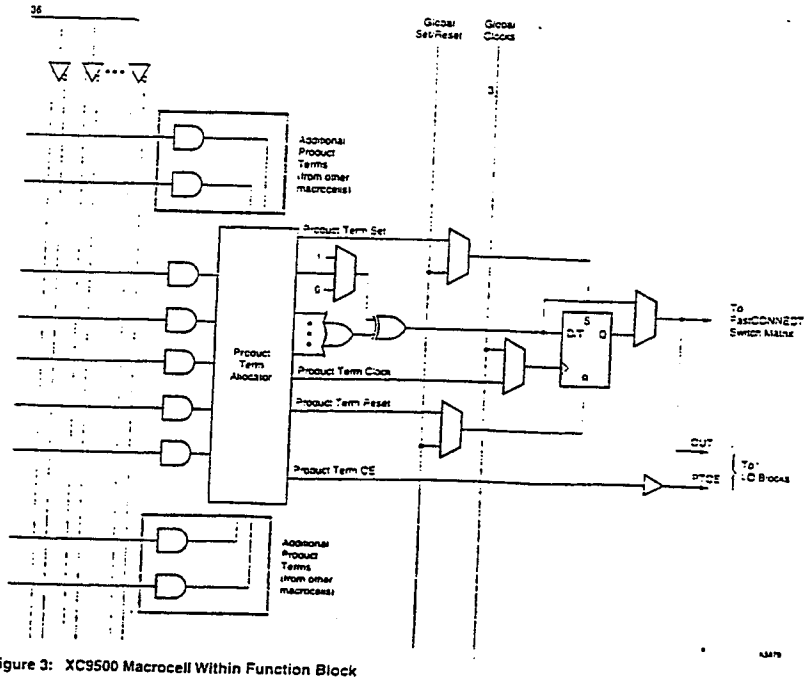


Figure 3: XC9500 Macrocell Within Function Block

All global control signals are available to each individual macrocell, including clock, set/reset, and output enable signals. As shown in Figure 4, the macrocell register clock originates from either of three global clocks or a product term clock. Both true and complement polarities of a GCK pin can be used within the device. A GSR input is also provided to allow user registers to be set to a user-defined state.

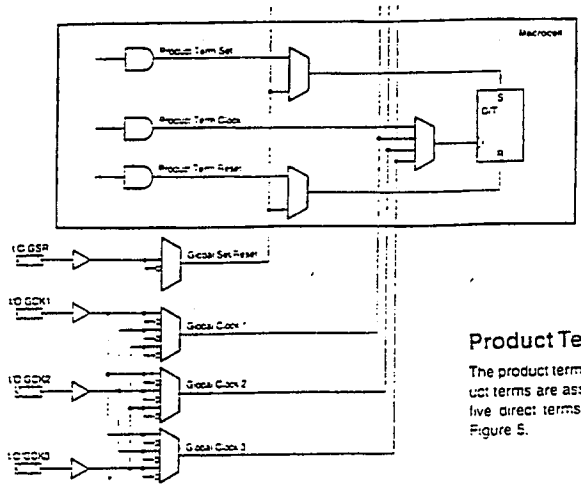


Figure 4: Macrocell Clock and Set/Reset Capability

Product Term Allocator

The product term allocator controls how the five direct product terms are assigned to each macrocell. For example, all five direct terms can drive the OR function as shown in Figure 5.

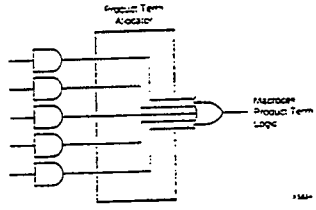


Figure 5: Macrocell Logic Using Direct Product Term

The product term allocator can re-assign other product terms within the FB to increase the logic capacity of a macrocell beyond five direct terms. Any macrocell requiring additional product terms can access uncommitted product terms in other macrocells within the FB. Up to 15 product terms can be available to a single macrocell with only a small incremental delay of t_{PTA} as shown in Figure 6.

