Méthodes de factorisation algébrique dédiées aux circuits intégrés complexes

Pierre Abouzeid

Thèse

Présentée par

Pierre ABOUZEID

Pour obtenir le grade de **Docteur**

de l'Institut National Polytechnique de Grenoble

(arrêté ministériel du 23 Novembre 1988)

(Spécialité **Microélectronique**)

Méthodes de Factorisation Algébrique Dédiées aux Circuits Intégrés Complexes

---

Date de soutenance: 18 Décembre 1992

Composition du Jury:

Mr J. Mossière	Président
Mme G. Saucier	
Mme A.M. Trullemans	Rapporteur
Mr A. Greiner	Rapporteur
Mr R. Brayton	

Thèse Préparée au sein du Laboratoire Conception de Systèmes Intégrés

Thèse

Présentée par

Pierre ABOUZEID

Pour obtenir le grade de **Docteur**

de l'Institut National Polytechnique de Grenoble

(arrêté ministériel du 23 Novembre 1988)

(Spécialité **Microélectronique**)

Méthodes de Factorisation Algébrique Dédiées aux Circuits Intégrés Complexes

---

Date de soutenance: 18 Décembre 1992

Composition du Jury:

Mr J. Mossière	Président
Mme G. Saucier	
Mme A.M. Trullemans	Rapporteur
Mr A. Greiner	Rapporteur
Mr R. Brayton	

Thèse Préparée au sein du Laboratoire Conception de Systèmes Intégrés

Remerciements

Je tiens d'abord à remercier Madame Gabrièle SAUCIER, Professeur à l'ENSIMAG et directrice du laboratoire CSI, pour m'avoir accueilli dans son laboratoire et pour avoir guidé mes recherches. Je la remercie également pour m'avoir transmis une part de son énergie qui a permis l'aboutissement de ce travail.

Je remercie Monsieur Jacques MOSSIERE, Professeur et directeur de l'ENSIMAG, de me faire l'honneur de présider ce Jury.

Je remercie, Professeur Anne-Marie TRULLEMANS de l'université de Louvain et Professeur Alain GREINER de l'université Paris 6, d'avoir accepté d'être rapporteurs de cette thèse.

Je remercie Professeur Robert BRAYTON de l'université de Californie à Berkeley, d'avoir accepté de faire partie de mon Jury.

Je remercie tous les membres du CSI pour l'agréable ambiance de travail, toute l'équipe de synthèse logique pour son aide et plus spécialement Jérôme FRON, Thierry BESSON et Mohammed BELRHITI, qui ont contribué à l'aboutissement de ce travail. Je voudrais enfin remercier Michel CRASTES pour ses nombreuses remarques et suggestions et pour les innombrables discussions constructives.

Résumé

Cette thèse propose des méthodes de synthèse dédiées aux circuits intégrés complexes. Elle concerne l'étape de factorisation algébrique dont le but est de réduire la complexité des expressions Booléennes évaluées en terme de littéraux. Les méthodes classiques généralement proposées, amènent une bonne minimisation de la surface active mais peuvent entraîner un mauvais contrôle de la connectique. Cette thèse présente d'abord l'état de l'art critique sur les techniques de factorisation algébrique poussées incluant les techniques dites Booléennes. Dans la suite, deux approches alternatives de factorisation plus restreintes sont proposées. La première est réduite à une division par conoyaux et la deuxième concerne une factorisation dite lexicographique encore plus restrictive, dont le but est de préparer une connectique simplifiée. Les résultats expérimentaux ont permis de définir à partir de quel seuil de complexité, il convient d'appliquer ces deux méthodes pour obtenir une bonne surface globale ainsi qu'un bon facteur de routage.

Mots-clés

Synthèse logique multicouche, factorisation algébrique, factorisation Booléenne, synthèse sur cellules standards, factorisation lexicographique.

Abstract

This PhD deals with synthesis methods for highly complex integrated circuits, and more precisely the algebraic factorization step. The goal of this step is to reduce the complexity of Boolean expressions in terms of litteral count. The classical factorization methods aim at reducing the active area but may lead to a poor control of the wiring complexity. This PhD presents a critical state of the art review of classical algebraic factorization methods including sophisticated methods using the so-called Boolean division. In the following, two alternative methods are proposed for complex combinational blocks. The first one restricts the algebraic division to a cokernel division ; the second one is even more restricted as it imposes a fixed input order ; it is called lexicographical factorization and its goal is to prepare a simplified wiring. Practical experiments show the complexity threshold for which these restricted factorization methods are suitable to yield optimized circuits.

Key-Words

Multilevel logic synthesis, algebraic factorization, Boolean factorization, synthesis for standard cell implementation, lexicographical factorization.

TABLE DES MATIERES

INTRODUCTION

Les outils de synthèse logique sont aujourd'hui inclus dans la plupart des logiciels commerciaux de conception de circuits intégrés digitaux. Ils permettent de générer de façon automatisée un réseau optimisé de portes dans une technologie cible à partir d'une description comportementale. Les critères d'optimisation sont soit la surface globale, soit les performances en terme de chemin critique, soit un compromis entre les deux. Quelque soit la complexité du circuit ou le niveau d'abstraction de la description initiale, des blocs combinatoires sont identifiés à une certaine étape de la synthèse et on est alors ramené au problème élémentaire de la synthèse d'un ensemble de fonctions Booléennes sur une cible technologique donnée. Cette thèse est donc dédiée au problème de la synthèse de blocs combinatoires et la cible technologique considérée est une bibliothèque de cellules standards couramment réalisée dans une technologie CMOS.

La synthèse d'un ensemble de fonctions Booléennes sur une bibliothèque de cellules standards encore appelée synthèse multicouche a été largement étudiée au cours de la dernière décennie. Les méthodes proposées sont classiquement décomposées en trois étapes. La première consiste, à partir d'une forme irrédondante en somme de monômes, à diminuer la complexité de telles expressions évaluée en nombre d'occurences de littéraux par des techniques de factorisation algébrique [Bra82,Bra86b,Bra87,Vas90] proche des techniques de factorisations algébriques courantes. Des techniques de factorisation plus sophistiquées, dites techniques de factorisation Booléennes, mettent en jeu des propriétés spécifiques de l'algèbre de Boole et font un usage plus poussé des valeurs indéfinies ou φ-Booléennees des fonctions [Bra87,Sat87,Mal89,Sal89, Dam90,Sav90,Sav91]. L'objectif de ces méthodes est d'exprimer l'ensemble des fonctions Booléennes par des expressions factorisées en minimisant le nombre de littéraux apparaissant dans ces expressions; les étapes de factorisation sont suivies par une étape de décomposition en sous-fonctions telle que chaque sous-fonction puisse être réalisée par une cellule de la bibliothèque. Cette étape est fortement dépendante des critères d'optimisation (surface ou chemin critique) et de nouveaux travaux de recherche ont mené à des méthodes efficaces [Det87,Leg88,Keu90,Cra91], actuellement implantées dans les outils du commerce. Enfin une dernière étape communément appelée étape d'optimisation de réseaux Booléens, consiste à modifier le réseau des cellules obtenu à l'étape précédente pour satisfaire un critère précis, en général

un chemin critique minimal exigé par le concepteur et non respecté dans le réseau proposé. Ces techniques d'optimisation peuvent modifier profondément le réseau [Tou91] ou simplement le modifier légèrement par insertion de buffers, ou d'arbres de buffers, de remplacement de portes lentes par des portes rapides ou de réplication de portes sur les chemins critiques [Sak90,Cra91,Yos91].

Le travail de cette thèse concerne la première étape de synthèse logique, à savoir l'étape de factorisation algébrique. Comme il a été dit précédemment, cette étape a comme objectif la réduction de la complexité d'un ensemble d'expressions Booléennes en terme d'occurence de littéraux. Ceci est principalement obtenu en recherchant des parties logiques communes qui ne seront pas répliquées soit à l'intérieur de fonctions Booléennes soit entre fonctions Booléennes. Il a été noté de façon expérimentale que ces techniques de factorisation pouvaient amener pour des blocs combinatoires complexes (gros contrôleurs par exemple) à des problèmes de temps de calcul trop important, à une surface de connectique ou de "routage" dans des canaux de routage excessive et à des chemins critiques très importants (problème de fanout). Cette thèse a pour objectif d'examiner ce problème et de proposer des techniques alternatives en diminuant la puissance de ces méthodes de factorisation et en prenant mieux en compte le problème de la connectique. Donc les approches présentées dans cette thèse, sont fondées sur des algorithmes de base développés à Berkeley [Bra82,Bra87], à savoir les algorithmes de recherche de noyaux et de division et substitution algébriques. En utilisant ces algorithmes, nous avons développé deux méthodes de factorisation, la première est une méthode de factorisation "faible" qui permet aussi bien une réduction importante du temps de calcul qu'une amélioration des résultats obtenus par les méthodes classiques, et la deuxième est une méthode de factorisation lexicographique qui permet de structurer les réseaux Booléens suivant un ordre des variables d'entrée.

Dans un premier chapitre, les définitions techniques de base de la factorisation algébrique sont rappelées. Il s'agit de rechercher des candidats diviseurs à savoir les noyaux, les monômes et parties de monômes communs à

plusieurs fonctions et à diviser les fonctions algébriquement par ces candidats diviseurs.

Dans un deuxième chapitre, une première technique de factorisation algébrique restreinte est proposée. Il s'agit de ne pas diviser une fonction Booléenne par les noyaux mais seulement par les conoyaux en évitant une substitution algébrique [Abo92,Sau92b] systématique de ces noyaux ou parties de noyaux dans les fonctions. Des techniques de filtrage développées dans le cadre de ce travail, ont permis de mieux contrôler la connectique dans les circuits pendant la phase de factorisation. L'impact de cette factorisation est étudié sur un grand nombre de cas test, et il est prouvé qu'à partir d'une certaine complexité, cette méthode de factorisation restreinte conduit à des meilleurs résultats en chemin critique et en surface globale. Ces comparaisons se feront en utilisant l'environnement de synthèse logique du laboratoire "Conception de Système Intégrés" à savoir l'outil de synthèse ASYL [Cra89].

Dans le chapitre III, une méthode novatrice de factorisation plus restrictive encore est proposée. Elle est dédiée aux blocs combinatoires de très haute complexité (gros contrôleurs, en particulier). Comme la méthode proposée au chapitre I, elle réduit la division algébrique à une division par les conoyaux. De plus, en vue de simplifier la connectique, cette factorisation dite factorisation lexicographique [Abo90a,b,Poi90] cherche à réaliser la logique par des cônes dans lesquels les variables d'entrée pénètrent en respectant un ordre identique pour tous les cônes. Ceci est obtenu en attachant à un couple conoyau/noyau une relation dite relation de précédence. Les noyaux sont alors choisis suivant une approche gourmande ("greedy") en vérifiant qu'un couple conoyau/noyau sélectionné ne viole pas les relations de précédence associées aux couples conoyaux/noyaux déjà choisis. On montrera que cette méthode de factorisation est efficace pour des blocs à très haute complexité, et on déterminera à partir de quel seuil de complexité il convient de l'appliquer. On s'intéresse en particulier à l'impact de cette factorisation sur la complexité de la connectique.

Dans le quatrième chapitre, on vérifiera que les techniques de factorisation Booléenne qui recherchent de façon plus sophistiquée des candidats diviseurs sont peu adaptées aux blocs combinatoires complexes. Cette expertise sera essentiellement expérimentale et sera faite dans le cadre du système de synthèse

de l'université de Berkeley "le système SIS". De nombreuses expérimentations effectuant des mesures sur des circuits achevés ("placés et routés") définiront également les plages de complexité où ces méthodes sont efficaces.

Un dernier chapitre rapportera sur des expérimentations dans les deux systèmes de synthèse ASYL et SIS. Ces experimentations mettent en jeu des techniques de décomposition diversifiées. L'analyse des résultats doit se faire avec prudence car les résultats finaux dépendent à la fois des méthodes de factorisation et de décomposition.

Cette thèse aura donc comme objectif de proposer des méthodes de factorisation algébrique spécifiques pour des blocs combinatoires à très haute complexité et fera une très grande part à l'expérimentation.

Chapitre I

Méthodes Classiques de Factorisation Algébrique

I.1 DÉFINITIONS PRÉALABLES

- ***Définition I.1: Algèbre de Boole***

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

- ***Définition I.2: 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.

- ***Définition I.3: Littéral***

  Un littéral est une variable Booléenne simple dans sa forme directe ou complémentée.

  Exemple: a et $\overline{a}$ sont deux littéraux distincts.

- ***Définition I.4: Fonction Booléenne***

  Une fonction Booléenne est une fonction de B^n dans B^m.

  – n = 1, m = 1 la fonction est dite fonction simple d'une variable simple.

  – n > 1, m = 1 la fonction est dite fonction simple d'une variable générale.

  – n > 1, m >1 la fonction est dite fonction générale d'une variable générale.

  – n = 1, m > 1 la fonction est dite fonction générale d'une variable simple.

  Exemple: la fonction F(a,b) = a + b est une fonction Booléenne simple d'une variable générale.

- ***Définition I.5: Fonction Booléenne incomplète (ou φ-Booléenne)***

  Une fonction Booléenne F est dite incomplète si elle est définie par:

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

  ϕ représente la Valeur Indéfinie VI (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:

  – $C_1(F)$ représente la couverture à 1 d'une fonction, c'est à dire l'ensemble des monômes pour lesquels la valeur de la fonction est 1.

  – $C_0(F)$ représente la couverture à 0 d'une fonction, c'est à dire l'ensemble des monômes pour lesquels la valeur de la fonction est 0.

  – $C_\phi(F)$ représente la couverture à ϕ d'une fonction, c'est à dire l'ensemble des monômes pour lesquels la valeur de la fonction est ϕ.

  – La borne supérieure de F est obtenue en remplaçant tous les ϕ par des 1. Elle est notée SUP(F).

  – La borne inférieure de F est obtenue en remplaçant tous les ϕ par des 0. Elle est notée INF(F).

- ***Définition I.6: Monôme (cube)***

  Un monôme m (dit également cube) est un produit de p variables simples distinctes sous forme normale (x) ou complémentée (x).

  p est le degré (ou longueur) du monôme.

  La taille d'un monôme est le nombre de points qu'il couvre.

  On rappelle qu'un monôme couvre un point si la valeur de ce monôme vaut 1 en ce point.

  Exemple: $m = a.b.\overline{c}$ est un monôme alors que $m = a.\overline{a}$ ne l'est pas.

- ***Définition I.7: Relation d'ordre entre monômes***

  On dit qu'un monôme m_1 est inférieur à un monôme m_2, que l'on note $m_1<m_2$, 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.

Exemple: $m_1 = a.b.c.d$ $m_2 = a.b$ $\Rightarrow$ $m_1 < m_2$

- ***Définition I.8: Expression Booléenne***

  Une expression Booléenne d'une fonction est construite à partir des variables Booléennes simples de la fonction et de l'ensemble des opérateurs logiques: l'union (+), l'intersection (.) et la négation (!).

- ***Définition I.9: Expression Booléenne polynomiale***

  Une expression Booléenne F est dite polynomiale si elle est sous la forme de somme de monômes: $F = \sum_{i=1}^{n} m_i$.

  On appelle M_F l'ensemble $\{m_i\}_{i=1..n}$ des monômes de F tel que $F = \sum_{i=1}^{n} m_i$.

  Exemple: $F = a.b + c.\overline{d} + g$ est une expression Booléenne polynomiale, et $M_F = \{a.b, c.\overline{d}, g\}$.

- ***Définition I.10: Relation d'ordre sur les fonctions Booléennes***

  Etant données deux fonctions Booléennes F et G, une relation d'ordre sur ces fonctions est définie par:

$$F \leq G \quad \Rightarrow \quad \begin{cases} F.G = F \\ \text{et} \\ F + G = G \end{cases}$$

  Autrement dit, tout point couvert par F l'est par G. Cette relation d'ordre est partielle.

  Exemple:

  $F = a.b.c$

  $G = a.b$ $\Rightarrow$ $F \leq G$

  $H = a + e$

  $I = b$ $\Rightarrow$ H et I sont incomparables.

- ***Définition I.11: Support d'une expression***

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

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

Exemple:

$F = a.b + \overline{b}.c \Rightarrow SUPP(F) = \{a,b,c\}$

- ***Définition I.12: Produit algébrique et Booléen***

Soient $F = \sum_{i}^{n} m_i$ et $G = \sum_{i}^{m} m_j'$ deux expressions polynomiales de deux fonctions Booléennes. Le produit algébrique F.G existe si F et G dépendent de deux ensembles de variables disjoints, et est défini par:

$$F.G = \sum m_i.m_j' \quad \text{pour } i = 1..n \text{ et pour } j = 1..m$$

Exemple:

$F = a + c$

$G = b + \overline{d}$

$F.G = a.b + a.\overline{d} + c.b + c.\overline{d}$

Le produit de deux expressions polynomiales dont les ensemble des variables ne sont pas disjoints est appelé produit Booléen et utilise les règles habituelles de l'algèbre de Boole ($a.\overline{a} = 0$; $a.a = a$).

Exemple:

$F' = a.c + b.\overline{e}$

$G' = a.e$

Le produit Booléen de l'expression F' par G' est le suivant:

$F'.G' = a.c.a.e + a.b.\overline{e}.e = a.c.e + 0 = a.c.e$

- ***Définition I.13: Diviseur algébrique et Booléen***

D est un diviseur algébrique de F si:

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

il n'existe pas H' tel que $H < H'$ ($H < H' \Leftrightarrow H \leq H'$ et $H \neq H'$) et $F = D.H' + R$, R étant une expression polynomiale.

Le quotient H et le reste R sont uniques.

Exemple:

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

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

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

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

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

Exemple:

$F = a.b + c + d$

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

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

- ***Définition I.14: Facteur algébrique et Booléen***

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

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

- ***Définition I.15: Substitution algébrique***

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

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

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

Exemple:

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

$G = a + b + c$

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

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

- ***Définition I.16: Expression libre***

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

  Exemples:

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

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

- ***Définition I.17: Noyaux algébriques***

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

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

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

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

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

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

Remarques:

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

– Un noyau local peut avoir plusieurs conoyaux associés.

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

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

Exemple:

Soit F_1 une fonction dont l'expression polynomiale est:

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

Les couples conoyaux/noyaux de F_1 sont:

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

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

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

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

- ***Définition I.18: Portée d'un noyau***

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

K: est un noyau de l'expression F.

C: est le conoyau associé au noyau K.

$\mathfrak{E}$: ensemble des monômes résultat du développement de C.K.

Exemple:

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

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

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

$\mathfrak{E}$ = {a.b.c, a.b.d} est l'ensemble des monômes de C.K écrit sous forme polynomiale.

Le triplet (C,K,$\mathfrak{E}$) définit la portée dans F de K.

- ***Définition I.19: Compatibilité algébrique***

  Soient deux couples de conoyaux/noyaux (C,K) et (C',K') d'une expression Booléenne F. Soit $\mathfrak{E}$ (resp.$\mathfrak{E}$') 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:

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

  Exemple:

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

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

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

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

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

  $\mathfrak{E}$ = {a.b.c, a.b.d}

  $\mathfrak{E}$' = {a.b.d, a.c.d}

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

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

I.2 FACTORISATION

I.2.1 Objectif de la factorisation

La factorisation est un processus qui part d'un ensemble de fonctions Booléennes exprimées par des sommes de monômes et le transforme en un ensemble d'expressions factorisées. En général la complexité d'un circuit combinatoire est étroitement liée au nombre de littéraux des formes factorisées représentant l'ensemble des fonctions Booléennes. Celui-ci est un bon estimateur de la surface active d'un circuit (surface occupée par les portes

logiques). La factorisation a donc pour objectif essentiel de minimiser la complexité des formes factorisées et donc la logique qui sert à les réaliser.

On peut associer de façon virtuelle un schéma de portes logiques ET/OU à des expressions en somme de monômes et des expressions factorisées.

La première implantation s'appelle une implantation 2 couches (figure I.1). Une première couche réalise les monômes des variables d'entrées au moyen de portes ET, et une deuxième couche réalise la fonction somme de monômes au moyen de portes OU.

Une implantation plus générale utilise plusieurs couches de logique. Elle est fondée sur des expressions Booléennes factorisées. Cette forme factorisée est ensuite décomposée en sous-fonctions correspondantes à des éléments de bibliothèque appelés "cellules standards".

Exemple:

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

$F_2 = a.f.g + b.f.g + e.f.g$

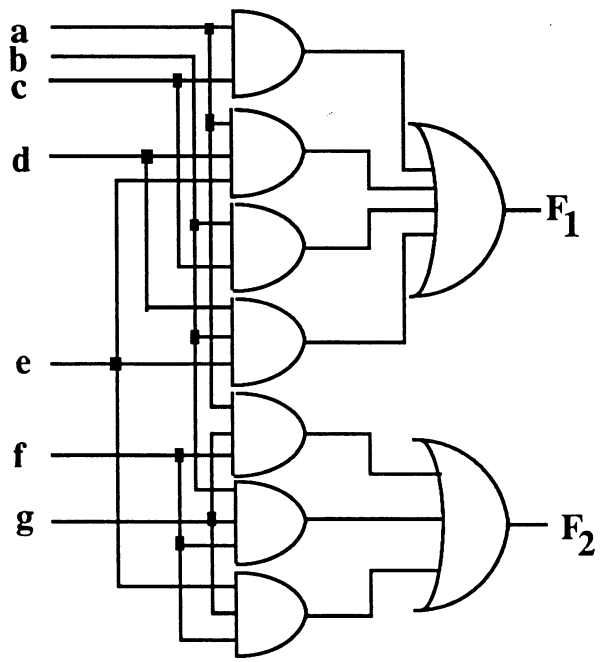


Figure I.1. Implantation deux couches.

Après factorisation, l'expression des deux fonctions Booléennes F_1 et F_2 devient:

$SF = a + b$

$F_1 = SF.(c + d.e)$

$F_2 = f.g.(SF + e)$

Une implantation multicouche utilisant uniquement des cellules réalisant des ET et OU Booléens est donnée dans la figure I.2.

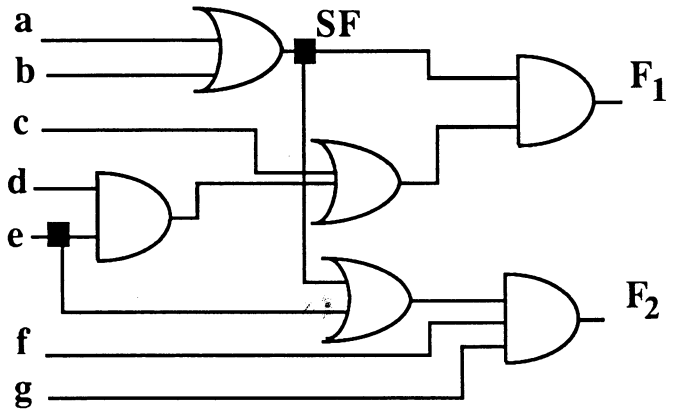


Figure I.2. Implantation multicouche.

I.2.2 Principe de la factorisation

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

On peut énumérer essentiellement deux types de factorisation:

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

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

L'exemple suivant illustre la différence entre les noyaux algébriques et les noyaux Booléens. La méthode de génération de noyaux Booléens sera exposée dans le chapitre IV.

Exemple:

Soient les deux fonctions Booléennes suivantes:

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

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

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

Les noyaux algébriques sont les suivants:

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

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

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

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

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

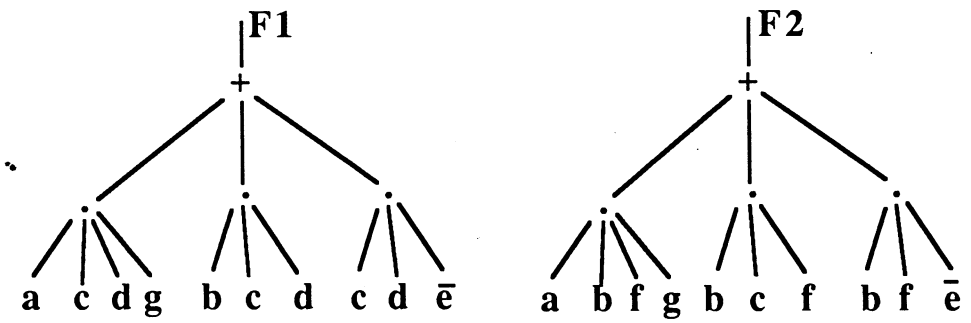


Figure I.3. Arbres de F_1 et F_2 avant la factorisation algébrique

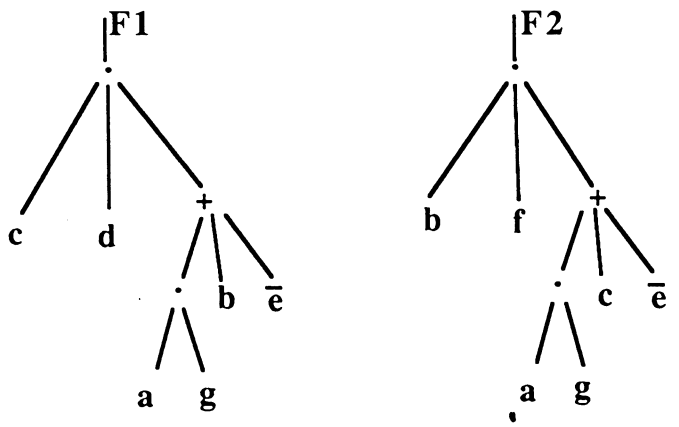


Figure I.4. Arbres après la factorisation algébrique

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

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

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

L'expression $(a.g + b.c + \overline{e})$ est commune à F_1 et à F_2 , et devient une sous-fonction commune notée SF (figure I.5).

$F_1 = c.d.SF$

$F_2 = b.f.SF$

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

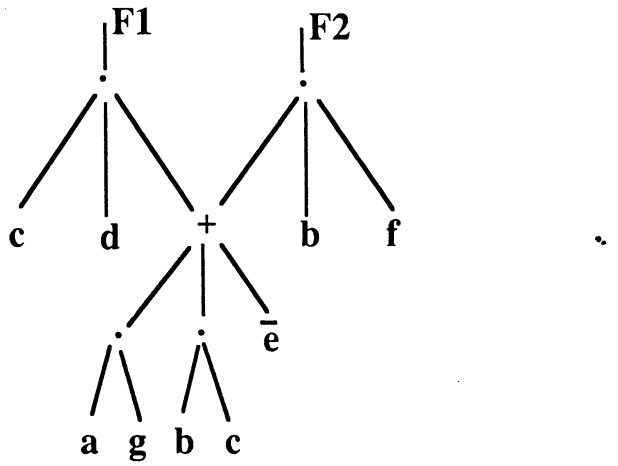


Figure I.5. Arbres après factorisation Booléenne

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

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

Une approche gourmande ou "greedy" de la factorisation consiste en la séquence d'étapes suivante:

A) Génération des candidats diviseurs.

B) Tant qu'il y a des candidats diviseurs

– Sélection des candidats de gain maximal en terme de nombre de littéraux.

– Division par les candidats sélectionnés.

Fin tant que

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

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

I.2.3 Les différents critères d'optimisation de la factorisation

De façon générale, une factorisation tend à faire décroître le nombre de littéraux des expressions Booléennes, ceci devant conduire à une diminution de la surface. En pratique, il s'agit souvent de prendre en compte d'abord l'optimisation de vitesse du circuit. Si on représente les expressions factorisées par des arbres factorisés, on cherche à diminuer la profondeur de ces arbres liée à la profondeur du circuit final en terme de cellules standards (plus long chemin entre les entrées et les sorties) et à la sortance des noeuds de l'arbre en supposant que la fonction est la racine de l'arbre. Bien sûr, le résultat dépendra de façon plus précise de la méthode de décomposition.

I.3 GÉNÉRATION DES CANDIDATS DIVISEURS

Dans la suite de ce chapitre nous nous intéressons aux candidats algébriques suivants:

• Noyaux algébriques et éventuellement leurs intersections.

• Monômes ou parties de monômes communs pour un ensemble de fonctions.

En fait, on montre facilement que la recherche de monômes ou parties de monômes communs se ramène à une recherche de noyaux. L'attention est donc particulièrement consacrée à l'étude des noyaux.