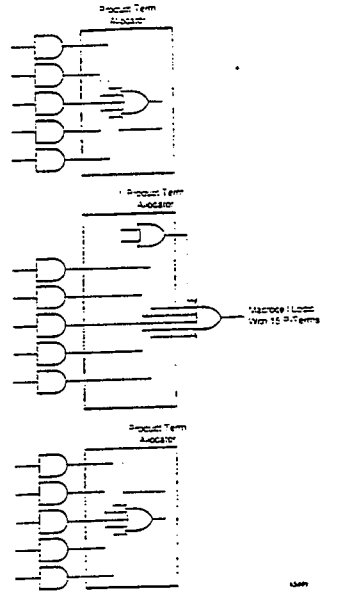


Figure 6: Product Term Allocation With 15 Product

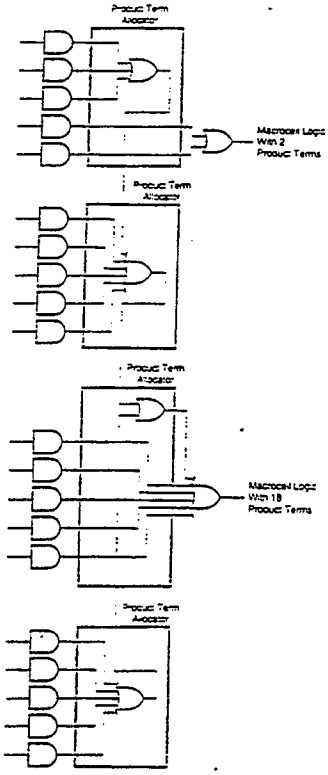


Figure 7: Product Term Allocation Over Several Macrocells

The internal logic of the product term allocator is shown in Figure 8.

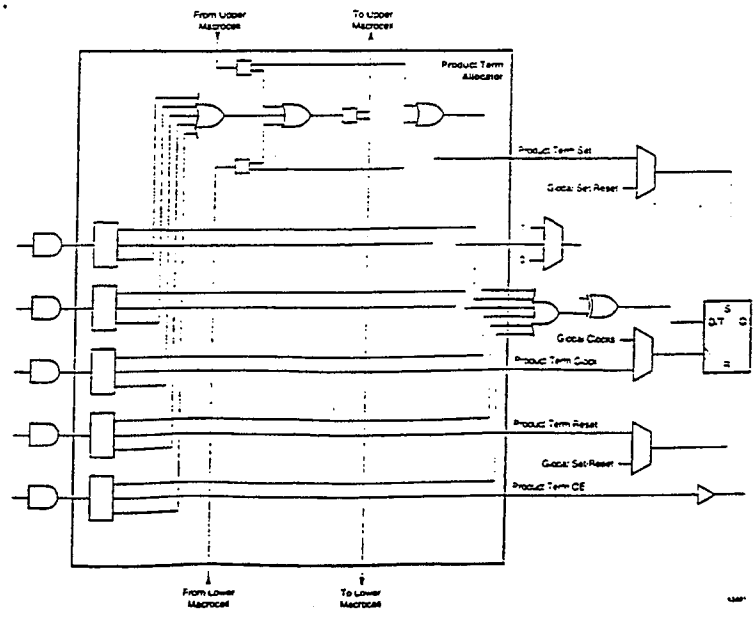


Figure 8: Product Term Allocator Logic

Résumé

Cette thèse propose et analyse de nouvelles méthodes de synthèse logique. L'analyse concerne des outils de la "troisième génération" d'écriture de bases irrédondantes de fonctions booléennes, à savoir les minimiseurs dits symboliques.
Cette génération de minimiseurs conduit à la solution optimale plus rapidement et avec moins d'espace mémoire que les heuristiques de la minimisation explicite. Elle permet également le calcul de la forme complémentée minimale sans être exposée à des problèmes d'explosion en complexité, ce qui permet d'aboutir à un choix efficace entre une fonction et son complément.

Nous avons abordé ensuite les problèmes de granularité des expressions factorisées. Nous avons proposé une méthode originale de réinjection qui intègre d'une façon concurrente une phase de minimisation symbolique des expressions booléennes. Cette méthode a permis de "corriger" la granularité : d'une part, des expressions booléennes obtenues par la factorisation, d'autre part, des équations obtenues par une description de haut niveau de type VHDL. La méthode proposée peut être également appliquée en tant que minimiseur logique qui tient compte du partage de la logique entre les expressions booléennes, ce qui n'est pas possible avec un minimiseur logique local ou global. Les expériences pratiques et l'application sur les réseaux programmables de type CPLD sont concluantes.

Enfin, nous avons proposé une méthode originale de l'exploration de l'espace des solutions des macro-générateurs de type additionneur. Cette méthode est fondée sur le filtrage des solutions générées et l'amélioration par dérivation d'une solution donnée. Cette approche peut être efficacement appliquée sur la macro-génération sous contraintes temporelles.

Mots clés : Synthèse logique, Minimisation symbolique, Calcul de De Morgan, Réinjection, Granularité technologique, Macro-génération sous contraintes temporelles.