I.3.1 Génération des noyaux algébriques

L'algorithme proposé pour la génération des noyaux est celui présenté dans [Bra82,Bra87]. Cet algorithme est fondé sur l'ordonnancement des littéraux d'une fonction Booléenne, puis 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)
début
        m ← le monôme, de plus grand degré, facteur
            algébrique de F;
        noyauxList ← rnoyaux(0,F/m) ;
        si m = 1 alors noyauxList ← noyauxList + F;
fin;
rnoyaux(j,F)
début
        noyauxList ← { F };
        pour i:=j+1 jusqu'à n faire
        début
                si li apparait dans plus d'un monôme de F alors
                début
                        m ← 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));
                fin;
        fin;
        retourner(noyauxList);
fin;
```

L'indice j correspond au j^{ième} littéral dans la fonction F.

Explication de l'algorithme

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 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.

Exemple:

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 est le suivant:

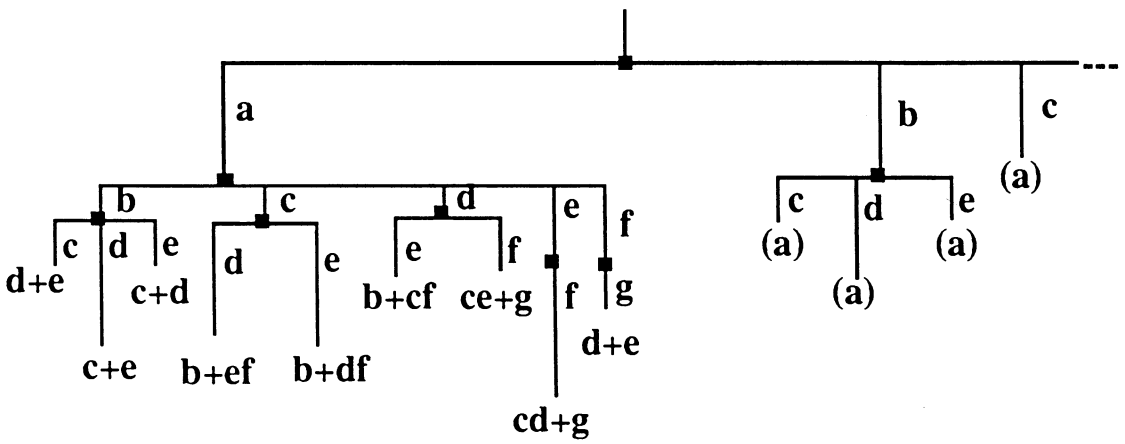


Figure I.6. Représentation des noyaux générés par l'algorithme ci-dessus

La racine de l'arbre est la fonction F qui est le noyau de 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 en largeur et en profondeur dans l'arbre.

Le "(a)" de la branche (b.c) veut dire qu'on est ramené à la branche (a) pour laquelle tous les noyaux ont été générés et par suite ceux de (a.b.c).

I.3.2 Génération des monômes ou parties de monômes communs pour un ensemble de fonctions

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

L'algorithme utilisé est le suivant:

début

- *On rajoute à chaque monôme de chaque fonction F_i le littéral v_i;*
- *On construit la fonction $\mathcal{F}$, la somme de tous les monômes (après l'ajout de v_i) de l'ensemble des fonctions Booléennes;*
- *On calcule tous les noyaux et conoyaux associés de $\mathcal{F}$;*
- *Les conoyaux où aucun littéral v_i n'apparait, correspondent à des parties communes de monômes. Si plusieurs conoyaux associés au même noyau existent, ils forment une somme de monômes communs;*

fin;

Exemple:

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

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

on construit F de la façon suivante:

$\mathcal{F} = a.b.c.d.v_1 + d.e.v_1 + h.v_1 + a.b.c.e.v_2 + d.e.v_2 + h.v_2$

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.

I.4 CALCUL DU GAIN D'UN CANDIDAT DIVISEUR

I.4.1 Gain des noyaux algébriques

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

1- Calcul de gain pour les noyaux locaux:

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

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

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

2- Calcul de gain pour les noyaux globaux:

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

$$\text{Gain}(K) = (p-1).(\text{NbLit}(K)) - p$$

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

$$\text{Gain}(K) = \sum_{i=1}^{m} ((\text{NbMon}(K) - 1).\text{NbLit}(C^i)) + (m-1).\text{NbLit}(K) + (p - 1).(\text{NbLit}(K)) - p \qquad (I.2)$$

Exemple:

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

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

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

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

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

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

I.4.2 Gain des monômes ou parties de monômes communs

On appelle:

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

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

$$Gain(m) = (NbOcc(m) - 1).NbLit(m) - NbOcc(m) \qquad (I.3)$$

Exemple:

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

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

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

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

I.5 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, exprimée en nombre de littéraux, mais 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 arête si les candidats diviseurs correspondants sont incompatibles.

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

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

Bien que la formulation de cette méthode semble intéressante, il s'avère que la complexité actuelle des circuits ne permet pas toujours de l'appliquer de manière optimale. 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 à l'utilisateur d'utiliser une méthode ou une autre, ce qui lui procure un bon compromis temps de calcul / qualité des résultats.

La deuxième méthode choisit le candidat diviseur permettant d'obtenir un gain local maximal (algorithme "greedy") sans vérifier sans compatibilité algébrique avec les autres.

I.6 DIVISION ET SUBSTITUTION ALGÉBRIQUES

I.6.1 Algorithme de la division algébrique

La division algébrique, au sens général du terme, est une méthode puissante qui permet de diviser exhaustivement tout ce qui peut l'être par le diviseur. Illustrons-le d'abord sur l'exemple suivant:

Exemple:

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

Le nombre de littéraux dans l'expression polynomiale de F est de 18, et la liste des noyaux ainsi que leur conoyaux et leur gain associés est:

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

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

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

Le nombre de littéraux dans ce cas est de 10 au lieu de 18 dans l'expression polynomiale de F.

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 $M_H \supseteq C$.

Principe de la division algébrique:

Soient 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.

La division s'effectue comme suit:

- Pour tout monôme m_i du diviseur D
  - Faire la somme des monômes de F multiples de m_i.
  - Diviser cette somme par m_i.
  - soit $V_{m_i} = \sum m'_j$ le quotient de cette division.
- L'intersection des $M_{V_{m_i}}$ constitue l'ensemble des monômes du quotient H de la division de F par D.

- Le reste R est constitué des monômes restants non impliqués dans la division $M_R = M_F - M_{D.H}$.

L'implantation algorithmique de cette division algébrique a été introduite par Brayton [Bra87b]:

Division_Algébrique(F,D)

début

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 $j^{ième}$ monôme de F */*

$V_i = \{ v_j \in V / \ u_j = d_i \}$

/ 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 est l'ensemble obtenu en divisant F par u_i */*

$M_H = \cap M_{V_i}$

/ 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)

fin;

Exemple:

Soit F une fonction de 5 monômes:

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

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:

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

$\Downarrow$

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

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

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

$M_R = a.b.k$

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

Et 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 de 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'occurences 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 donc d'effectuer la procédure de division algébrique.

I.6.2 Algorithme de la substitution 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:

début

- *division algébrique de F par le candidat diviseur D → (H,R)*
- *calcul du complément de D*

- *division algébrique du reste par* $\overline{D} \rightarrow (H',R')$
- *utiliser une nouvelle variable x afin de représenter le diviseur D*
- *substitution dans F:* $F = H.x + H'.\overline{x} + R'$

fin;

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. Le coût du complément est difficile à évaluer: Le pire des cas est celui des OUs exclusifs où la couverture à 1 de la fonction contient $2^{(v-1)}$ points et où v est le nombre de variables, donc exponentielle. Cependant, en général dans la pratique le coût est acceptable.

Exemple:

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

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

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

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

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

$F = (d + e).x + f.\overline{x} + g$

$x = a + b + c$

Chapitre II

Proposition d'une Méthode de Division Restreinte par Conoyaux

II.1 INTRODUCTION

Cette étude a comme point de départ, la constatation de difficulté de contrôler la connectique et le chemin critique des circuits après l'application des méthodes de factorisation algébrique classiques, et ceci à partir d'un certain seuil de complexité. Comme nous l'avons vu précédemment, la division algébrique associée à la substitution algébrique permet de diviser par des noyaux, parties de noyaux ainsi que par le complément de ces diviseurs. Ceci signifie que plusieurs parenthèses sont créées, indiquant la création de sous-fonctions éventuellement partagées. Ceci pourrait avoir 3 inconvénients: le premier réside en un nombre important des couches logiques, ce qui rendra difficile la minimisation ultérieure de chemin critique, le deuxième pourra être un mauvais contrôle de la connectique car la mise en place de facteurs communs entraîne une connectique plus riche, le troisième est une mauvaise maîtrise du choix des candidats diviseurs dans l'approche heuristique. En effet le choix d'un candidat diviseur entraîne dans le cas classique le choix d'autres candidats (partie de noyau, complément d'un candidat). Ces choix masquent ou empêchent le choix d'autres candidats "ultérieurs" dont le gain aurait pu être supérieur.

Ce chapitre propose 2 éléments d'une approche alternative plus efficace pour les blocs combinatoires complexes. D'une part la division algébrique sera restreinte à une division par conoyau et non plus à une division par noyau. Ceci empêche la division par les parties de noyaux et l'extension aux facteurs complémentaires. D'autre part, une attention particulière sera portée à la sélection et au filtrage plus poussé des candidats diviseurs. Ce choix étant guidé par la réduction ultérieure de la connectique.

II.2 MÉTHODE DE DIVISION RESTREINTE

II.2.1 Principe général

La division algébrique développée précédemment est une méthode puissante qui permet d'effectuer la factorisation d'un ensemble de fonctions Booléennes quelques soient les candidats diviseurs choisis (noyaux et leurs intersections, monômes et plus généralement toutes expressions non libres). La méthode de

division restreinte, proposée ici, est fondée sur la notion de la portée d'un noyau (C,K,$\mathfrak{E}$) (définition I.18). Cette portée est localisée sur l'ensemble des monômes donnant naissance au noyau.

Principe de la division restreinte par les conoyaux:

Supposons qu'un noyau candidat K_i soit sélectionné suivant la fonction gain définie dans §I.4.1. Soient les triplets $(C_i^j, K_i, \mathfrak{E}_i^j)$ associés à ce noyau. Le résultat de la division restreinte s'écrit sous la forme suivante:

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

Où le reste R est la somme de monômes restants de F ($M_R = M_F - \mathfrak{E}_i^j$).

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

Soit la fonction F (cf § I.6.1) telle que:

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

La liste des noyaux, conoyaux et gains associés est:

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

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

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

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

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

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

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

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

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

Division_par_Conoyaux(F,Cj)

début

pour tous les conoyaux Cj associés à un noyau K faire

$\mathfrak{E}_j$ = *Ensemble des monômes de F multiples de Cj*

finpour

$M_{\mathfrak{E}} = \cup M_{\mathfrak{E}_j}$

$M_R = M_F - M_{\mathfrak{E}}$ /* *R est le reste de la division de F par les conoyaux Cj* */

$F = \left[\sum_j Cj\right].K + R$ /* *mise en commun de K* */

fin;

Cette approche peut être illustrée par l'arbre factorisé (figure II.1) construit à partir des triplets $(C_i^j, K_i, \mathfrak{E}_i^j)$ déterminés par l'algorithme de recherche de noyaux. L'arbre factorisé est obtenu itérativement à chaque étape.

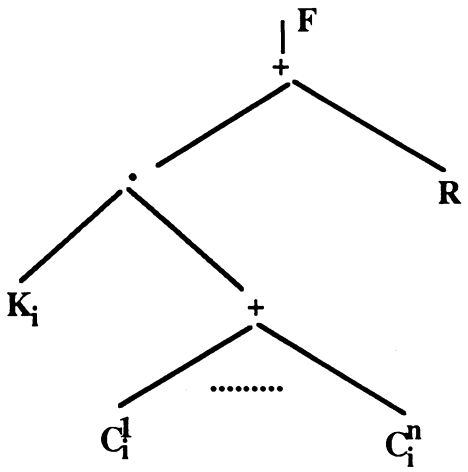


Figure II.1. Arbre factorisé après la mise en facteur de K_i

Exemple:

La fonction factorisée dans l'exemple précédent est représentée sous la forme d'un arbre (figure II.2).

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

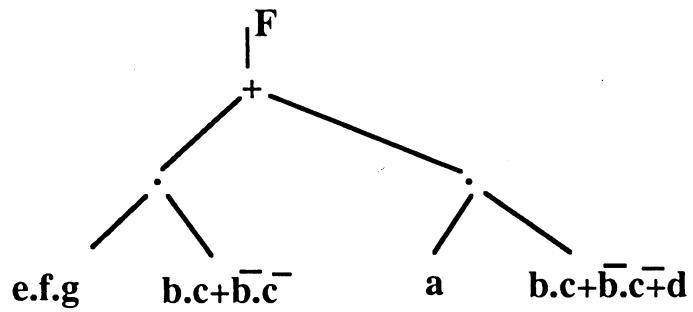


Figure II.2. Arbre factorisé de F.

II.2.2 Comparaison du gain de la division restreinte à celui de la division algébrique classique

Soit F une fonction Booléenne et supposons qu'à une étape donnée, un noyau K ait été choisi. Soient C^i les conoyaux associés à K. Si on applique la méthode de division par les conoyaux, que l'on note méthode1, le gain est le suivant:

$$Gain_{méthode1}(K) = \sum_{i=1}^{I} (NbMon(K)-1).NbLit(C^i) + (I-1).NbLit(K)$$

Si on applique la méthode de division et de substitution algébriques (méthode2), le gain devient:

$$Gain_{méthode2}(K) = Gain_{DivNo}(K) + Gain_{DivNoComp}(K) + Gain_{global}(K)$$

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

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

Les $(C^j)_{j=1..J}$, tels que $(J \geq I)$, sont les cofacteurs du noyau K. Les $(C^i)_{i=1..I}$ sont les conoyaux de K et les $(C^k)_{k=I+1..J}$ sont les conoyaux associés aux noyaux K_l où $K \subset K_l$, d'où:

$$Gain_{DivNo}(K) \geq Gain_{méthode1}(K)$$

La différence de gain entre $Gain_{DivNo}(K)$ et $Gain_{méthode1}(K)$ est:

$$Gain_{DivNo}(K) - Gain_{méthode1}(K) = \sum_{k=I+1}^{J} (NbMon(K)-1).NbLit(C^k) + (J-I).NbLit(K)$$

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

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

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

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

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

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

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

L'estimation de la fonction gain est exacte (celle calculée au moment de la génération des noyaux) dans la méthode de division par les conoyaux c'est celle calculée dans $Gain_{méthode1}(K)$. En ce qui concerne la division et la substitution algébriques, le $Gain_{méthode2}(K)$ ne peut être estimé au stade de la génération des noyaux, la valeur exacte ne peut être calculée que si on fait effectivement la division et la substitution algébriques ce qui ne peut être envisageable pour tout noyau candidat. On se contente donc de faire la sélection du noyau K selon $Gain_{méthode1}(K)$ qui constitue une limite inférieure de la fonction gain.

II.3 SÉLECTION ET FILTRAGE DES CANDIDATS DIVISEURS

II.3.1 Sélection des candidats diviseurs

Comme annoncé précédemment, les méthodes de factorisation étudiées sont de type greedy. Elles sélectionnent le candidat permettant d'obtenir un gain local maximal et éliminent de la liste des candidats ceux qui ne sont plus compatibles après le choix de ce dernier. Cette mise à jour permet de garder uniquement les candidats diviseurs compatibles algébriquement avec ceux déjà choisis.

Exemple de sélection de conoyaux/noyaux:

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

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

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

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

La méthode "greedy" se déroule de la façon suivante:

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

L'expression factorisée de F est:

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

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

début

Tant qu'il existe des candidats diviseurs faire

{candidats} ← *{noyaux globaux ou locaux}.*

ou

{candidats} ← *{monômes et parties de monômes communs}.*

tri rapide des candidats par gain décroissant.

choix du meilleur candidat.

division par le meilleur candidat sélectionné.

mise à jour de la liste des candidats diviseurs.

fin tant que

fin;

II.3.2 Critère de sélection des candidats diviseurs pour l'amélioration de la connectique

Dans l'exemple du paragraphe précédent, le choix des candidats diviseurs s'est porté sur ceux qui avaient le meilleur gain, exprimé en terme de nombre de littéraux. En fait, étant donné que l'objectif qu'on s'est fixé est l'amélioration de la structure de la connectique, ce gain est pondéré par un facteur lié directement à la sortance des portes logiques associées à ces candidats. En effet, dans le cas de candidats ayant un gain équivalent en terme de nombre de littéraux, le choix du meilleur d'entre eux portera sur celui qui affecte le moins de fonctions possibles. La réduction du nombre de fonctions affectées par les sous-fonctions choisies permet de minimiser la sortance

moyenne des portes logiques dans le réseau de cellules et par conséquent la structure de la connectique.

II.3.3 Filtrage des candidats diviseurs

Le filtrage des candidats diviseurs est fait selon deux critères. Le premier critère concerne l'optimisation de la structure de la connectique en limitant la taille des sous-fonctions choisies, le deuxième concerne l'optimisation de la profondeur des fonctions Booléennes factorisées.

II.3.3.1 Limitation de la taille des sous-fonctions

Ce premier critère permet de choisir dans l'ensemble des candidats diviseurs lors de la phase de génération ceux dont la taille, en terme de nombre de littéraux, est supérieure à une valeur donnée k. Les expériences montrent que la taille critique des candidats diviseurs pour une implantation sur des bibliothèques classiques de cellules standards est de 3. Les sous-fonctions dont la taille est inférieure à cette valeur sont rejetées lors de la phase de génération de candidats diviseurs. En effet, ces sous-fonctions sont en général partagées par un grand nombre de fonctions. Les cellules correspondantes risquent d'avoir une sortance très élevée et amènent une augmentation importante de la surface de routage. Le filtrage des sous-fonctions acceptées permet un bon contrôle de la surface occupée par la connectique et une bonne préparation du chemin critique.

II.3.3.2 Optimisation de profondeur des fonctions Booléennes

En partant d'un ensemble de fonctions Booléennes représentées sous forme polynomiale, les candidats diviseurs sont choisis de manière qu'ils n'augmentent pas la profondeur initiale de l'ensemble des fonctions. Cette profondeur est calculée en fonction de la bibliothèque de cellules standards utilisée, ou plus exactement en fonction de la cellule de bibliothèque ayant le fanin (entrance) le plus important. L'idée de base est que si un noeud OU ou ET ayant un fanin égal à FI, la profondeur minimale de ce nœud est égale à $\lceil \log_k FI \rceil$, k étant le plus grand fanin des cellules de la bibliothèque. D'une manière générale, le calcul de la profondeur d'un nœud OU ou ET dans un arbre factorisée est fait de la manière suivante:

Si on désigne par $p_1, p_2, ..., p_n$ les profondeurs des nœuds successeurs du nœud N, la profondeur de N est:

$$p_N = \lceil \log_k \sum_{i=1}^{n} k^{(p_i)} \rceil$$

Cette profondeur est obtenue en remplacant le nœud N par un sous-arbre de nœuds OU ou ET tel que leur fanin est limité à k [Sak93].

Ce critère de filtrage pendant la phase de factorisation ne sera utilisée que dans le cas de la comparaison avec les résultats du système SIS dans le dernier chapitre.

II.4 DISCRÉTISATION DES ÉTAPES DE FACTORISATION

Comme on l'a vu précédemment, la factorisation d'un ensemble de fonctions Booléennes recherche plusieurs types de candidats diviseurs (noyaux, monômes et parties de monômes communs, intersection de noyaux, etc...).

La méthode de factorisation présentée ici considère deux types de candidats: les noyaux et les monômes et parties de monômes communs.

Une méthode de discrétisation, traitant alternativement des sous-ensembles de noyaux et de monômes et parties de monômes communs, utilise l'algorithme suivant:

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

début

$S = s_0$

Tant qu'il existe des candidats faire

*/*Division par les conoyaux ou les noyaux et leurs compléments*/*

Division par des candidats (gain des noyaux ≥ S)

*/*Recherche des monômes ou parties de monôme communs*/*

Recherche de parties communes de gain ≥ S

$S \leftarrow Succ\ (S)$

fin tant que

fin;

Dans l'algorithme de factorisation nous avons adopté une suite de valeurs du gain obtenue d'une façon empirique. Cette suite correspond aux valeurs suivantes:

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

Etant donné que les deux types de candidats choisis sont confrontés à une éventuelle incompatibilité algébrique, cette méthode de discrétisation ne favorise aucun type de candidats et donne une bonne vision globale. L'exemple suivant illustre l'incompatibilité qui peut avoir lieu entre les deux types de candidats diviseurs.

Exemple:

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

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

Les noyaux des deux fonctions Booléennes sont:

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

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

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

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

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

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

$SF = a.b.c.d$

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

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

II.5 DOMAINE D'APPLICATION DE LA DIVISION RESTREINTE

II.5.1 Classification des circuits tests

Pour permettre une bonne évaluation des différentes méthodes de factorisation développées, les circuits ont été classés en trois catégories:

- Circuits de faible complexité (nombre de littéraux initial ≤ 800)
- Circuits de moyenne complexité (800 ≤ nombre de littéraux ≤ 3000 à 3500)
- Circuits de haute complexité (nombre de littéraux ≥ 3500)

Le Tableau II.1 montre les différentes caractéristiques des circuits tests utilisés. Pour les comparaisons entre la méthode de division restreinte par les conoyaux et la méthode de division et substitution algébriques, on a considéré uniquement les deux premières catégories de circuits tests. Les évaluations et les comparaisons entre la méthode de division restreinte et la méthode de factorisation lexicographique détaillée dans le troisième chapitre se feront sur les circuits de moyenne et haute complexité.

Benchmarks	nb d'entrées	nb de sorties	nb de littéraux
circuits de faible complexité			
bbtas	5	5	36
dk17	5	6	97
dk15	5	7	115
rd53	5	3	144
donfile	7	6	166
misex2	25	18	188
z4ml	7	4	252
kirkman	17	11	267
bbsse	11	11	281
5xp1	7	10	296
f51m	8	8	319
bw	5	28	348
sao2	10	4	495
frg1	28	3	792
vg2	25	8	804
rd73	7	3	840
Circuits de moyenne complexité			
s1	13	11	851
jay	10	39	858
platcos	88	136	873
planet	13	25	1010
styr	14	15	1023
planet1	13	25	1084
tbk	11	8	1210
sand	16	14	1426
scf	34	63	2190
cache_ctr	34	62	3690
Circuits de haute complexité			
zeegers	18	18	4531
hyeti2	30	69	6246
is	46	57	6563
hyeti1	35	80	9829
imec5	14	84	10995
apex2	39	3	14728
imec9	27	102	15321
seq	41	35	17262
imec7	17	146	20423
imec4	26	135	26723
imec3	26	151	32053
imec2	27	125	41074

Tableau II.1. Caractéristiques des circuits tests de faible, moyenne et haute complexité utilisés dans les comparaisons des différentes méthodes de factorisation.

II.5.2 Facteur de routage

Les expériences faites sur les circuits tests permettent de mesurer, non seulement la surface active du circuit représentée par la surface des cellules, mais aussi la surface du routage nécessaire pour réaliser les interconnexions entre ces cellules après placement routage. Le facteur de routage F_r est défini comme le rapport de la surface de routage S_r sur la surface active S_a:

$$F_r = \frac{S_r}{S_a}$$

avec $S_T = S_r + S_a$ S_T étant la surface totale du circuit

Les expériences faites sur un grand nombre de circuits tests montrent l'importance du facteur du routage dans le circuit final. Ce facteur est étroitement lié à la complexité du circuit.

Pour les circuits de faible complexité, le facteur de routage mesuré après une synthèse par des outils du commerce est faible, généralement inférieur à 1. Dans ce cas, la surface de routage est plus petite que celle des cellules. C'est la surface active qui influence donc en premier les résultats en surface globale. Dans le cas de circuits de moyenne et grande complexité, le facteur de routage est plus important (>1) et le poids de la surface de routage est dominateur.

II.5.3 Environnement d'expérimentations

Les comparaisons ont été faites dans le cadre du système de synthèse logique ASYL. La cible technologique utilisée est une bibliothèque de cellules standards, à savoir la bibliothèque VSC370 de la société VLSI technology Inc. en technologie CMOS 1.2µ. Les étapes de factorisation et décomposition technologique [Cra91] ont été faites par le système ASYL et l'étape de placement routage est faite par les outils de CAO de la société COMPASS. Les résultats sont donnés en termes de surface globale et de chemin critique après estimation des capacités d'interconnexion obtenue après la phase de placement routage.

Les options de décomposition en cellules standards utilisées dans les expérimentations sont:

• Décomposition orientée surface minimisant le nombre de cellules dans le circuit final.

• Décomposition orientée vitesse ou chemin critique du circuit.

II.5.4 Comparaison des résultats des deux approches

Dans les tableaux de résultats donnés en annexe, la surface est exprimée en "mils square" (1 mils square = 645,1 μ^2) et le chemin critique est exprimé en nanoseconde (10^{-9} seconde).

La méthode de division par les conoyaux est en moyenne 3 fois plus rapide que la méthode de division et substitution algébriques. Ceci est dû au calcul du complément du noyau et à la division algébrique par le noyau et son complément dans la boucle principale de la procédure de factorisation.

II.5.4.1 Circuits de faible complexité

Pour les circuits de faible complexité, la méthode de division et de substitution algébriques donne de meilleurs résultats par rapport à la méthode de division par les conoyaux en surface active. Le gain est évalué selon la formule:

$$\text{Gain } \% = 100 * \frac{\text{résultat de la division restreinte - résultat de la division algébrique}}{\text{résultat de la division restreinte}}$$

Ce gain en surface active atteint dans le meilleur des cas (bbsse) environ 16% (Annexe, tableau A.4) et en moyenne, le gain est d'environ 3%. La méthode de division et de substitution algébriques reste meilleure en terme de surface globale et chemin critique (en moyenne 3% pour la surface globale et 6% pour le chemin critique) (Annexe, tableau A.1).

Les figures II.3 et II.4 montrent les différents gains pour la surface globale et le chemin critique. Tous les points situés au-dessus de la droite de gain nul de chaque figure indique un meilleur résultat de la méthode de division et substitution algébriques comparée à celle de la division restreinte. Les figures II.3 (resp. II.4) concernent une optimisation en surface (resp. en vitesse).

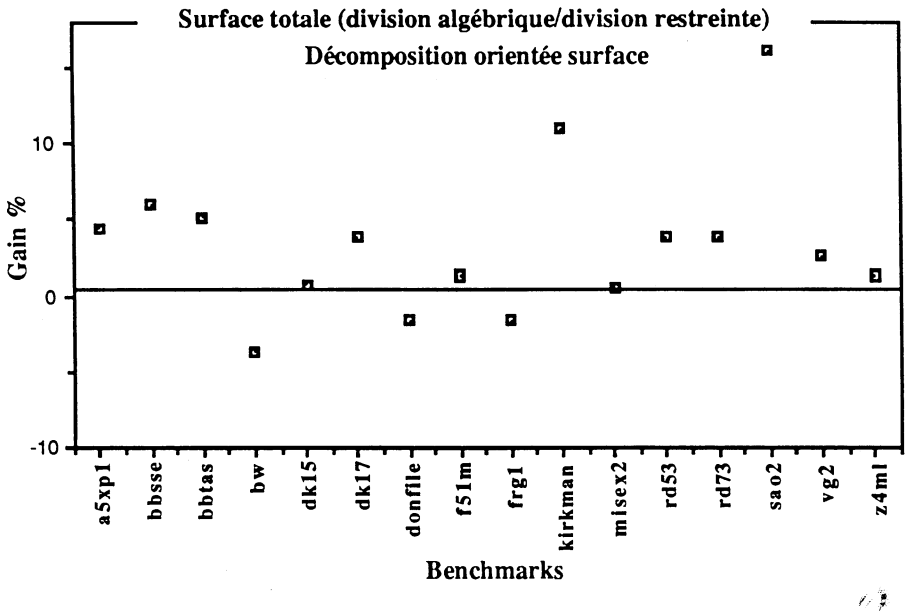


Figure II.3a. Représentations des gains en surface globale de la méthode de division et substitution algébriques par rapport à la division restreinte pour une optimisation en surface (circuits de faible complexité)

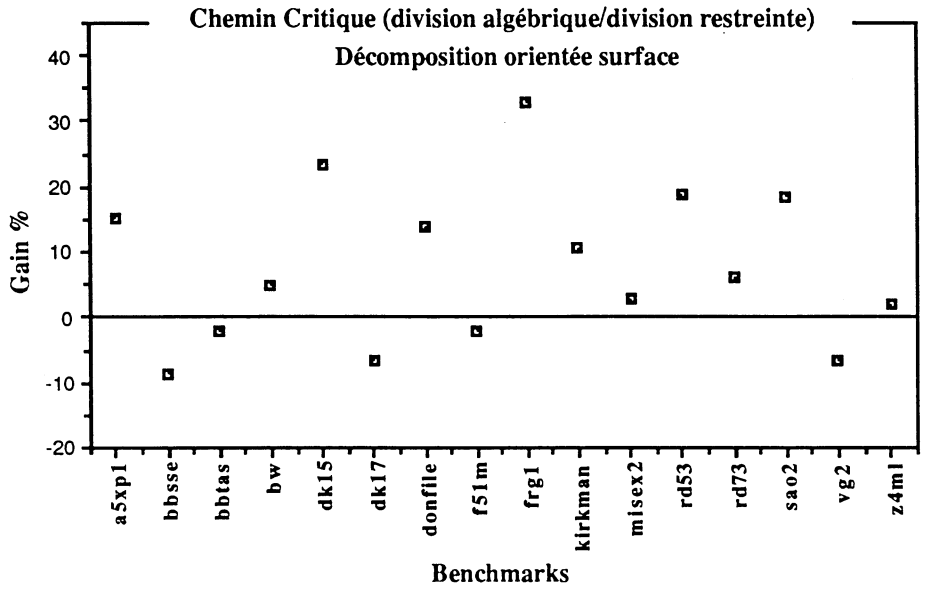


Figure II.3b. Représentations des gains en chemin critique de la méthode de division et substitution algébriques par rapport à la division restreinte pour une optimisation en surface (circuits de faible complexité)

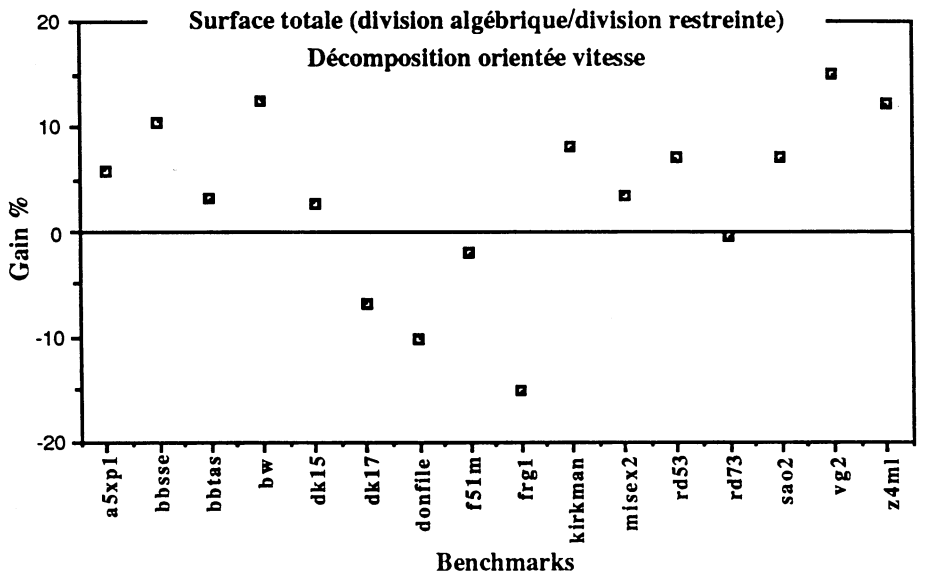


Figure II.4a. Représentations des gains en surface globale de la méthode de division et substitution algébriques par rapport à la division restreinte pour une optimisation en vitesse (circuits de faible complexité)

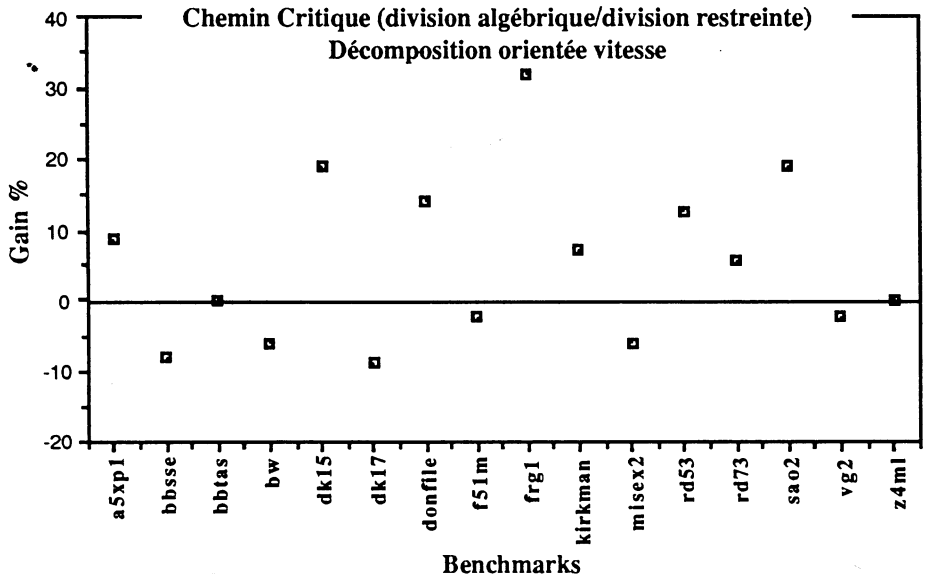


Figure II.4b. Représentations des gains en chemin critque de la méthode de division et substitution algébriques par rapport à la division restreinte pour une optimisation en vitesse (circuits de faible complexité)

II.5.4.2 Circuits de moyenne complexité

Pour les circuits de moyenne complexité, la division par les conoyaux est plus performante en terme de surface globale et chemin critique. Le gain est calculé selon la formule donnée dans le paragraphe précédent. Tous les points situés au-dessus de la droite de gain nul de chaque figure (II.5 et II.6) indique un meilleur résultat de la méthode de division restreinte comparée à celle de la division et substitution algébriques. Dans le cas d'une optimisation en surface, Le gain en surface globale est de l'ordre 3%. Quant au gain en chemin critique, il est de l'ordre de 13% (Annexe, tableau A.5). Pour une optimisation en chemin critique, le gain en surface et en chemin critique atteint respectivement 6% et 4% environ (tableau A.7)

Ces différents gains sont représentés dans les figures II.5 et II.6. Ces figures correspondent respectivement à une optimisation en surface et en vitesse.

Les résultats obtenus sont très important, car ils permettent de mettre en évidence qu'une méthode restreinte de division algébrique peut donner de bons résultats. Le gain en chemin critique est significatif car les améliorations sont généralement très difficiles à obtenir. Cela est d'autant plus important que le temps de calcul est largement inférieur à celui d'une division algébrique classique.

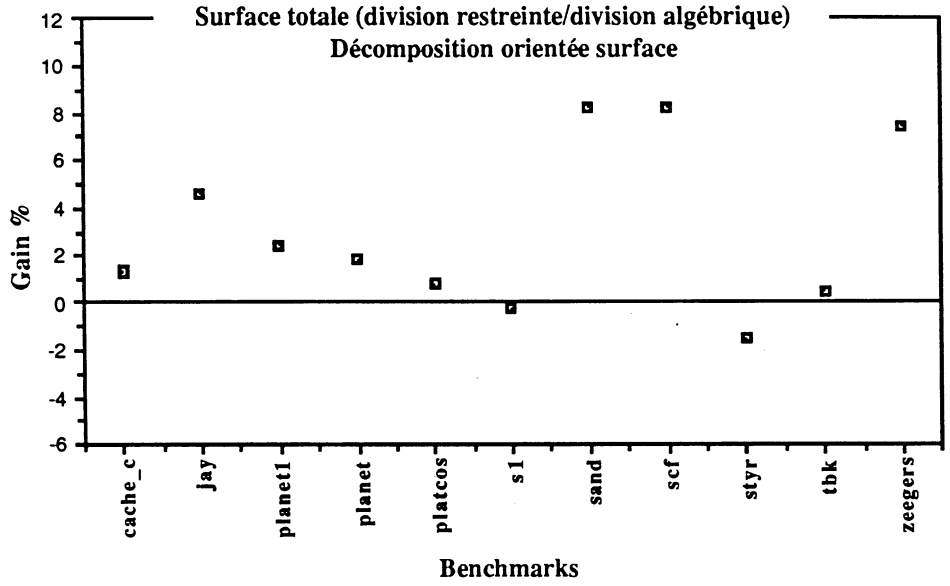


Figure II.5a. Représentations des gains en surface globale de la méthode de division restreinte par rapport à la division et substitution algébriques pour une optimisation en surface (circuits de moyenne complexité)

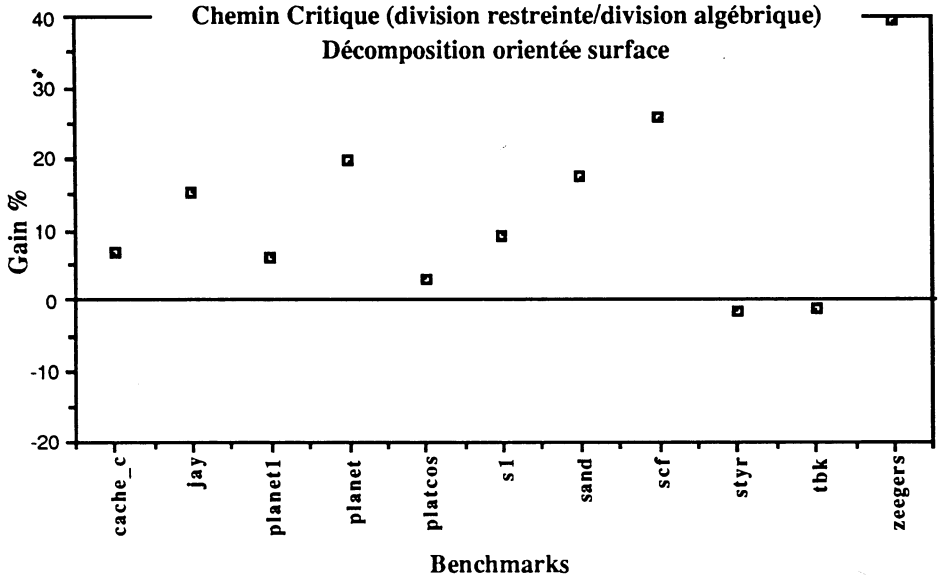


Figure II.5b. Représentations des gains en chemin critique de la méthode de division restreinte par rapport à la division et substitution algébriques pour une optimisation en surface (circuits de moyenne complexité)

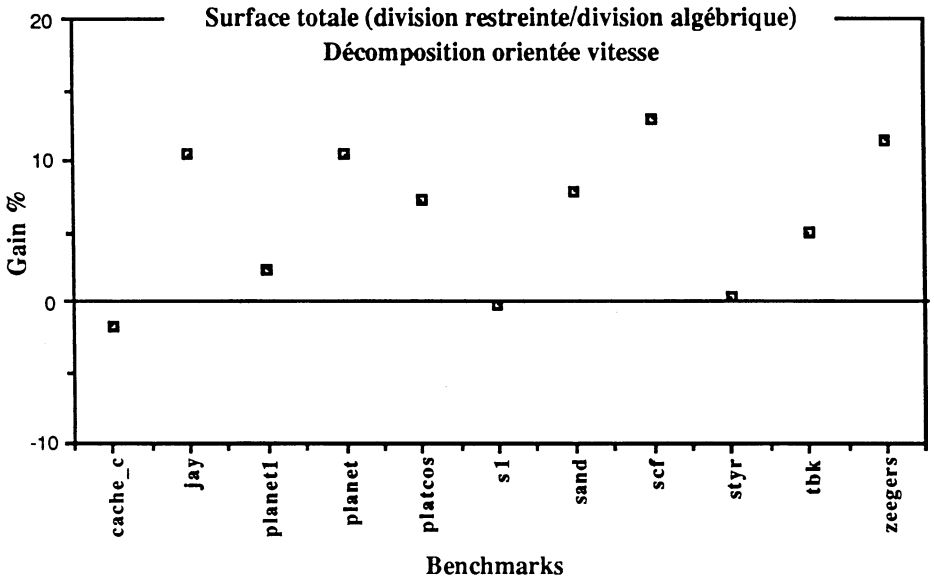


Figure II.6a. Représentations des gains en surface globale de la méthode de division restreinte par rapport à la division et substitution algébriques pour une optimisation en vitesse (circuits de moyenne complexité)

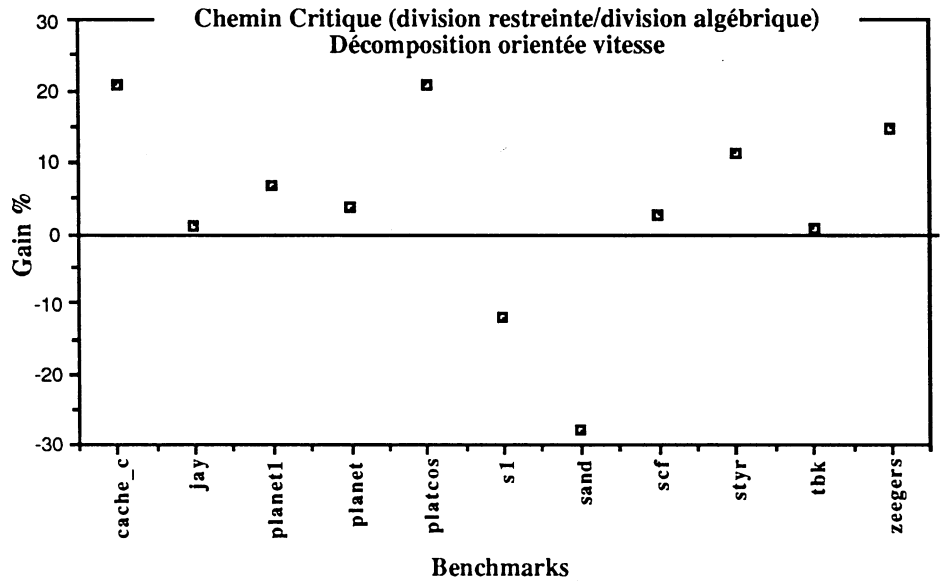


Figure II.6b. Représentations des gains en chemin critique de la méthode de division restreinte par rapport à la division et substitution algébriques pour une optimisation en vitesse (circuits de moyenne complexité)

La performance de la méthode de division par les conoyaux est atteinte grâce à la sortance relativement faible par rapport à celle de la méthode de

division et substitution algébriques. Comme on l'a signalé précédemment, dans la deuxième approche on favorise plus de logique de petite taille partagée, donc une sortance plus importante. Les expériences faites dans ce sens montrent une nette amélioration de la surface du routage dans le cas d'une division restreinte par les conoyaux. Le gain du facteur de routage dans ce cas est en moyenne de 13% (Annexe, tableaux A.6 et A.8), ce qui permet de compenser une perte en surface active de l'ordre de 4%. Le calcul de la sortance moyenne dans les réseaux de portes, qui est égale à la moyenne des sortances de l'ensemble des portes dans le circuit, montre une amélioration de 5 à 10% par rapport à celle de la division algébrique. Cela permet d'expliquer l'amélioration de la surface occupée par la connectique dans le cas d'une division restreinte.

Les figures II.7, II.8 et II.9 montrent les différents gains obtenus pour le facteur de routage, la surface de routage et la surface active des circuits tests de moyenne complexité dans le cas d'une optimisation en surface. Dans le cas d'une optimisation en vitesse, les résultats étant identiques que dans le cas précédent, ils seront donnés uniquement en annexe (tableau A.8).

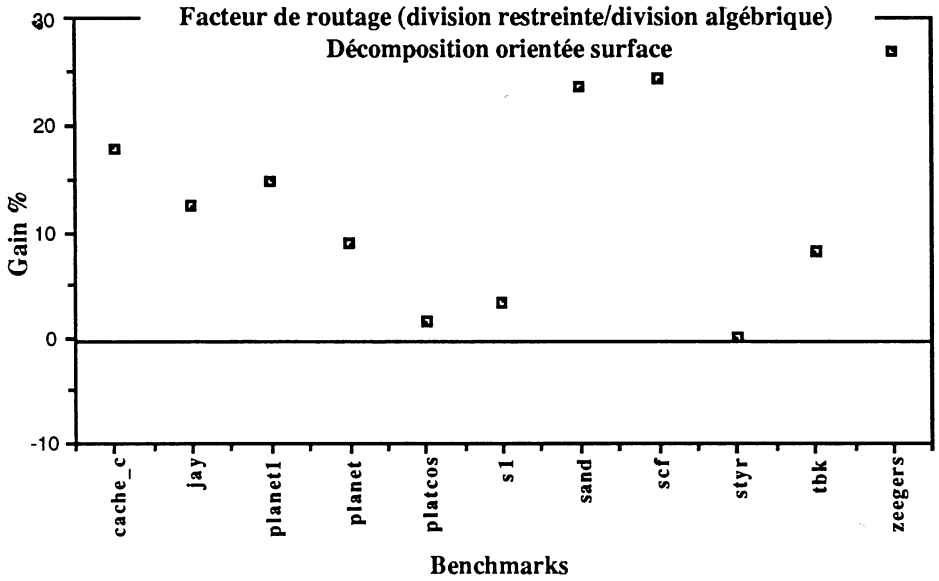


Figure II.7. Représentation des gains en facteur de routage de la méthode de division restreinte par rapport à la division et substitution algébriques pour une optimisation en surface (circuits de moyenne complexité)

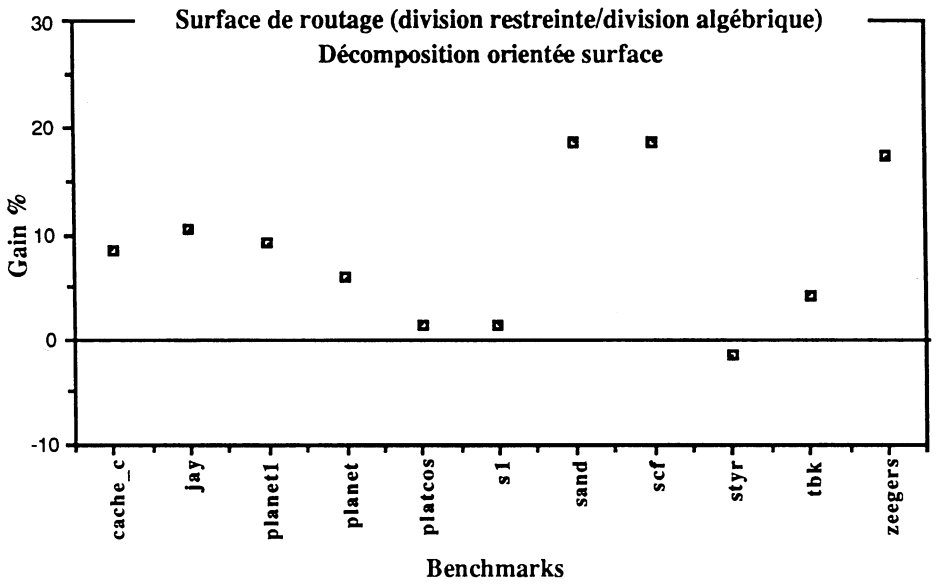


Figure II.8. Représentation des gains en surface de routage de la méthode de division restreinte par rapport à la division et substitution algébriques pour une optimisation en surface (circuits de moyenne complexité)

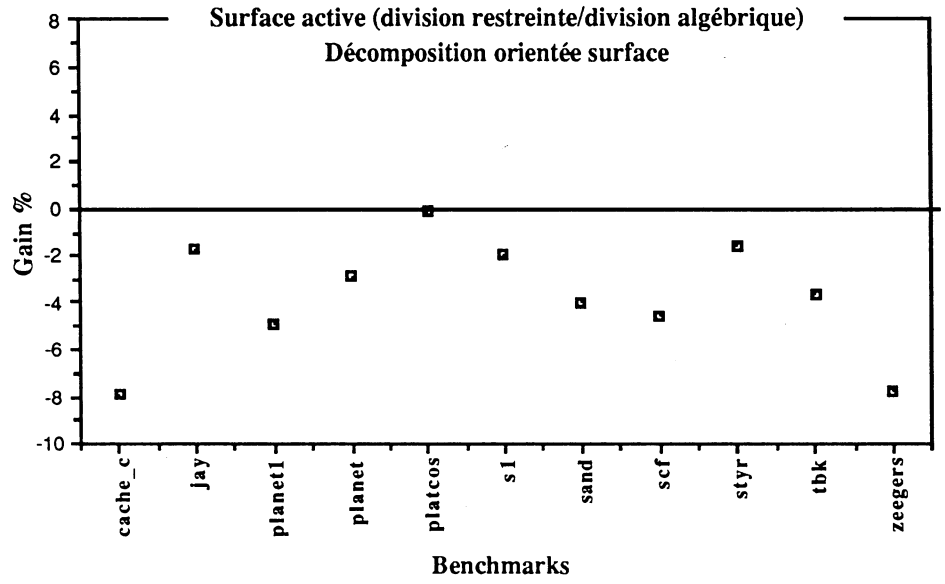


Figure II.9. Représentation des pertes en surface active de la méthode de division restreinte par rapport à la division et substitution algébriques pour une optimisation en surface (circuits de moyenne complexité)

La figure II.10 montre l'impact de la logique partagée de petite taille sur le facteur de routage et sur la surface de routage. Nous avons rapporté ici les

résultats obtenus avec ou sans le filtre cité précédemment pour la méthode de division et substitution algébriques, nous constatons une amélioration systématique du facteur de routage (en moyenne de 16%) qui peut atteindre jusqu'à 30% environ dans le cas du circuit test zeegers. Une amélioration du chemin critique est également enregistrée, elle est de l'ordre de 5% en moyenne.

Bien que les surfaces globales sont identiques dans les 2 cas (tableau A.9), on peut dire que les résultats de l'application de ce filtre sont intéressants de point de vue performance. Une amélioration importante du chemin critique pour la plupart des circuits tests est obtenue grâce à la réduction de la sortance moyenne dans circuits.

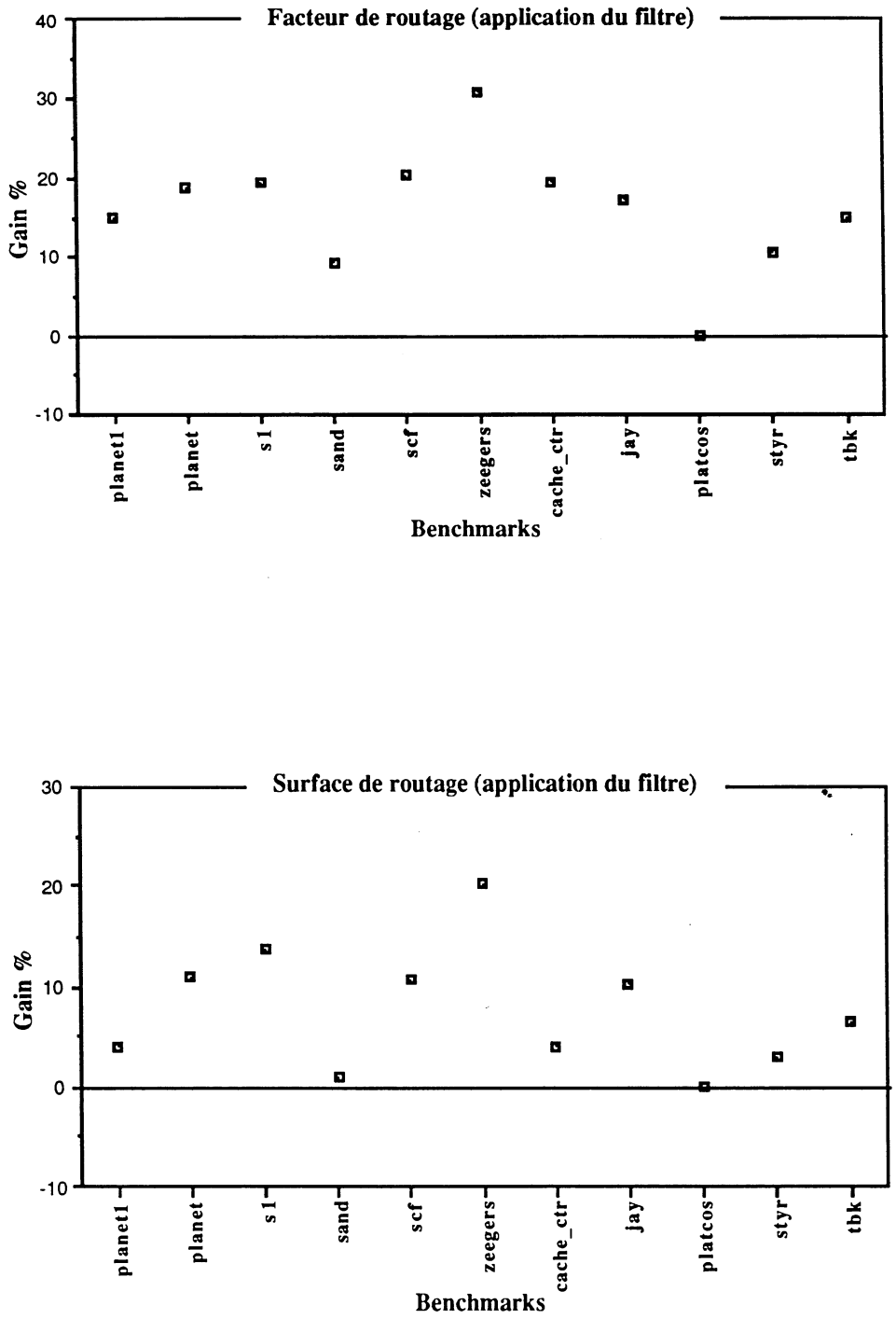


Figure II.10. Représentation des gains du facteur de routage et de la surface de routage par l'application du filtre pour la méthode de division et substitution algébriques (circuits de moyenne complexité)

II.6 CONCLUSION

D'après les résultats obtenus, la méthode de division et de substitution algébriques est plus intéressante pour les circuits de faible complexité d'autant plus que l'augmentation du temps de calcul par rapport à la méthode de division par les conoyaux n'est pas très pénalisante.

Il est confirmé que la méthode de factorisation restreinte est particulièrement adaptée aux circuits de moyenne complexité. Elle donne de meilleurs résultats en un temps d'exécution plus court.

L'introduction du filtre pour la méthode de division et substitution algébriques, montre qu'on peut améliorer les résultats en terme de chemin critique. Mais l'amélioration de la surface de routage, due au non partage d'une façon systématique des sous-fonctions de petite taille, ne permet pas d'améliorer la surface globale, car la perte en surface active n'est pas compensée par le gain en surface de routage.

Il est clair enfin que l'amélioration de la surface globale et plus particulièrement de la surface de routage des circuits dépend d'une manière très étroite de la phase de factorisation. La taille, ainsi que la sortance des sous-fonctions partagées est un facteur important dans l'évaluation de la surface globale d'un circuit.

Chapitre III

Factorisation Lexicographique des Fonctions Booléennes

III.1 INTRODUCTION

Nous avons défini précédemment le cadre dans lequel nous allions développer les méthodes de factorisation. La division restreinte par conoyaux que nous avons proposée dans le chapitre précédent cherche à prendre en considération l'optimisation des réseaux de cellules standards en tenant compte de la connectique et en introduisant des techniques de filtrage. Ces techniques sont appliquées essentiellement sur les sous-fonctions créées lors de la factorisation.

Nous proposons dans ce chapitre une méthode originale, fondée sur la méthode de division par les conoyaux. Cette méthode restreint encore plus l'étape de factorisation en imposant un ordre sur les variables d'entrée pour l'ensemble des fonctions, afin de faciliter la connectique ultérieure. Elle permet de contrôler, d'une part, la dépendance des entrées en créant des cônes logiques structurés [Abo90,Poi90] et, d'autre part, les connections entre cellules, en contrôlant aussi le fanout des cellules logiques utilisées (figure III.1).

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

La phase de décomposition technologique sur cellules standards [Sak90,Cra91] sera réalisée couche par couche en respectant ces cônes logiques.

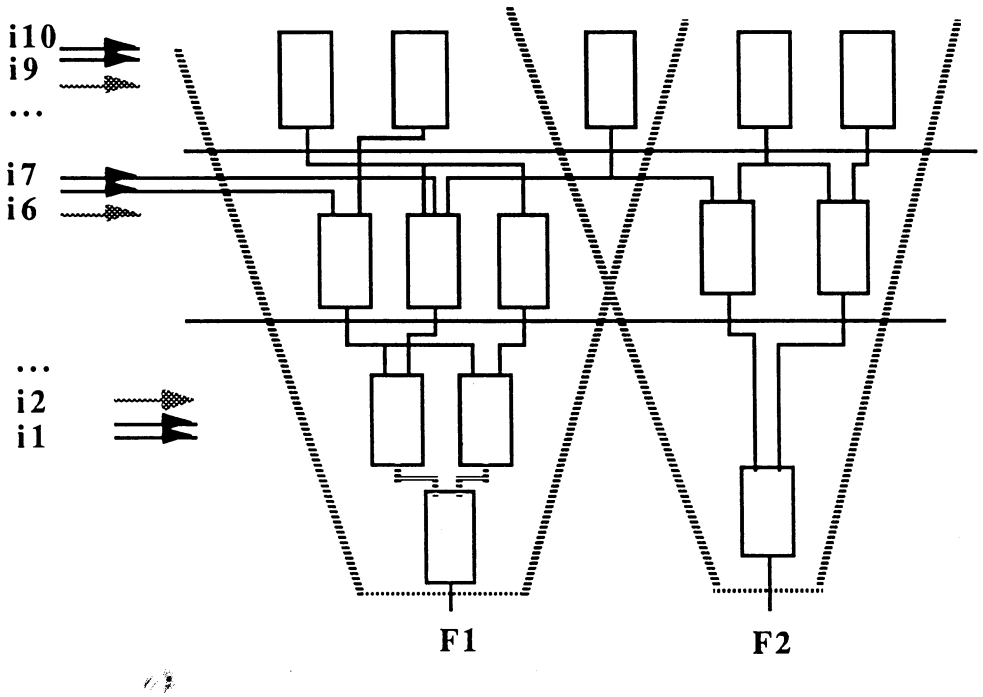


Figure III.1. Cônes logiques virtuels avec l'ordre des entrées i1,i2...i6,i7...i9,i10

III.2 EXPRESSIONS LEXICOGRAPHIQUES DE FONCTIONS BOOLÉENNES

III.2.1 Définitions de base

• ***Définition III.1: Factorisation élémentaire***

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

• ***Définition III.2: Relation de précédence induite par une factorisation élémentaire***

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

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

Exemple:

La factorisation élémentaire $\overline{a}.c.(b.d + \overline{b}.\overline{d})$ implique la relation de

précédence suivante: (a,c) « (b,d).

Remarque:

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

• ***Définition III.3: Compatibilité d'une factorisation élémentaire avec un ordre de référence donné***

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

Exemple:

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

• ***Définition III.4: Factorisations élémentaires lexicographiquement compatibles***

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

Exemple:

Soient $\overline{a}.b.(c.e + d)$ et $\overline{c}.\overline{e}.(\overline{d} + f)$ deux factorisations élémentaires de l'expression $F = \overline{a}.b.c.e + \overline{c}.\overline{d}.\overline{e} + \overline{b}.\overline{d} + \overline{a}.b.d + \overline{c}.\overline{e}.f$.

Elles sont lexicographiquement compatibles car les ordres induits par ces factorisations élémentaires (ab « ced et ce « df) respectent l'ordre de référence {bacedf}.

• ***Théorème III.1:***

Considérons l'expression Booléenne polynomiale d'une fonction F, représentée sous forme de somme de monômes, un ordre de référence et l'ensemble des triplets $(C_i,K_i,\mathfrak{E}_i)$ (cf définition I.18) compatibles avec cet ordre de référence.

Alors:

a) Si C_i est un facteur algébrique de $C_j \Rightarrow \mathfrak{E}_j \subset \mathfrak{E}_i$.

b) Si C_i n'est pas un facteur algébrique de C_j et si C_j n'est pas un facteur algébrique de $C_i \Rightarrow \mathfrak{E}_j \cap \mathfrak{E}_i = \emptyset$.

Preuve:

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

$\forall\ m \in \mathfrak{E}_j,\ m = p.C_j = p.q.C_i$ où p est un monôme de K_j

$\Rightarrow\ m \in \mathfrak{E}_i$

Conclusion: $\mathfrak{E}_j \subset \mathfrak{E}_i$

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

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

Supposons qu'il existe un monôme m tel que $m \in \mathfrak{E}_j \cap \mathfrak{E}_i$

$\Rightarrow$ $\left|\begin{array}{ll} m = p'.C_i & \text{où p' est un monôme de } K_i \\ m = p''.C_j & \text{où p'' est un monôme de } K_j \end{array}\right.$

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

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

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

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

Ceci est impossible, donc $\mathfrak{E}_j \cap \mathfrak{E}_i = \emptyset$.

- ***Corollaire:***

  Deux factorisations élémentaires lexicographiquement compatibles sont algébriquement compatibles.

  Ceci Découle directement de la définition de la compatibilité algébrique (définition I.19) et du théorème III.1.

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

- ***Théorème III.2:***

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

Preuve:

Soit l'ensemble des conoyaux/noyaux (C_i,K_i) d'une expression Booléenne polynomiale minimisée F compatibles avec un ordre de référence donné. Cet ensemble est unique et le corollaire du théorème III.1 nous montre que ces conoyaux/noyaux sont algébriquement compatibles. Par suite, il existe une unique expression factorisée obtenue en divisant algébriquement F par les conoyaux.

III.2.2 Graphe de conoyaux/noyaux

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

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

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

Soient F une fonction Booléenne et M l'ensemble des monômes de F. Si F est une expression polynomiale libre, le graphe de conoyaux/noyaux admet une racine qui est représentée par le nœud N_0 = (1,F,M).

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

Un chemin du graphe est un chemin qui relie le nœud N_0 à une feuille du graphe.

Exemple:

soit la fonction F suivante:

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

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

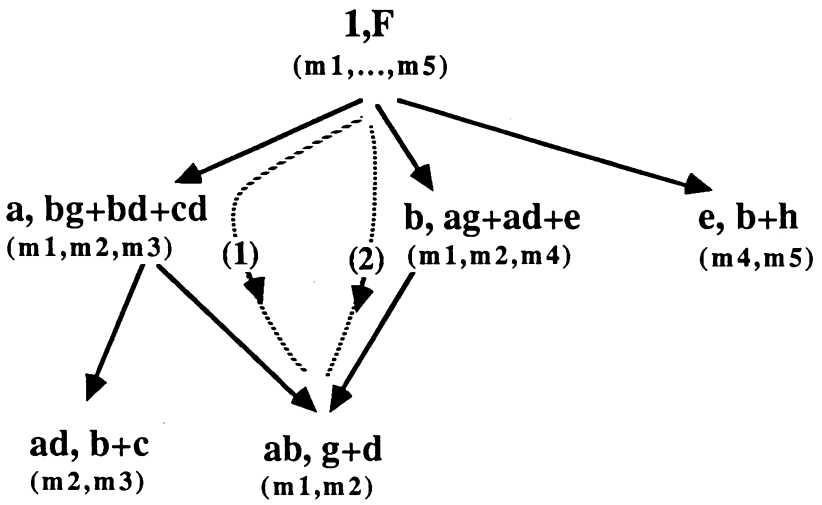


Figure III.2. Graphe des conoyaux/noyaux de la fonction F

Théorème III.3:

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

Preuve:

Soit N_1, N_2, ..., N_n les nœuds d'un chemin du graphe, avec $N_1 = (C_1,K_1,\mathfrak{E}_1)$, $N_2 = (C_2,K_2,\mathfrak{E}_2)$, ..., $N_n = (C_n,K_n,\mathfrak{E}_n)$. Cela veut dire que:

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

Remarque:

Si deux chemins du graphe convergent vers le même nœud alors leurs noyaux sont incompatibles.

Exemple:

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

III.2.3 Arbre de conoyaux/noyaux

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

- Les nœuds N_i sont associés de façon bijective avec un ensemble de factorisations élémentaires algébriquement compatibles caractérisées par les triplets $(C_i, K_i, \mathfrak{K}_i)$,

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

Exemple:

Soit F une fonction Booléenne formée de 7 monômes:

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

L'ensemble des factorisations élémentaires de cette fonction est:

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

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

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

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

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

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

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

La figure III.3 illustre l'ensemble des arbres de conoyaux/noyaux compatibles algébriquement.

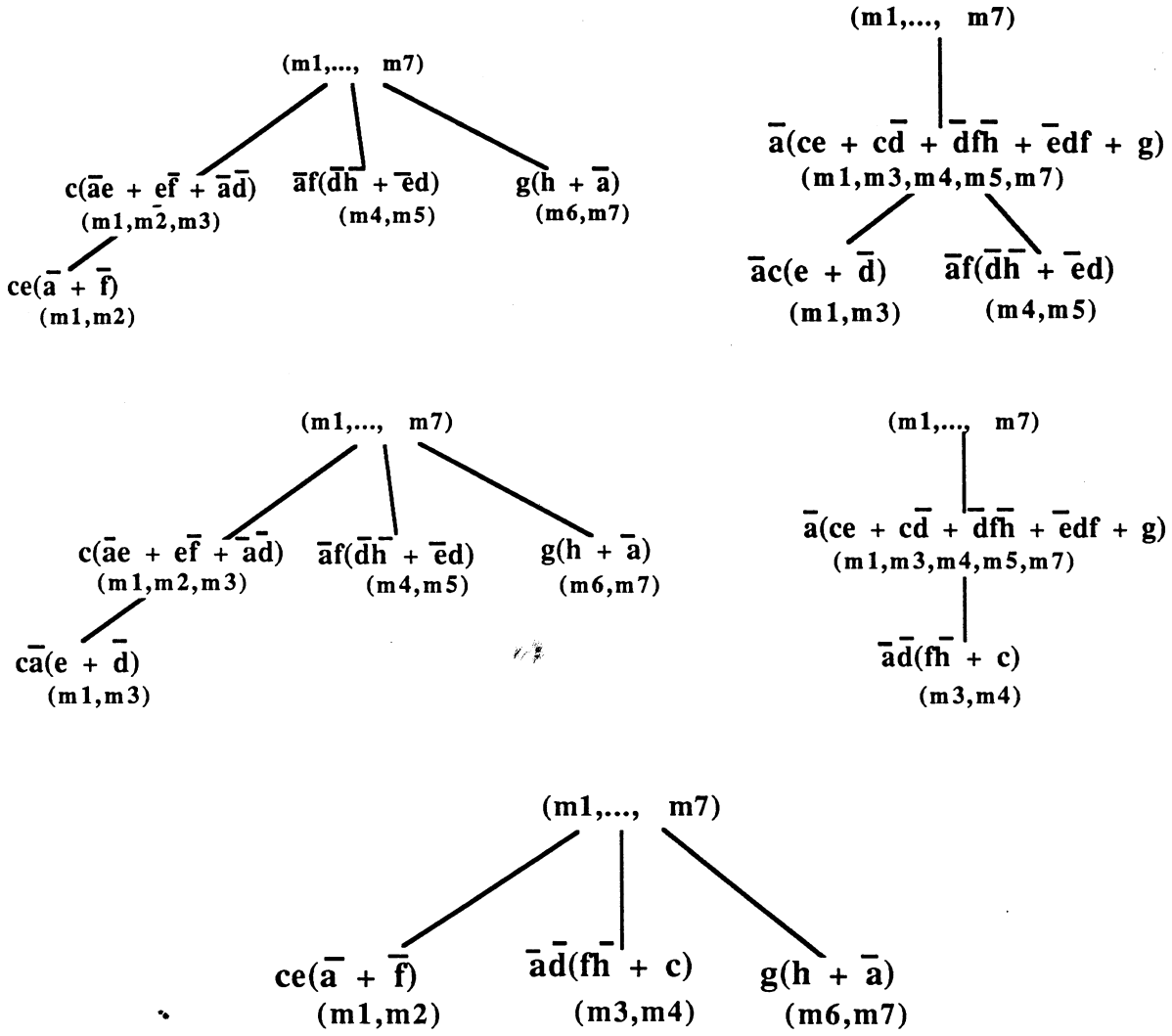


Figure III.3. Arbres de conoyaux/noyaux algébriques.

- ***Propriété III.1:***

  Un nœud $(C_i,K_i,\mathfrak{E}_i)$, d'un chemin dans un arbre de conoyaux/noyaux, est un fils d'un nœud $(C_j,K_j,\mathfrak{E}_j)$ si et seulement si: $V(C_j) \subset V(C_i)$.

- ***Propriété III.2:***

  Pour tout chemin dans un arbre de conoyaux/noyaux pris dans le sens de la racine vers les feuilles, un nœud $(C_i,K_i,\mathfrak{E}_i)$ est un fils d'un nœud $(C_j,K_j,\mathfrak{E}_j)$ si et seulement si: $\mathfrak{E}_j \supset \mathfrak{E}_i$

  La relation $V(C_j) \subset V(C_i)$ de la propriété précédente montre qu'un monôme contenant $V(C_i)$ contient aussi $V(C_j)$. Ceci implique (théorème III.1) que $\mathfrak{E}_j \supset \mathfrak{E}_i$.

- ***Propriété III.3:***

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

  Supposons que cette propriété n'est pas vraie. Alors il existe deux nœuds N_i, N_j appartenant à deux chemins différents d'un arbre et étiquetés par $(C_i,K_i,\mathfrak{E}_i)$, $(C_j,K_j,\mathfrak{E}_j)$, et qui vérifient $\mathfrak{E}_i \cap \mathfrak{E}_j \neq \emptyset$.

  $\Rightarrow \mathfrak{E}_j \supset \mathfrak{E}_i$ ou $\mathfrak{E}_i \supset \mathfrak{E}_j$ (théorème III.1)

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

III.2.4 Arbre de conoyaux/noyaux compatibles lexicographiquement

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

Cette structure de base est fondamentale dans notre algorithme générale de factorisation lexicographique. Elle est utilisée en fait pour effectuer une division lexicographique rapide. La première étape de cette division consiste à rechercher l'ensemble des factorisations élémentaires compatibles avec un ordre de référence et le théorème III.1 nous garantie la compatibilité algébrique. Ce résultat est très important car il permet d'éviter le test, très couteux en temps de calcul, de la compatibilité algébrique de l'ensemble des factorisations élémentaires choisies.

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

Exemple:

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

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

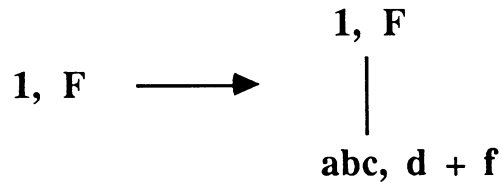
• La paire (a.b.c, d + f) est considérée en premier et l'arbre des conoyaux/noyaux est initialisé (figure III.4a).

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

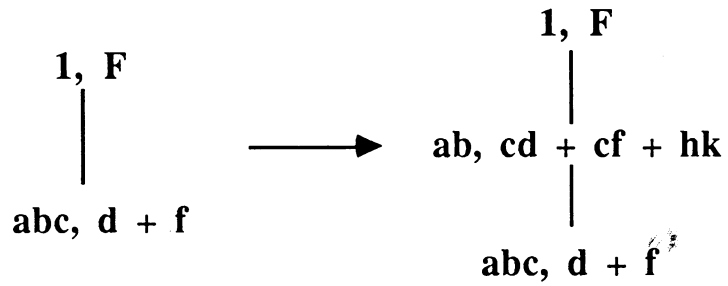
• Pour (h, i.j + i.f + e), étant donné que h et abc ne sont pas liés par une relation d'inclusion un nouvel arc est créé en partant de la racine.

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

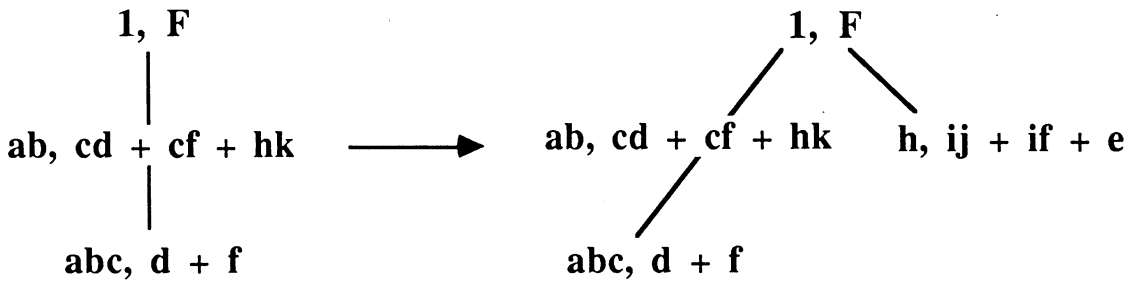
L'arbre final est donné en figure III.4d.



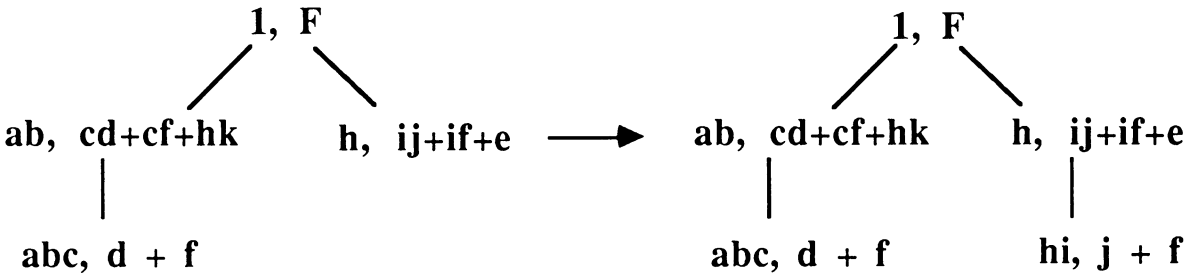
(a)



(b)



(c)



(d)

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

III.3 MATRICE DE PRÉCÉDENCE

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

III.3.1 Définitions et propriétés de la matrice de précédence

- ***Définition III.5: matrice de précédence***

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

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

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

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

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

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

- ***Définition III.6:***

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

- ***Définition III.7: matrice antisymétrique***

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

– $Pr[i,j] = 0 \Rightarrow Pr[j,i] = 1$

– $Pr[i,j] = 1 \Rightarrow Pr[j,i] = 0$

– $Pr[i,j] = (-) \Rightarrow Pr[j,i] = (-)$

- ***Propriété III.4:***

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

- ***Propriété III.5:***

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

III.3.2 Mise à jour de la matrice de précédence

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

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

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

$$\Delta Pr_c = Pr_c \times Pr(R) + Pr(R) \times Pr_c \qquad \text{(III.1)}$$

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

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

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

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

Remarque:

En fait le calcul de ΔPr_c obtenu par la relation (III.1) peut être simplifié car ΔPr_c est une matrice antisymétrique. Il suffit donc d'effectuer le calcul suivant:

$$\Delta Pr_c = (Pr_c \times Pr(R))^* \qquad \text{(III.2)}$$

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

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

• Dans la deuxième étape, il s'agit de calculer la nouvelle matrice de précédence $Pr_c(i+1)$ en considérant toutes les relations de précédence caractérisées par les matrices $Pr_c(i)$, Pr(R) et ΔPr_c. Nous obtenons la matrice $Pr_c(i+1)$ à l'aide de la formule suivante:

$$Pr_c(i+1) = Pr_c(i) + Pr(R) + \Delta Pr_c \qquad \text{(III.3)}$$

Remarque:

La relation (III.3) peut être modifiée si les matrices Pr_c et Pr(R) sont étendues à Pr_c' et Pr'(R), où Pr' est définie par Pr' = Pr + Id, avec Id la matrice identité. Nous obtenons alors la formule suivante:

$$Pr_c'(i+1) = (Pr_c'(i) + Pr'(R))^* \qquad \text{(III.4)}$$

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

Exemple:

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

(R_1) ab « efg

(R_2) e « fh

(R_3) cd « ah

Supposons de plus que les factorisations élémentaires sont choisies dans

l'ordre R_1, R_2, R_3.

• Etape 1: Relation (R_1)

Cette première relation donne la matrice de précédence initiale $Pr_c(1) = Pr(R_1)$ (figure III.5).

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

Figure III.5. Matrice de précédence initiale $Pr_c(1)$.

• Etape 2: Relation (R_2)

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

$$\Delta Pr_c = (Pr_c(1) \times Pr(R_2))^* \quad \text{(figure III.6)}$$

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

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

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

Figure III.6. Matrice de mise à jour ΔPr_c après l'étape 2.

La formule (III.3) nous donne la matrice de précédence $Pr_c(2)$ (figure III.7).

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

Figure III.7. Matrice de précédence $Pr_C(2)$

• Etape 3: Relation (R_3)

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

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

Figure III.8. Matrice finale

III.3.3 Extraction de l'ordre des variables à partir de la matrice de précédence

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

• On sélectionne dans la matrice de précédence finale les lignes ne contenant pas de zéros; Les entrées correspondantes constituent le premier sous-ensemble des variables d'entrées.

• On élimine les lignes et les colonnes correspondant à ces variables.

• On itère le processus pour extraire les sous-ensembles suivants, et jusqu'à ce qu'on détermine le sous-ensemble de variables ne contenant que des zéros ou des (-). Les variables appartenant à ce dernier sous-ensemble apparaissent à la fin de l'ordre lexicographique.

De la matrice de la figure III.8, on extrait une relation d'ordre partielle entre les variables. Cela donne:

bcd « a « eg « fh

III.3.4 Division Algébrique et lexicographique respectant un ordre initial des variables d'entrée

Dans certains cas, un ordre sur les variables d'entrée peut être imposé. C'est le cas, par exemple, lorsque les signaux d'entrée sont asynchrones. Les variables d'entrée ayant des retards plus importants que celui d'autres variables du réseau doivent être considérées comme prioritaires dans le processus de synthèse. Donc, ces variables sont placées le plus près possible des sorties du réseau (figure III.9). Par conséquent le délai à travers la logique entre ces entrées et les sorties est minimal.

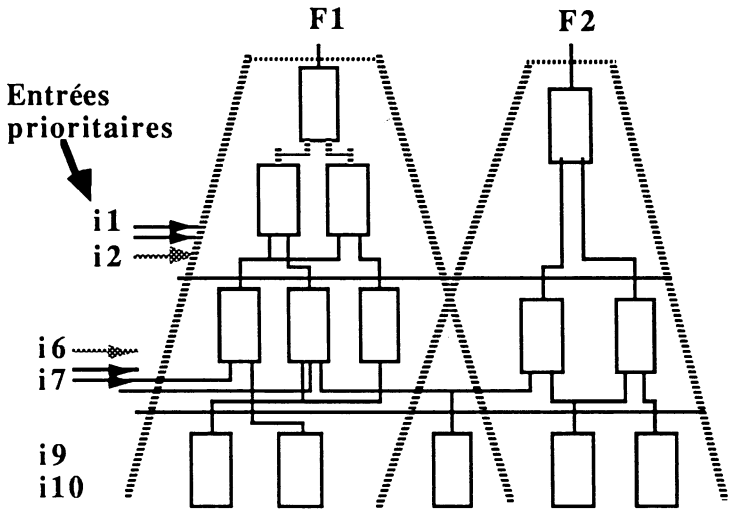


Figure III.9. Illustration de la structure des fonctions Booléennes avec une priorité sur certaines variables d'entrée.

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

Procédure de division

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

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

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

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

Division_Alg_Lex $([M_{cour}, m_{cour}], L)$

début

si $(L = NULL)$ *retourner;* /* *cela signifie qu'il n'y a plus de littéraux dans la liste ordonnée* */

if $(|M_{cour}| = 1)$ *retourner;* /* *Cardinal de* M_{cour} *doit être >1*/

$M_{inter} = M_{cour} \cap M_L^{init}$;

$M_{res} = M_{cour} - M_{inter}$;

Pour chaque monôme m_i *tel que* $m_i \in M_{inter}$ *faire*

/* *Test de la compatibilité lexicographique*/

début

si (*la relation* $(V(m_{cour}.L) \ll V(\frac{m_i}{(m_{cour}.L)}))$ *n'est pas compatible avec l'ordre de référence*) *alors*

début

$M_{inter} = M_{inter} - m_i$; /* *mise à jour de* M_{inter} *et* M_{res} *suivant la*

```
            M_res = M_res + m_i;        compatibilité lexicographique*/
        fin
    fin
    si ( |M_inter| > 1) alors
    début
        si ( |M_res| = 0) alors
        début
            Division_Alg_Lex ([M_cour,m_cour.L], LittéralSuivant(L));
            retourner;
        fin
        BlocGauche = [M_inter,m_cour.L];
        Division_Alg_Lex (BlocGauche, LittéralSuivant(L));
        BlocDroit = [M_res,m_cour.L];
        Division_Alg_Lex (BlocDroit, LittéralSuivant(L));
        FaireArbreDivision ([M_cour,m_cour], BlocGauche , BlocDroit);
        retourner;
    fin
    Division_Alg_Lex ([M_cour,m_cour], LittéralSuivant(L));
    retourner;
fin;
```

L'appel initial de la procédure est:

Division_Alg_Lex ([M_F,1], PremierLittéral)

où:

F est la fonction initiale minimisée.

M_F est l'ensemble des monômes de F.

1 est le cofacteur de F.

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

Exemple:

Soit une fonction Booléenne F formée de 6 monômes:

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

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

$M_a^{init} = \{m_1, m_2, m_3, m_4\}$

$M_b^{init} = \{m_1, m_4, m_5, m_6\}$

$M_c^{init} = \{m_2, m_3, m_4, m_6\}$

La procédure de division respectant l'ordre {a,b,c,...} est illustrée ci-dessous, et conduit à l'arbre donné dans la figure III.10:

- littéral a

  $M_a = M_F \cap M_a^{init} = \{m_1, m_2, m_3, m_4\}$ $\qquad M_{res}^a = M_F - M_a = \{m_5, m_6\}$

- littéral b

  $M_{ab} = M_a \cap M_b^{init} = \{m_1, m_4\}$ $\qquad M_{res}^{ab} = M_a - M_{ab} = \{m_2, m_3\}$

  $M_b = M_{res}^a \cap M_b^{init} = \{m_5, m_6\}$ $\qquad M_{res}^b = \emptyset$

- littéral c

  $M_{abc} = M_{ab} \cap M_c^{init} = \{m_4\}$

  $M_{ac} = M_{res}^{ab} \cap M_c^{init} = \{m_2, m_3\}$ $\qquad M_{res}^{ac} = \emptyset$

  $M_{bc} = M_b \cap M_c^{init} = \{m_6\}$

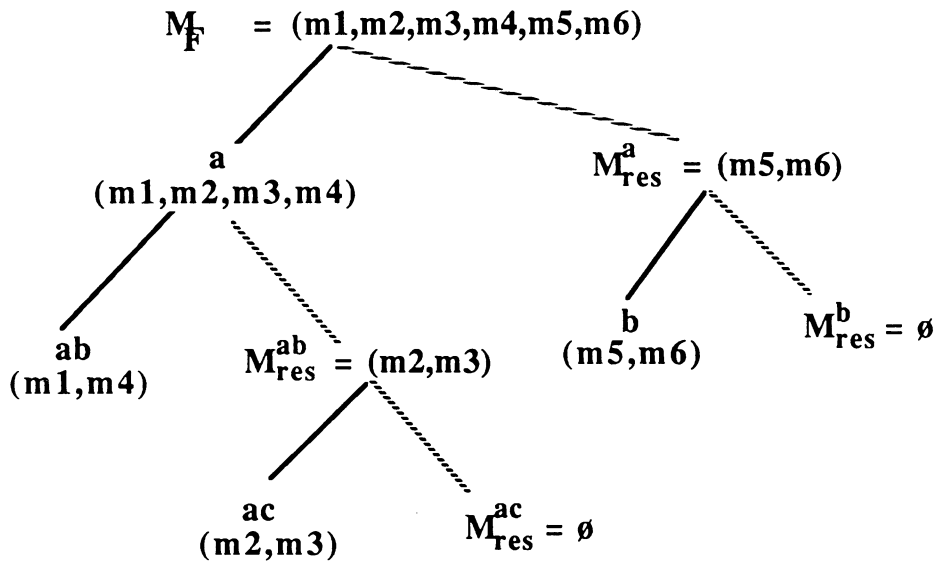


Figure III.10. Division algébrique et lexicographique respectant l'ordre de référence {abc...}.

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

III.4 FACTORISATION LEXICOGRAPHIQUE D'UN ENSEMBLE DE FONCTIONS BOOLÉENNES EN VUE D'UNE IMPLANTATION SUR CELLULES STANDARDS.

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

Il est évident qu'une recherche exhaustive du meilleur ordre possible n'est pas envisageable. Un tel algorithme ne pourrait pas donner une solution dans un temps raisonnable (solution en N!, N étant le nombre de variable). Nous proposons donc [Abo90,Poi90], une heuristique, utilisant la notion de gain relatif à une factorisation élémentaire (cf. § I.4).

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

• Sélection d'un ensemble de factorisations élémentaires compatibles lexicographiquement et construction de l'ordre de référence des variables

d'entrée.

• Construction de l'arbre de conoyaux/noyaux et application de la première étape de division rapide.

• Division des restes.

• Recherche de sous-expressions communes.

III.4.1 Génération de l'ordre des variables et de l'ensemble des factorisations élémentaires compatibles.

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

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

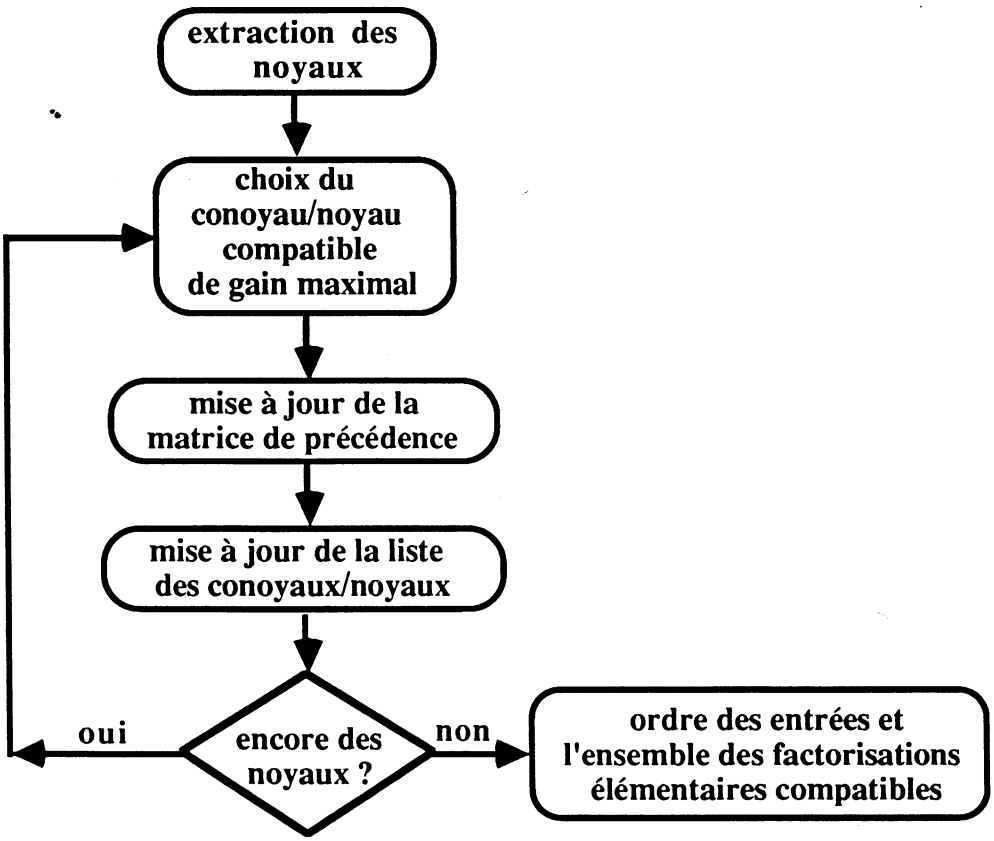


Figure III.11. Algorithme de sélection des conoyaux/noyaux lexicographiquement compatibles.

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

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

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

III.4.2 Construction de l'arbre de conoyaux/noyaux et première étape de division

L'étape précédente de sélection des factorisations élémentaires donne un ensemble de factorisations élémentaires (C_i, K_i) compatibles lexicographiquement. Le théorème III.1 montre que tous les couples (C_i, K_i) sont compatibles algébriquement. Cette approche novatrice très importante nous permet, pour chaque fonction, de construire l'arbre de conoyaux/noyaux sans se préoccuper de la compatibilité algébrique nécessaire dans les approches classiques de factorisation. On a vu également que l'arbre de conoyaux/noyaux est unique pour un ensemble de factorisations élémentaires. Il constitue la structure de base pour l'algorithme de division.

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

Exemple:

soit la fonction

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

Soit {cagfedh} un ordre de référence donné, l'arbre lexicographique de conoyaux/noyaux est donné dans la figure III.12.

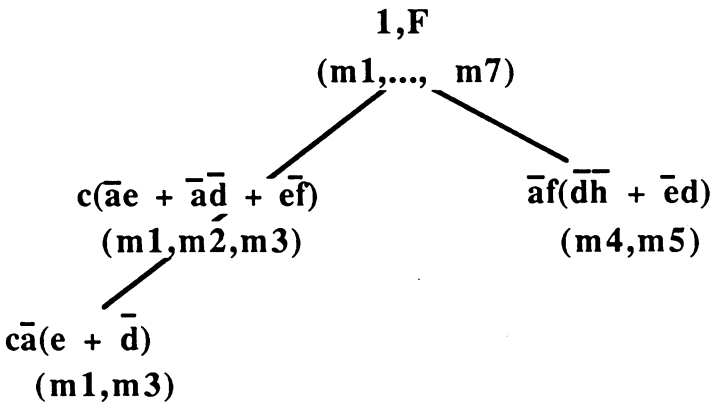


Figure III.12. Arbre de conoyaux/noyaux lexicographique.

A l'étape 1, la fonction F s'écrit:

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

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

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

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

Cette expression finale est représentée sous la forme d'un arbre lexicographique dont les nœuds sont maintenant des opérateurs Booléens classiques OU et ET (figure III.13).

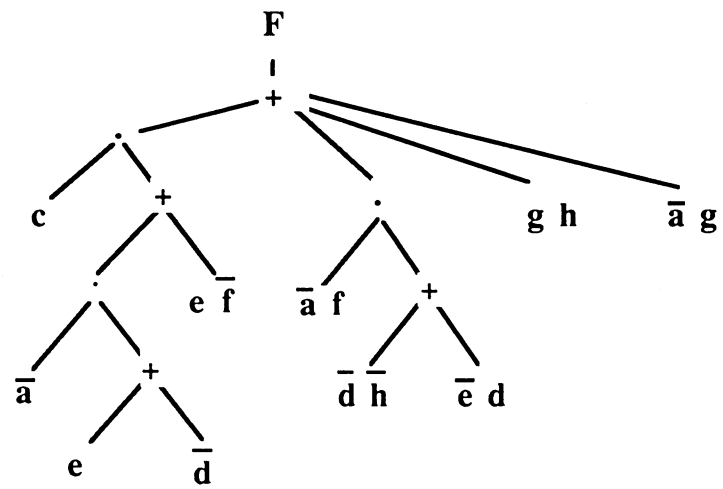


Figure III.13. Arbre de factorisation lexicographique.

III.4.3 Division des restes

A l'issue de l'étape de construction de l'arbre de conoyaux/noyaux, certain

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

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

Cette étape est composée de deux phases:

•1ère phase: Extraction de noyaux

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

Notons que les noyaux extraits pendant cette étape ne font pas partie de l'ensemble initial des noyaux de la fonction. Pour illustrer ce point, on considère la fonction F = a.b + a.d + a.f + a.i + i.j.h + i.e. Il existe deux factorisations élémentaires possibles {(a, b + d + f + i), (i, a + g.h + e)}.

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

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

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

partielles compatibles avec l'ordre des variables.

Exemple:

Soit la fonction

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

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

III.4.4 Recherche de sous-expressions communes

Cette dernière étape concerne la reconnaissance de sous-expressions communes (autres que les noyaux globaux, qui eux, sont repérés et mis en commun pendant la phase de construction de l'arbre de conoyaux/noyaux). Elle peut être considérée comme optionnelle car elle perturbe la régularité du routage obtenue à partir des trois étapes précédentes. Néanmoins, les expériences faites dans ce domaine montrent que c'est une étape primordiale dans le cas où une optimisation en surface est requise. Certains paramètres peuvent être pris en compte pour améliorer la surface du routage dans le circuit final. C'est le cas, par exemple, du filtre appliqué sur certaines sous-fonctions, en fonction de leur taille (nombre de littéraux admis dans l'expression de la sous-fonction), qui permet d'éviter la création d'un très grand ensemble de sous-fonctions de faibles tailles dans le réseau final. Nous avons vu dans le chapitre précédent que la taille de ces sous-fonctions doit être limitée à 3 littéraux pour les circuits de moyenne et haute complexité. Les résultats présentés dans ce chapitre tiennent compte de cette dernière étape ainsi que du critère de filtrage précédent.

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

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

Pour chaque nœud N_i de l'arbre factorisé dont l'opérateur Booléen est un opérateur OU, nous partitionnons les successeurs du nœud N_i en deux blocs,

le bloc A contient les feuilles étiquetées par un seul littéral (l_1^i, ..., l_k^i), le bloc B contient les feuilles étiquetées par un monôme (m_1^i, ..., m_n^i).

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

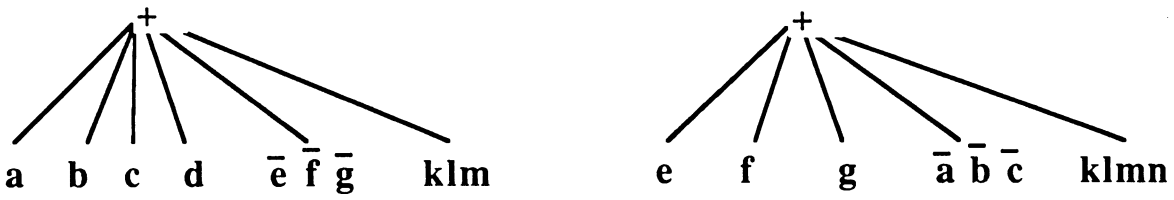
• Phase 2: Extraction des monômes et sous-monômes communs.

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

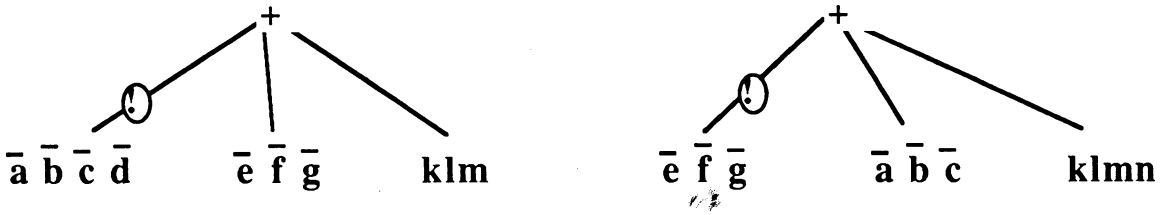
A la fin de cette étape, les inverseurs créés pendant la phase 1 seront propagés jusqu'aux feuilles et les nouvelles sous-fonctions sont rajoutées à l'ensemble des fonctions du réseau.

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

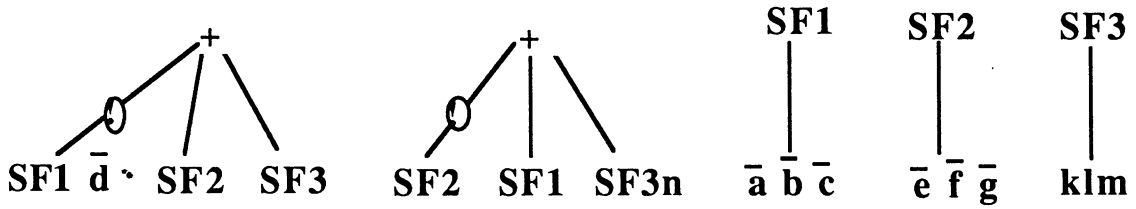
Pour illustrer cet algorithme, considérons les deux nœuds OU de la figure III.14a. La figure III.14b montre la transformation des feuilles étiquetées par un seul littéral en monômes. La figure III.14c est le résultat de l'algorithme de recherche de sous-expressions communes, et la figure III.14d donne le résultat final après propagation des inverseurs vers les feuilles.



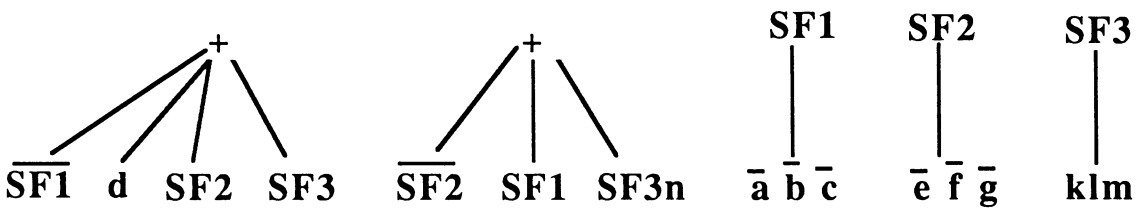
(a)



(b)



(c)



(d)

Figure III.14. Illustration de la recherche de sous-expressions communes.

III.5 RÉSULTATS EXPÉRIMENTAUX

III.5.1 Contrôle de la connectique

Le premier résultat attendu dans le cas de la factorisation lexicographique, est le bon contrôle de la connectique. La structure des réseaux Booléens sous forme de cônes logiques structurés dans lesquels les variables entrent suivant le même ordre, permet aux outils de décomposition technologique de tenir compte de cette structure et de la garder au niveau des réseaux de cellules standards. Un outil classique de placement routage doit en conséquence tirer un avantage de cette structure pour favoriser des connexions locales entre les cellules standards. Cette simplification du routage est due aussi au fait qu'une entrée est utilisée uniquement dans une seule couche logique du réseau.

Pour illustrer cet aspect de la factorisation lexicographique, une première expérimentation est faite sur des circuits tests de moyenne et de haute complexité. Nous avons fait la synthèse de ces circuits en utilisant la bibliothèque de cellules standards VSC320 en technologie CMOS 1,2µ de VLSI Technology Inc. Ces circuits ont été synthétisés par trois outils différents: le premier est l'outil MIS2.2 de Berkeley [Bra87], et le script utilisé est le script standard proposé par l'outil. Le deuxième est l'outil de CAO de la société "COMPASS Design Automation", et enfin le troisième est le système ASYL dans lequel est implanté la factorisation lexicographique. L'outil de placement routage utilisé est celui de la société COMPASS. La figure III.15 montre les différentes valeurs du facteur de routage obtenues par les trois outils pour quelques circuits tests de moyenne et de haute complexité. Dans ce cas, le gain moyen de la factorisation lexicographique comparée à la solution obtenue par MIS2.2 et à celle de COMPASS est respectivement de l'ordre de 15% et 20%.

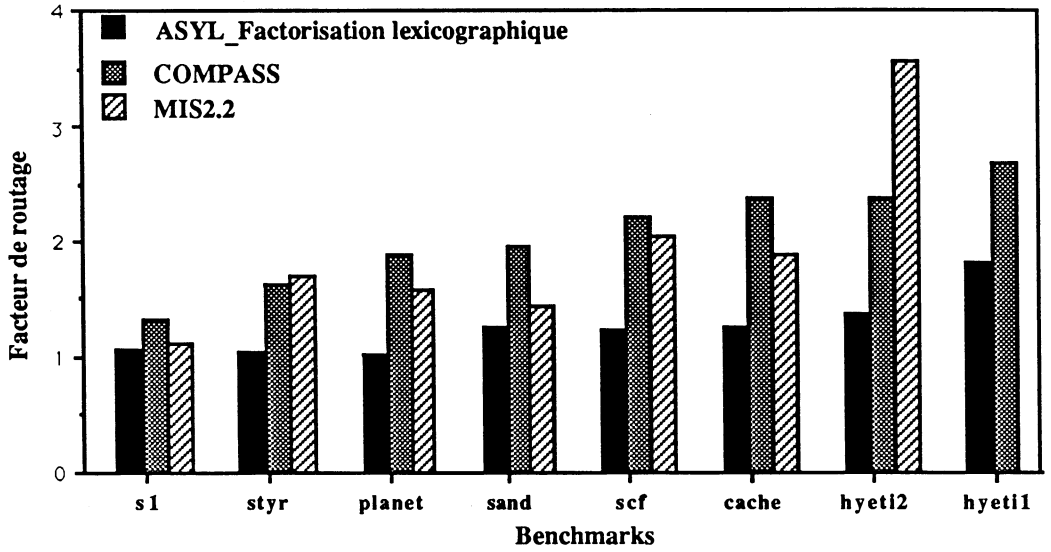


Figure III.15. Comparaison entre les différents facteurs de routage

III.5.2 Résultats et évaluations

Le graphique précédent montre une amélioration importante du facteur de routage dans les circuits d'une manière générale. Ce gain influence d'une manière très importante la surface globale des circuits surtout pour les circuits de haute complexité qui ont généralement un facteur de routage important.

Une première série d'experimentations consiste à comparer la méthode de factorisation lexicographique à celle de la division restreinte développée dans le chapitre précédent. Les expériences ont été faites sur les circuits de moyenne et haute complexité. L'environnement expérimental est celui du système ASYL pour les phases de factorisation et décomposition technologique et les outils de COMPASS pour les phases de placement routage et extraction de chemin critique. Dans ce cas la bibliothèque de cellules standards utilisée est la bibliothèque VSC370 1,2µ de VLSI Technology Inc. Les résultats de surface et de chemin critique sont donnés respectivement en (mils sq) et en (ns).

III.5.2.1 Circuits de moyenne complexité

Le tableau III.1 montre les résultats obtenus par les deux méthodes. Il résume respectivement les résultats obtenus dans le cas d'une optimisation en surface ou en vitesse. D'une manière générale, les résultats de la division

restreinte sont meilleurs que ceux de la factorisation lexicographique. Le gain de surface, dans le cas d'une optimisation en surface ou en vitesse, est de l'ordre de 3%. Le gain en chemin critique est de l'ordre de 5% dans les deux cas. Ce gain est évalué selon la formule:

$$\text{Gain } \% = 100 * \frac{\text{résultat de la factorisation lexico - résultat de la division restreinte}}{\text{résultat de la factorisation lexico}}$$

Bien que le facteur de routage obtenu par la factorisation lexicographique soit en moyenne inférieur de 5% à celui obtenu par la division restreinte, l'approche lexicographique ne donne pas de résultats satisfaisants. Le gain obtenu en surface de routage ne compense pas la perte en surface active due à la restriction supplémentaire imposée sur l'ordre des variables pendant la phase de la factorisation.

	Méthode de division restreinte			Méthode de factorisation lexicographique			Gain en %		
Optimisation orientée surface									
Name	**S_T**	**F_r**	**CC**	**S_T**	**F_r**	**CC**	**$G(S_T)$**	**$G(F_r)$**	**G(CC)**
jay	695	1,13	16,3	713	1,05	17,4	2,52	-7,62	6,32
s1	719	1,21	14,2	730	1,11	15,4	1,51	-9,01	7,79
tbk	780	1,12	19	801	1,1	18,9	2,62	-1,82	-0,53
platcos	793	1,24	12,2	834	1,16	13,6	4,92	-6,90	10,29
styr	956	1,16	20	988	1,08	21,8	3,24	-7,41	8,26
planet	1071	1,13	18,8	1052	1,06	19,6	-1,81	-6,60	4,08
planet1	1094	1,09	21,5	1117	1,06	20,7	2,06	-2,83	-3,86
sand	1224	1,19	19,3	1298	1,21	22,9	5,70	1,65	15,72
cache_c	1851	1,29	26,7	1938	1,22	28,2	4,49	-5,74	5,32
scf	1892	1,23	28,1	1942	1,23	28,2	2,57	0,00	0,35
Gain moyen							**2,78**	**-4,63**	**5,37**
Optimisation orientée vitesse									
Name	**S_T**	**F_r**	**CC**	**S_T**	**F_r**	**CC**	**$G(S_T)$**	**$G(F_r)$**	**G(CC)**
jay	698	1,03	10,7	732	1,08	11,5	4,64	4,63	6,96
s1	741	1,17	12,6	753	1,07	10,9	1,59	-9,35	-15,60
platcos	790	1,19	8,2	852	1,18	9,4	7,28	-0,85	12,77
tbk	798	1,1	15,8	843	1,07	15,3	5,34	-2,80	-3,27
styr	984	1,15	13,2	1003	1,05	15,3	1,89	-9,52	13,73
planet	1106	1,15	11,4	1063	1,02	12,9	-4,05	-12,75	11,63
planet1	1134	1,1	12	1141	1,08	13,1	0,61	-1,85	8,40
sand	1284	1,24	16,1	1352	1,24	14,7	5,03	0,00	-9,52
scf	1986	1,28	14,9	2005	1,22	16,9	0,95	-4,92	11,83
cache_c	1997	1,36	15,4	2137	1,25	18,3	6,55	-8,80	15,85
Gain moyen							**2,98**	**-4,62**	**5,28**

S_T : surface totale du circuit (en mils sq)

F_r : facteur de routage

CC : chemin critique (en ns)

Gain en % : gain de la méthode de division restreinte par rapport à la méthode de factorisation lexicographique.

Tableau III.1. Résultats de comparaison entre la division restreinte et la factorisation lexicographique pour les circuits de moyenne complexité.

III.5.2.2 Circuits de haute complexité

Dans ce cas, les résultats obtenus par la factorisation lexicographique sont légérement meilleurs que ceux de la division restreinte sauf pour le chemin critique qui perd en moyenne 9% dans le cas d'une optimisation en vitesse (tableau III.2). Cela montre l'intérêt de la factorisation lexicographique d'autant plus que le temps de calcul, dans le cas des circuits de haute complexité, est 4 à 5 fois inférieur à celui de la division restreinte.

	Méthode de division restreinte			Méthode de factorisation lexicographique			Gain en %		
Optimisation orientée surface									
Name	**S_T**	**F_r**	**CC**	**S_T**	**F_r**	**CC**	**G(S_T)**	**G(F_r)**	**G(CC)**
zeegers	3766	1,56	37,3	3725	1,4	36,5	-1,10	-11,43	-2,19
hyeti2	5966	1,61	48,5	5979	1,43	46,7	0,22	-12,59	-3,85
hyeti1	10703	2,01	71,6	10055	1,86	73,2	-6,44	-8,06	2,19
Gain moyen							**-2,44**	**-10,69**	**-1,29**
Optimisation orientée vitesse									
Name	**S_T**	**F_r**	**CC**	**S_T**	**F_r**	**CC**	**G(S_T)**	**G(F_r)**	**G(CC)**
zeegers	3641	1,42	18,2	3780	1,4	21,8	3,68	-1,43	16,51
hyeti2	6389	1,84	24,8	6253	1,38	26,1	-2,17	-33,33	4,98
hyeti1	11487	2,2	28,5	10792	1,82	31,5	-6,44	-20,88	9,52
Gain moyen							**-1,65**	**-18,55**	**9**

S_T : surface totale du circuit (en mils sq)

F_r : facteur de routage

CC : chemin critique (en ns)

G(%) : gain de la méthode de division restreinte par rapport à la méthode de factorisation lexicographique.

Tableau III.2. Résultats de comparaison entre la division restreinte et la factorisation lexicographique pour les circuits de haute complexité.

Le tableau III.3 montre quelques circuits tests de très haute complexité qui n'ont pas pu être synthétisés par la méthode de division restreinte sans l'utilisation de méthodes de partitionnement pour réduire la complexité du circuit. Par contre, la factorisation lexicographique permet de synthétiser ces circuits en moins de 4h pour le circuit le plus complexe.

Cette différence importante en temps de calcul qui existe entre les deux méthodes est due au fait que, dans la factorisation lexicographique la phase de recherche de noyau n'est faite qu'une seule fois. De plus, la compatibilité algébrique entre les différents candidats est assurée grâce à la compatibilité lexicographique.

Benchmarks	nb de littéraux initial	nb de littéraux final	temps CPU (min)
imec5	10995	4206	25
apex2	14728	1730	7
imec9	15321	5771	53
seq	17262	3472	26
imec7	20423	7640	75
imec4	26723	9506	144
imec3	32053	10876	173
imec2	41074	14762	216

Tableau III.3. circuits tests de très haute complexité

III.5.2.3 Comparaison avec d'autres outils de synthèse

Dans ce paragraphe, nous allons comparer les résultats de la factorisation lexicographique avec ceux obtenus par les deux outils cités précédemment (MIS2.2 et COMPASS).

Les circuits tests utilisés sont les circuits de moyenne et de haute complexité. Les figures III.16 et III.17 représentent les résultats obtenus pour ces différents circuits par les trois outils ASYL, MIS2.2 et COMPASS. La bibliothèque de cellules standards est la bibliothèque VSC320. La surface globale est mesurée en mils square (1 mils sq = 645,1 μ^2) et le chemin critique, obtenu après extraction et estimation des capacités de connexion après la phase de placement routage, est donné en nanoseconde (ns). Les

différents résultats correspondent à:

• ASYL_LEX_S: dans ce cas, la factorisation lexicographique est utilisée. Elle est suivie d'une décomposition technologique orientée surface (_S).

• ASYL_LEX_D: correspond à une factorisation lexicographique suivie d'une décomposition orientée délai ou chemin critique (_D).

• MIS_22_S: c'est le résultat obtenu par MIS2.2 (script standard) suivi de la décomposition orientée surface du système ASYL.

• MIS_22_D: c'est le résultat de MIS2.2 suivi de la décomposition orientée délai du système ASYL.

• IND: correspond aux résultats obtenus par l'outil COMPASS.

Le temps de calcul pour chaque méthode est indiqué, si possible, entre parenthèse. Notons que, ce temps pour la factorisation lexicographique est très faible comparé à celui de MIS2.2 qui ne donne aucun résultat pour les 4 derniers circuits du tableau III.4.

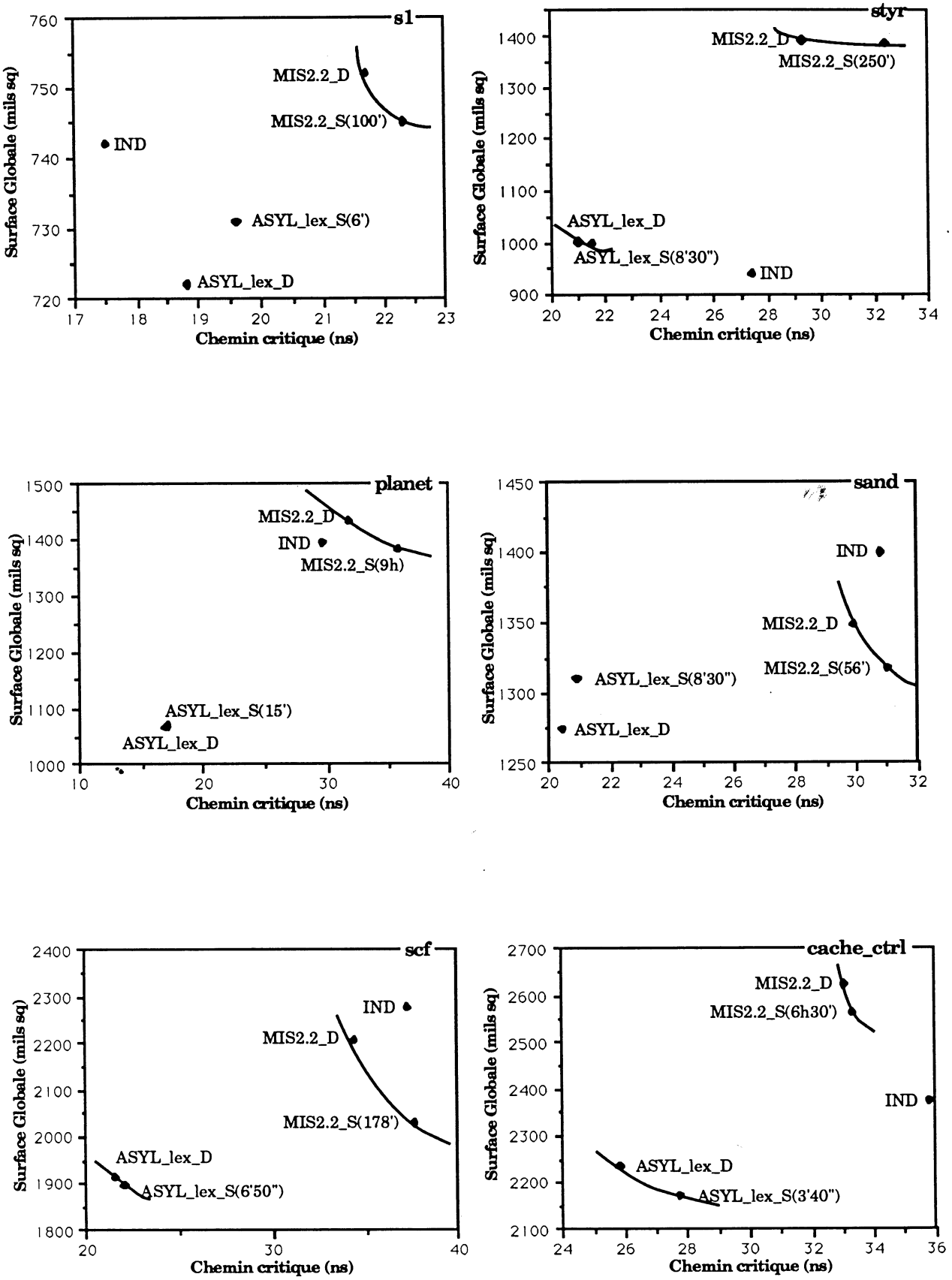


Figure III.16. Exemples de complexité moyenne.

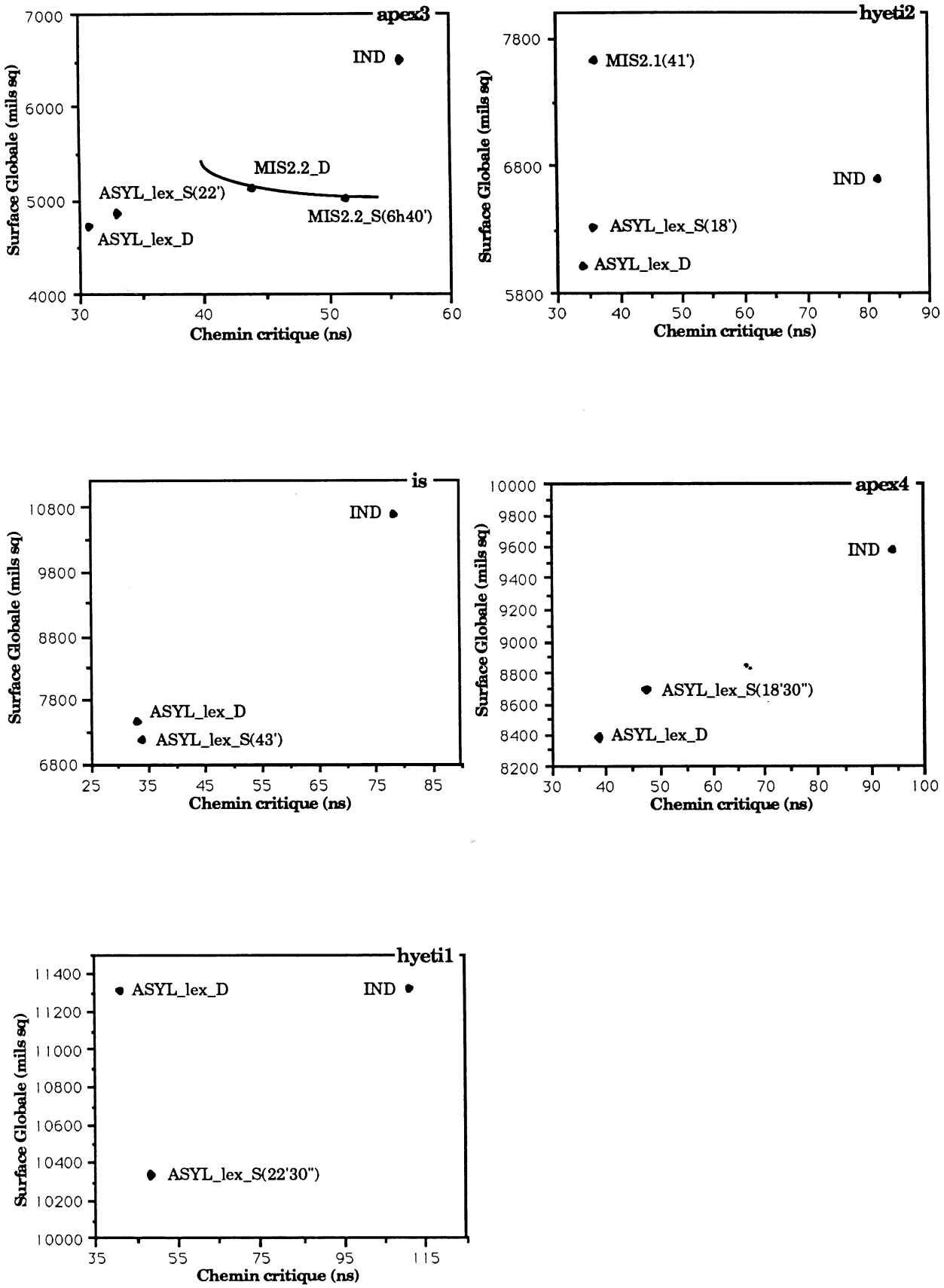


Figure III.17. Exemples de haute complexité.

	ASYL_lex_S			ASYL_lex_D			IND (COMPASS)		
benchmarks	S_T (mils sq)	F_r	C C (ns)	S_T (mils sq)	F_r	C C (ns)	S_T (mils sq)	F_r	C C (ns)
s1	731	1.11	19.6	722	1.32	18.8	742	1.33	17.5
styr	998	1.08	21.5	1001	1.11	21.0	943	1.63	27.4
planet	1069	1.06	17.1	1066	1.26	16.9	1394	1.87	29.6
sand	1311	1.69	20.9	1274	1.5	20.4	1400	1.96	30.8
scf	1897	1.24	22.0	1916	1.26	21.5	2277	2.22	37.3
cache_ctrl	2172	1.27	27.7	2237	1.34	25.8	2375	2.39	35.8
apex3	4880	1.75	32.9	4741	1.66	30.6	6509	3.84	55.9
hyeti2	6326	1.79	35.7	6018	1.65	33.9	6689	2.39	81.4
is	7188	2.01	34.0	7462	2.13	33.0	10687	3.63	78.2
apex4	8696	2.25	47.6	8385	2.11	38.8	9581	2.78	94.2
hyeti1	10344	2.36	29.8	11313	2.39	41	11325	2.68	48.7

	MIS 2.2_S			MIS2.2_D		
benchmarks	S_T (mils sq)	F_r	C C (ns)	S_T (mils sq)	F_r	C C (ns)
s1	745	1.12	22.3	752	1.18	21.7
styr	1385	1.69	32.4	1392	1.70	29.3
planet	1384	1.58	35.7	1434	1.66	31.8
sand	1317	1.44	31.0	1349	1.54	29.9
scf	2031	2.05	37.6	2209	2.27	34.3
cache_ctrl	2564	1.88	33.3	2622	1.94	33.0
apex3	5037	3.24	51.4	5138	3.09	43.8
hyeti2	Arrêt après 30h CPU 7628* 3.57* 36*			-	-	-
is	Arrêt après 30h CPU			-	-	-
apex4	Arrêt après 30h CPU			-	-	-
hyeti1	Arrêt après 30h CPU			-	-	-

*: Résultat de MIS2.1

Tableau III.4. Résultats comparatifs sur les circuits tests de moyenne et haute complexité.

Remarque:

Pour certains circuits tests, une amélioration de la surface globale est obtenue après la phase de décomposition technologique orientée délai. Si on examine la surface active uniquement, on voit qu'elle est supérieure à celle obtenue dans le cas d'une optimisation en surface. Par contre la surface de routage est inférieure. Cette diminution de la surface de routage dans le cas d'une optimisation orientée délai est due à l'existence de portes logiques plus grandes dans le réseau de cellules standards qui ont plus de transparence par rapport aux portes classiques qui elles, sont optimisées en surface. Cette transparence donne à l'outil de placement routage plus de liberté, et lui permet de mieux tenir compte de la structure du réseau de cellules obtenu par la factorisation lexicographique.

III.6 CONCLUSION

Dans ce chapitre nous avons développé une méthode originale de factorisation, fondée sur un ordonnancement des variables d'entrée dans les réseaux Booléens. Cet ordre permettant de structurer les réseaux uniformément en fonction des entrées, est obtenu à partir des propriétés des noyaux algébriques. Il a permis d'obtenir une réduction de la surface de routage dans les circuits de haute complexité et par conséquent une réduction de la surface globale de ces circuits. Les propriétés algébriques obtenues grâce à l'ordonnancement des variables, à savoir la compatibilité algébrique qui découle automatiquement de la compatibilité lexicographique, ont permis également de traiter des circuits qui n'ont pas pu être synthétisés par d'autres outils, et cela avec un temps de calcul extrêmement court si on tient compte de la complexité de ces circuits.

Chapitre IV

Factorisations Booléennes et Optimisation de Réseaux Booléens

IV.1 INTRODUCTION

Comme on l'a annoncé dans l'introduction, des techniques plus élaborées appelées techniques Booléennes sont apparues donc pendant la dernière décennie. Elles ont pour but de réduire la complexité des expressions factorisées, exprimée en terme de littéraux. Elles ont été utilisées et développées à l'université de Berkeley [Bra87,Sal87,Saj90,Saj91], et sont fondées sur le principe de la substitution Booléenne qui utilise les valeurs non définies des sous-fonctions d'un réseau Booléen. Ces conditions permettent de trouver des candidats diviseurs Booléens tels que les noyaux Booléens.

Ce chapitre reprend un certain nombre de définitions concernant ces techniques Booléennes et a comme objectif de délimiter le champ d'application pratique. Pour ceci, des expérimentations seront conduites dans l'environnement du système de synthèse SIS de Berkeley; en effet les techniques de factorisation Booléenne n'ont pas été implantées dans le système ASYL précisément à la lueur des expériences reportées ici. L'objectif des expérimentations de ce chapitre est précisément d'examiner si, après placement routage, ces techniques amènent réellement un bénéfice en terme de surface et de chemin critique par rapport aux mèthodes algébriques classiques.

IV.2 NOYAUX BOOLÉENS

La génération de l'ensemble des noyaux Booléens est un problème délicat et relativement complexe. Elle met en oeuvre 3 étapes différentes: la première est l'étape d'extraction des noyaux algébriques, la deuxième étape concerne la génération des noyaux Booléens et consiste à étendre les noyaux algébriques par inclusion des littéraux des conoyaux associés, et enfin la troisième étape détermine les intersections des noyaux étendus [Bra87,Lis87].

Nous allons illustrer ces trois étapes sur l'exemple suivant:

Exemple:

Soient deux fonctions Booléennes F_1 et F_2 représentées par les deux expressions suivantes:

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

$F_2 = a.b.f.e.g + b.c.f$

• 1ère étape: détermination de l'ensemble des noyaux algébriques
$\{(c.d, a.e.g + b)_{F_1}, (b.f, a.e.g + c)_{F_2}\}$

• 2ième étape: extension des noyaux algébriques; elle se fait en incluant successivement toutes les parties du conoyau:

$$K_{ext1} = a.e.g + b + \underset{\uparrow}{c.a.e.g + c.b} + \underset{\uparrow}{d.a.e.g + d.b} + \underset{\uparrow}{c.d.a.e.g + c.d.b}$$

insertion de c de b de c.d

$K_{ext2} = a.e.g + c + b.a.e.g + b.c + f.a.e.g + f.c + b.f.a.e.g + b.f.c$

• 3ième étape: intersection des noyaux étendus; cette intersection détecte la plus grande somme de monôme communs à tous les noyaux étendus.

$K_{Bool1} = a.e.g + b.c$

Les fonctions F_1 et F_2, utilisant successivement les noyaux algébriques et les noyaux Booléens étendus, sont:

$$\begin{cases} F_1 = (a.e.g + b).c.d \\ F_2 = (a.e.g + c).b.f \end{cases} \quad \text{et} \quad \begin{cases} F_1 = (a.e.g + b.c).c.d \\ F_2 = (a.e.g + b.c).b.f \end{cases}$$

Comme le noyau Booléen (a.e.g + b.c) est commun à F_1 et F_2, la solution Booléenne donne le résultat suivant:

$F_1 = c.d.SF$

$F_2 = b.f.SF$

$SF = a.e.g + b.c$

L'implantation de cet algorithme exhaustif n'est pas envisageable [Lis87]. Il convient donc d'utiliser des heuristiques. Le nombre de noyaux étendus est égal à: $\Sigma(2^{nbLit(C)})$ pour tous les noyaux algébriques (nbLit(C) est le nombre de littéraux des conoyaux). Une solution heuristique est de n'étendre les noyaux algébriques en utilisant uniquement les littéraux pouvant créer des noyaux Booléens et en se limitant à l'extension par un littéral. Par conséquent la complexité devient inférieure à $\Sigma(nbLit(C))$ pour tous les noyaux algébriques.

IV.3 VALEURS INDÉFINIES

Les techniques d'extraction de valeurs indéfinies (Don't Cares) sont des techniques sophistiquées qui permettent d'obtenir une réduction importante des surfaces actives des circuits. Ces valeurs indéfinies sont de trois types:

- Valeurs indéfinies Externes.
- Valeurs indéfinies internes qui sont liées à la structure interne du réseau Booléen:
  - Valeurs indéfinies de satifaisabilité.
  - Valeurs indéfinies d'observabilité.

IV.3.1 Valeurs Indéfinies Externes (VIE)

Les valeurs indéfinies externes sont des valeurs indéfinies précisées par le concepteur ou imposées par l'environnement. Ce sont des valeurs ne se présentant pas, voir interdites. Un cas particulier concerne, les codes non utilisés lors de la synthsèe d'une machine d'états finis.

A titre d'exemple, considérons un contrôleur de 5 états. Un codage compact utilise 3 bits d'états c'est à dire qu'il existe potentiellement 2^3 états. Ainsi 3 états sont non utilisés.

$$e_1 = 000 \quad e_2 = 001 \quad e_3 = 010 \quad e_4 = 011 \quad e_5 = 100$$

Les états suivants ne correspondent pas à des états utilisés dans la spécification intiale: $e_6 = 101$ $e_7 = 110$ $e_8 = 111$

Les codes de ces pseudo-états e_6, e_7 et e_8 représentent des valeurs indéfinies dans l'équation des variables internes et des sorties de l'automate, et permet une simplification de ces fonctions.

IV.3.2 Valeurs Indéfinies Internes

Cette notion s'applique à un ensemble de fonctions Booléennes déjà décomposées en sous-fonctions partagées ou non et représentées par un réseau Booléen. Un tel réseau est défini comme suit:

- A un Nœud N_i du réseau est associée une sous-fonction Booléenne SF_i.

• Les entrées d'un nœud sont, soit la sortie d'un autre nœud, soit les entrées externes (primaires) du réseau qui sont les variables de l'ensemble des fonctions.

• La sortie d'un nœud N_i est étiquetée par la sous-fonction SF_i.

Dans le graphe, un arc relie un nœud N_i à un nœud N_j ssi $SF_i \in SUPP(SF_j)$. Les entrées primaires $\{e_1, e_2, ..., e_n\}$ et les sorties primaires $\{s_1, s_2, ..., s_n\}$ du réseau Booléen correspondent à des nœuds spéciaux dans le graphe appelés nœuds sources et les nœuds puits.

L'entrance (fanin) d'un nœud N_i est le cardinal de l'ensemble de tous les nœuds origine des arcs dont la destination est N_i.

La sortance (fanout) d'un nœud N_i est le cardinal de l'ensemble de tous les nœuds destination des arcs ayant comme origine N_i.

Exemple:

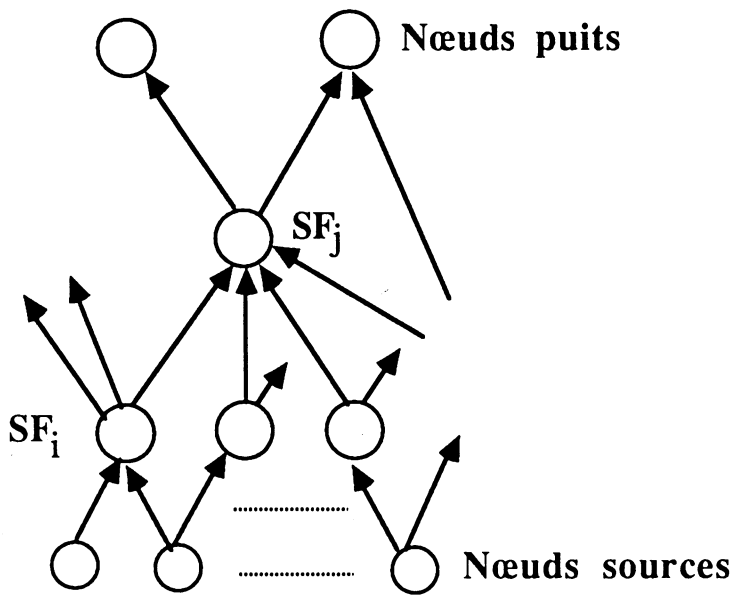


Figure IV.1. Exemple de réseau Booléen

SF_j a une sortance de 2 et une entrance de 3. SF_j apparait dans les expressions correspondants aux nœuds destinations, et l'expression de SF_j est exprimée en fonction des nœuds sources et de SF_i.

IV.3.2.1 Valeurs Indéfinies de Satisfaisabilité (VIS)

Supposons qu'un nœud N_j implantant une sous-fonction SF_j reçoit en entrée la sortie d'un Nœud N_i (figure IV.2).

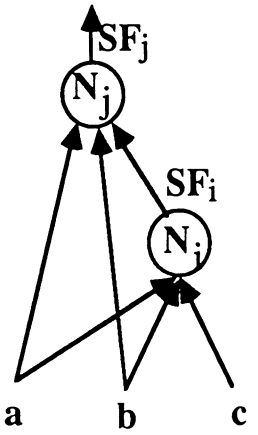


Figure IV.2. Exemple de réseau Booléen

SFj s'exprimera localement comme une sous-fonction qui dépend de a,b,SF_i: $y_j = SF_j(a,b,SF_i)$. Or SF_i est elle même une sous-fonction qui dépend des entrées a,b,c: elle s'écrit $y_i = SF_i(a,b,c)$. Il convient donc de noter que y_i qui est vue comme variable d'entrée pour SF_j est en fait toujours égale à $SF_i(a,b,c)$; ceci veut dire que toutes les combinaisons d'entrées pour SF_j des trois entrées a,b,y_i n'existent pas forcément. En particulier les valeurs couvertes par $y_i \oplus SF_i(a,b,c)$ n'existent pas car $y_i = SF_i(a,b,c)$. On dit que les points couverts par cette fonction ne sont jamais atteints ou utilisés en entrée du nœud N_j. Ils constituent des valeurs non définies pour SF_j et peuvent être utilisées pour simplifier l'expression de SF_j.

Exemple: Soient 2 expressions Booléennes F et SF associées à deux nœuds d'un réseau Booléen telle que:

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

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

L'introduction des valeurs non définies $SF \oplus (a.\overline{b}.c + a.b.\overline{c} + \overline{a}.\overline{b}.\overline{c})$ permet de réduire la complexité du nœud N_F. F s'écrit après minimisation utilisant les valeurs non définies: F = a.c + SF

- ***Définition IV.1: Extension à l'ensemble d'un réseau Booléen***

  Soit N le nombre de nœuds du réseau Booléen. la sous-fonction SF_i est associée au nœud (N_i) du réseau et y_i représente une variable Booléenne simple qui est égale à SF_i.

  On définit une Valeur Indéfinie de Satisfaisabilité associée à un nœud N_i par:

$$VIS_i = y_i \oplus SF_i$$

  On définit les Valeurs Indéfinies de Satisfaisabilité associées à l'ensemble des nœuds du réseau Booléen de la façon suivante:

$$F = \sum_{i=1}^{N} VIS_i$$

Les valeurs indéfinies de satisfaisabilité expriment le fait qu'une variable interne ou variable locale dépend elle-même des entrées externes et qu'en conséquence toutes les combinaisons (variables internes, variables d'entrée) ne peuvent se présenter.

Exemple:

Soit la fonction Booléenne F dont une expression minimale est:

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

On veut diviser F par le diviseur suivant:

$$SF = a + b + c$$

Il est clair qu'aucune mise en facteur algébrique n'est possible. La génération des valeurs indéfinies de satisfaisabilité donne l'expression suivante:

$$VIS = SF \oplus (a + b + c) = \overline{SF}.a + \overline{SF}.b + \overline{SF}.c + SF.\overline{a}.\overline{b}.\overline{c}$$

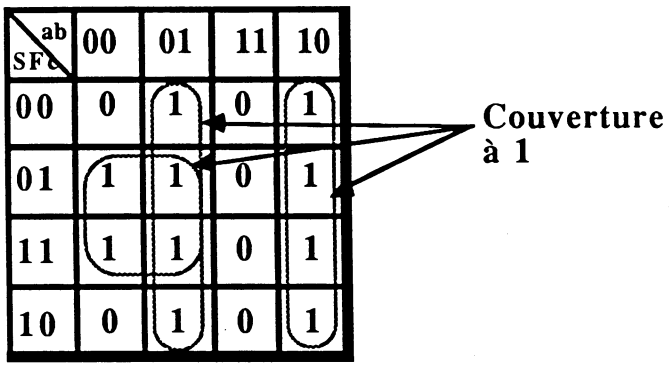
La minimisation de la fonction φ-Booléenne F dont:

la couverture à 1 de la fonction F est: $C_1 = \overline{a}.b + \overline{a}.c + a.\overline{b}$ et

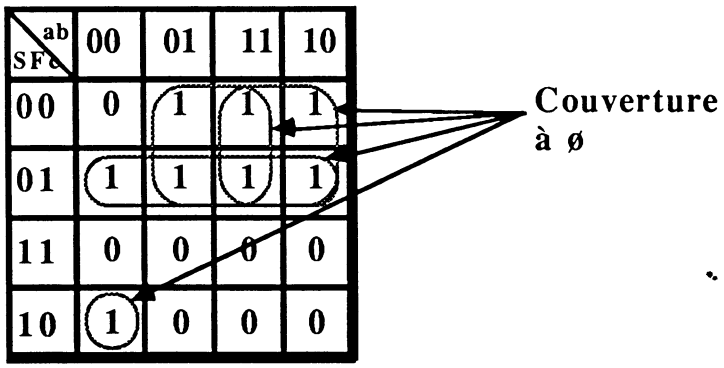
la couverture à ϕ est: $C\phi = \overline{SF}.a + \overline{SF}.b + \overline{SF}.c + SF.\overline{a}.\overline{b}.\overline{c}$

donne le résultat suivant: $mini(C_1, C\phi) = \overline{a}.SF + \overline{b}.SF.$

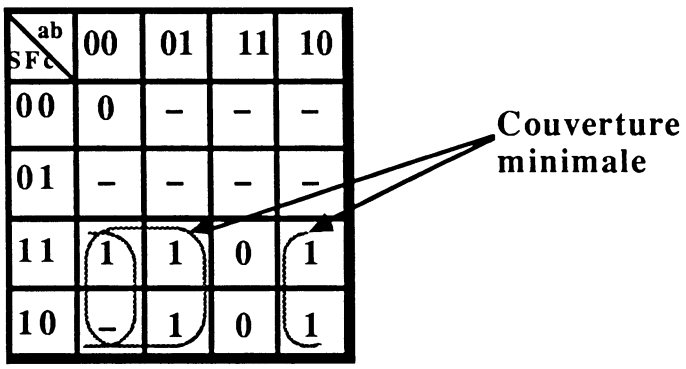
Soit le tableau de Karnaugh suivant représentant la fonction F, c'est à dire ses couvertures à 1 et à 0:



La couverture à ϕ engendrée par les valeurs indéfinies de satisfaisabilité est représentée par le tableau de Karnaugh suivant:



La couverture minimale de la fonction F en nombre de monômes en tenant compte des VIS est précisée sur le tableau de Karnaugh suivant:



L'expression de F devient donc: $F = \overline{a}.SF + \overline{b}.SF$ avec $SF = a + b + c$. Le nombre de littéraux dans ce cas est égal à 4 au lieu de 6 dans l'expression initiale de F.

La figure IV.3 montre le réseau Booléen avant et après l'utilisation des VIS.

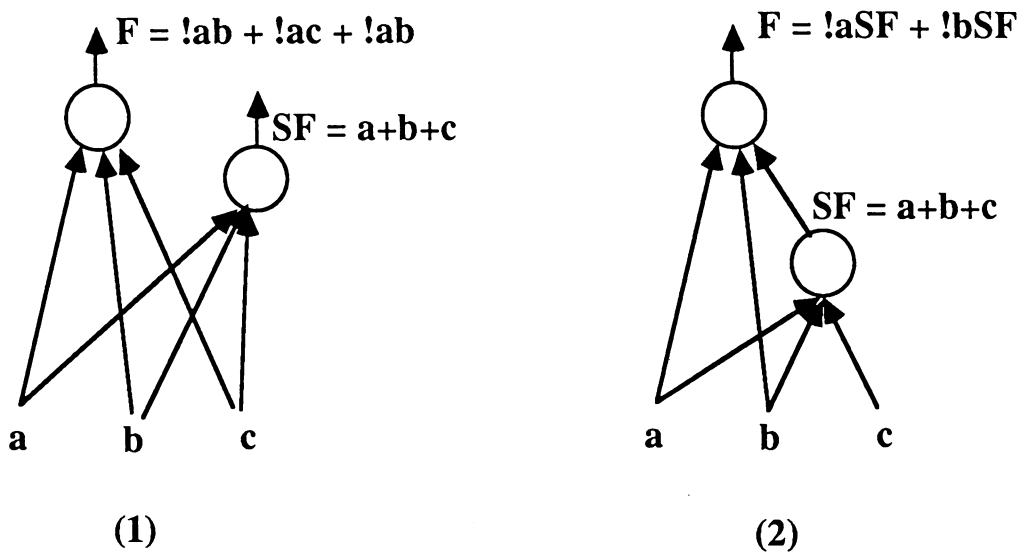


Figure IV.3. Réseau Booléen avant (1) et après (2) utilisation des VIS.

IV.3.2.2 Valeurs Indéfinies d'Observabilité (VIO)

Les Valeurs Indéfinies d'Observabilité sont également utilisées dans les réseaux Booléens pour réduire la complexité des nœuds internes. Ces valeurs permettent de modifier un nœud sans que cette modification soit "observable" de l'extérieur du réseau. Cette notion est fondée sur les définitions de différences Booléennes [Bra90a].

- ***Définition IV.2:***

On définit la différence Booléenne d'une fonction F par rapport à une variable x de la façon suivante:

$$\frac{\partial F}{\partial x} = F_x \oplus F_{\overline{x}}$$

F_x (resp. $F_{\overline{x}}$) est le cofacteur de F par rapport à x (resp. $\overline{x}$). C'est une nouvelle fonction obtenue en remplaçant x (resp. $\overline{x}$) par 1 (resp. 0) dans chaque monôme de F.

- ***Définition IV.3:***

Soit x une variable d'entrée primaire ou interne au réseau Booléen, on dit que x est observable en F si:

$$\frac{\partial F}{\partial x} = F_x \oplus F_{\overline{x}} \neq 0$$

- ***Définition IV.4:***

  On définit les Valeurs Indéfinies d'Observabilité relatives à un nœud ou une sous-fonction ou à une variable d'entrée primaire x par:

$$VIO_x = \pi_i \overline{\frac{\partial F_i}{\partial x}}$$

- ***Définition IV.5:***

  On définit les valeurs indéfinies d'observabilité d'un réseau Booléen par:

$$VIO = \sum_{i=1}^{N} VIO_{x_i}$$

N étant le nombre de nœud dans le réseau Booléen.

Exemple:

Soit une variable interne t ($t = x \oplus y = \overline{x}.y + x.\overline{y}$) du réseau Booléen et U une fonction Booléenne dépendant de t définie par:

$$U = x.y + t.\overline{z} + \overline{t}.\overline{y}$$

les cofacteurs de U par rapport à t et $\overline{t}$ sont:

$$U_t = x.y + \overline{z}$$

$$U_{\overline{t}} = x.y + \overline{y}$$

La différence Booléenne de U par rapport à t est : $\frac{\partial U}{\partial t} = \overline{y}.z + \overline{x}.y.\overline{z}$

$$\Rightarrow \qquad VIO_t = \overline{\frac{\partial U}{\partial t}} = x.y + \overline{y}.\overline{z} + y.z$$

Il est clair que les VIO_t représentent les configurations des variables x,y,z de U pour lesquelles le changement de t n'affecte pas la valeur de U.

La minimisation de t avec les valeurs indéfinies VIO_t donne le résultat suivant: $t = x + y + \overline{z}$

Et nous pouvons vérifier facilement que la fonction initiale U avec $t = x \oplus y$ est égale à la fonction U avec $t = x + y + \overline{z}$:

$$U_{(t = x \oplus y)} = x.y + (\overline{x}.y + x.\overline{y}).\overline{z} + (x + \overline{y}).(\overline{x} + y).\overline{y}$$

$$= x.y + \overline{x}.\overline{y}.z + x.\overline{y}.\overline{z} + \overline{x}.\overline{y}$$

$$= x.(y + \overline{z}) + \overline{x}.(\overline{y} + \overline{z})$$

$$= x.y + \overline{x}.\overline{y} + \overline{z}$$

$$U_{(t = x + y + \overline{z})} = x.y + (x + y + \overline{z}).\overline{z} + \overline{(x + y + \overline{z})}.\overline{y}$$

$$= x.y + \overline{z} + \overline{x}.\overline{y}.z$$

$$= x.y + \overline{x}.\overline{y} + \overline{z}$$

Cela nous permet de minimiser d'une manière générale la complexité des différents nœuds du réseau.

IV.4 DOMAINE D'UTILISATION DES TECHNIQUES BOOLÉENNES

Les techniques Booléennes cherchent uniquement à réduire la complexité des réseaux Booléens en terme de nombre de littéraux. Nous nous proposons d'étudier leur impact sur la connectique du réseau, la surface globale et le chemin critique obtenu après placement routage.

Ces différentes techniques Booléennes développées essentiellement à Berkeley et implantées dans l'outil de référence en synthèse logique multicouche (SIS) ont été appliquées, dans le cadre de cette thèse, sur un grand nombre de circuits tests (benchmarks internationaux [MCN89] et circuits industriels) et une expertise des résultats est effectuée.

Nous allons donner un aperçu des différentes méthodes (scripts) de synthèse utilisés dans SIS. Nous ferons ensuite une comparaison détaillée entre les différentes options proposées par SIS pour expertiser l'intérêt de ces méthodes Booléennes.

IV.4.1 Système de synthèse SIS

Le système SIS est la dernière génération des outils de synthèse multicouche développés à l'université de Berkeley. Il est fondé essentiellement sur les différentes techniques Booléennes données précédemment. Ce système offre un

très grand nombre d'options qui permettent à l'utilisateur de définir lui-même son propre "script".

On peut séparer les différentes options du système SIS en trois parties, la première concerne uniquement les options de factorisation, la deuxième celles de décomposition technologique et la troisième partie concerne les techniques d'optimisation.

IV.4.1.1 Les options de factorisation

Les options de factorisation proposées par le système SIS sont les suivantes:

• Factorisation algébrique: elle est fondée sur les techniques décrites précédemment [Bra87]. Les candidats diviseurs (noyaux, intersection de noyaux, monômes ou sous-monômes communs) impliqués dans cette factorisation sont uniquement des candidats algébriques (cf chapitre I). La division algébrique ainsi que la substitution algébrique sont utilisées.

• Factorisation Booléenne: cette méthode est fondée sur des techniques plus sophistiquées de recherche de candidats Booléens. La phase de minimisation deux couches est faite en utilisant les Valeurs Indéfinies sur les entrées (VIE) [Sal87,Sav90,Sav91a] pour réduire la complexité du réseau Booléen, exprimée en terme de nombre de littéraux [Bra87], et par conséquent la surface active du circuit. La substitution algébrique, dans ce cas, est utilisée pour l'ensemble des nœuds du réseau Booléen et non seulement pour les candidats diviseurs.

• Factorisation poussée (Rugged): cette factorisation est supposée utiliser l'ensemble des techniques Booléennes pour réduire le nombre final de littéraux dans les réseaux Booléens. Cette méthode simplifie chaque nœud du réseau Booléen utilisant l'ensemble des Valeurs Indéfinies: externes, observables et satisfaisables. L'ensemble des Valeurs Indéfinies est généré à chaque étape jusqu'à ce que le nombre final de littéraux du réseau ne puisse plus être réduit.

• Factorisation orientée délai: Cette méthode a été introduite récemment dans l'outil SIS [Tou90,Tou91]. Elle est inspirée d'une méthode de factorisation développée dans [Vas90,Raj90], elle a pour but de réduire la profondeur des réseaux Booléens représentée par le nombre de couches logiques sur le chemin le plus long. Cette réduction de profondeur, faite d'une manière uniforme sans détection du chemin critique du réseau, nécessite dans

certains cas une duplication de la logique dans le réseau pour éviter une augmentation de sa profondeur.

IV.4.1.2 Les options de décomposition technologique (mapping)

Dans le cas du système SIS, les techniques de décomposition technologique utilisées sont fondées sur la décomposition des arbres factorisés et des cellules de la bibliothèque en NAND2 et Inverseurs. Ces techniques utilisent la programmation dynamique [Det87] pour obtenir une surface minimale ou un chemin critique minimal du circuit.

IV.4.1.3 Techniques de filtrage pour l'optimisation de délai des circuits

L'étape d'optimisation de délai dans les réseaux de cellules standards utilise, soit des techniques locales, soit des techniques plus globales. Les techniques locales consistent à localiser certaines zones critiques du réseau des portes logiques et à isoler ces zones pour pouvoir leur appliquer à nouveau les trois étapes de synthèses citées, en tenant compte cette fois-ci des critères d'optimisation.

Pour permettre donc de repousser les limites d'optimisation de circuits, des techniques plus globales, peuvent "revisiter" la structure du réseau Booléen en cherchant à minimiser la profondeur par des réinjections contrôlées de sous-fonctions [Tou91]. Cela permet d'obtenir un faible nombre de cellules sur les chemins les plus longs du réseau qui favorisera la diminution du chemin critique pendant la phase d'optimisation. Dans ce cas, la prédiction de la profondeur dans les réseaux Booléens peut être faite en tenant compte de plusieurs critères, le nombre de couches de sous-fonctions partagées ou le degré des candidats diviseurs (noyaux).

IV.4.2 Résultats expérimentaux

IV.4.2.1 Classification des circuits

Les expériences ont été faites sur grand nombre de circuits tests (MCNC). Pour l'expertise des options utilisées dans le système SIS, ces circuits ont été classés en trois catégories:

- Circuits combinatoires.

• Circuits séquentiels de faible complexité.

• Circuits séquentiels de moyenne complexité.

Remarque:

La limite de complexité pour les circuits séquentiels est celle définie dans le chapitre II.

IV.4.2.2 Analyse de l'impact des techniques de factorisations Booléennes

Les différentes factorisations existantes ont été combinées avec les options de décomposition technologique et les résultats obtenus sur les différents types de circuits ont été analysés en détail. Certaines combinaisons des options de factorisation et de décomposition n'étant pas cohérentes, les résultats ne sont pas donc donnés (factorisation orientée délai suivie d'une décomposition orientée surface par exemple).

Notons que chaque script dans le cas du système SIS a été itéré tant qu'il y avait une amélioration possible.

Ces résultats sont donnés en terme de surface (mils sq) et de chemin critique (ns) obtenu après placement et routage par les outils de COMPASS. La bibliothèque de cellules standards utilisée est la bibliothèque VSC320 1,2µ CMOS de la société "VLSI Technology Inc.".

IV.4.2.3 Evaluation des différentes méthodes pour une optimisation de délai ou de chemin critique

Pour obtenir des circuits plus rapides, la décomposition technologique orientée délai est utilisée. L'expertise consiste à analyser le chemin critique final après placement routage et après estimation des capacités d'interconnexion par les outils de COMPASS en utilisant les différentes factorisations possibles (algébrique, Booléenne, orienté délai et factorisation poussée). Les figures IV.4(a,b,c) montrent les résultats obtenus des chemins critiques donnés en nanosecondes pour les différents types de circuits (combinatoires, séquentiels de faible et moyenne complexité). Pour chaque circuit test, nous avons sélectionné le meilleur chemin critique parmis les quatres solutions obtenues ainsi que la surface correspondante. Ensuite, nous avons calculé le gain moyen pour chaque catégorie de circuits.

Pour les circuits combinatoires, les résultats montrent la supériorité de la factorisation algébrique. Le gain en chemin critique par rapport aux factorisations Booléenne, délai et poussée est respectivement de 83%, 9% et 66%. Un gain en surface est également obtenu par rapport à la factorisation Booléenne et la factorisation délai (respectivement 7% et 26%). Une perte très faible de l'ordre de 5% est obtenue par rapport à la factorisation poussée.

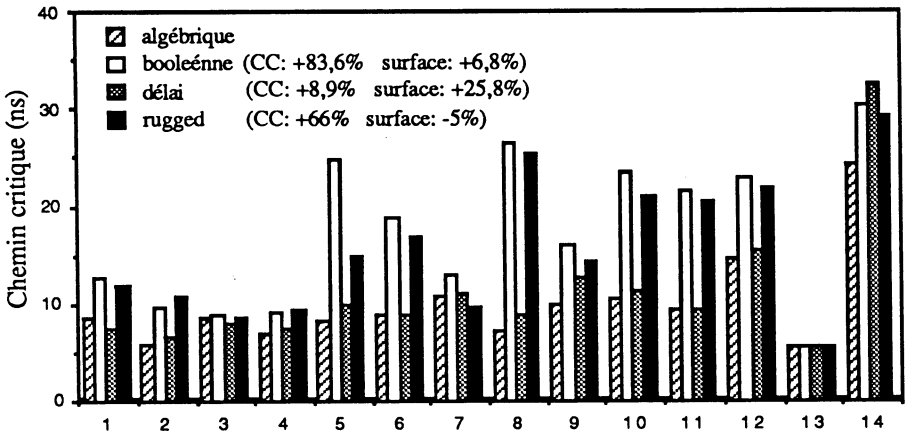


Figure IV.4a. Comparaison en chemin critique des différentes factorisations de SIS pour les circuits combinatoires.

Pour les circuits séquentiels de faible complexité, les meilleurs résultats sont obtenus toujours par la factorisation algébrique suivie par la factorisation orientée délai avec une pénalité en surface totale (10% de perte par rapport à la factorisation algébrique). La factorisation poussée (rugged) ne donne pas de résultats satisfaisants car le gain en surface par rapport à la factorisation algébrique est très faible (gain $\simeq$ 5%), par contre le chemin critique est largement supérieur à celui de la factorisation algébrique (>100%).

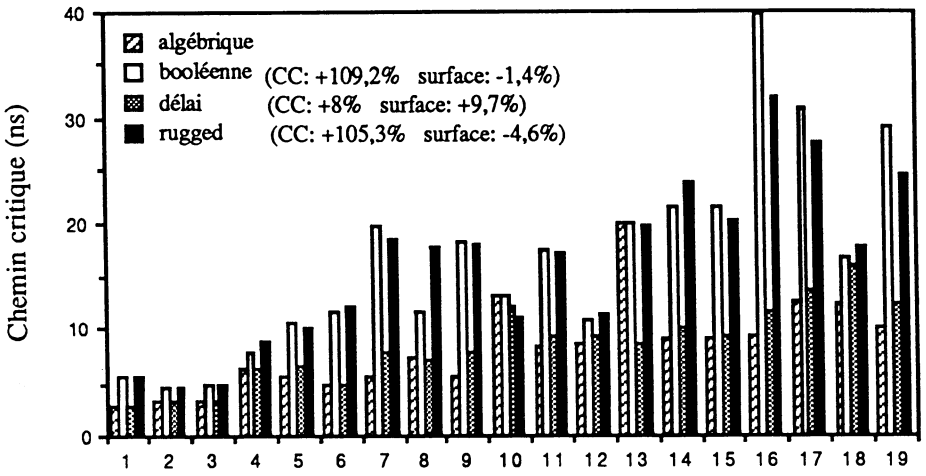


Figure IV.4b. Comparaison en chemin critique des différentes factorisations de SIS pour les circuits séquentiels de faible complexité.

Pour les circuits séquentiels de moyenne complexité, les plus mauvais résultats en chemin critique sont obtenus par la méthode Booléenne: 133% de perte en chemin critique et 5% seulement de gain en surface par rapport à la factorisation algébrique. La factorisation orientée délai ne donne aucun résultat satisfaisant: une perte en chemin critique de 15% et en surface de 18% comparée à la factorisation algébrique. Enfin la factorisation poussée donne des résultats comparables à ceux des circuits séquentiels de faible complexité sauf dans le cas de la surface où le gain est de l'ordre de 13% par rapport à la factorisation algébrique.

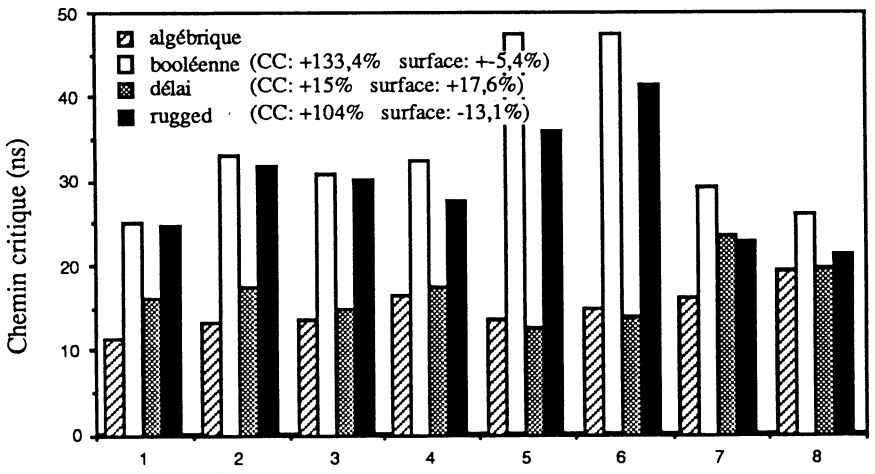


Figure IV.4c. Comparaison en chemin critique des différentes factorisations de SIS pour les circuits séquentiel de moyenne complexité.

En conclusion, l'optimisation en chemin critique des différents types de circuits utilisant les trois méthodes: Booléenne, rugged et délai, ne donne pas de résultats satisfaisants. La factorisation algébrique reste la meilleure solution pour les trois types de circuits, et doit être suivie d'une décomposition technologique orientée délai.

IV.4.2.4 Résultats en surface des factorisations algébrique, Booléenne et l'approche "rugged"

Dans ce cas, une décomposition technologique orientée surface est utilisé. Les résultats sont donnés en figures IV.5(a,b,c) pour les trois méthodes de SIS (algébrique, Booléenne et rugged) et cela pour les trois types de circuits cités précédemment.

les meilleurs résultats pour les circuits combinatoires sont obtenus par la factorisation poussée, qui donne un gain de 10% par rapport à la factorisation algébrique. Par contre, la perte en chemin critique est de l'ordre de 50%. La factorisation Booléenne ne donne pas de gain significatif en surface (3%) avec une perte en chemin critique de 50%.

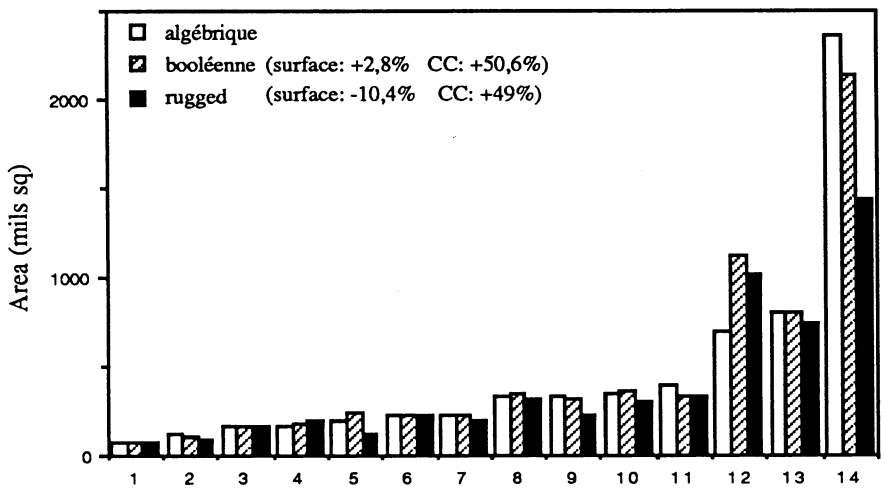


Figure IV.5a. Comparaison en surface des différentes factorisations de SIS pour les circuits combinatoires.

Dans le cas des circuits séquentiels de faible complexité la factorisation algébrique reste toujours la meilleure solution, car les factorisations Booléenne et poussée ne donnent qu'un gain très faible en surface (respectivement 4% et 5%) avec une perte en chemin critique de l'ordre de 100% dans les deux cas.

Cette tendance est confirmée pour les circuits séquentiels de moyenne complexité: un gain inférieur à 13% pour les deux factorisations Booléenne et poussée avec une pénalité supérieure à 110% dans les deux cas.

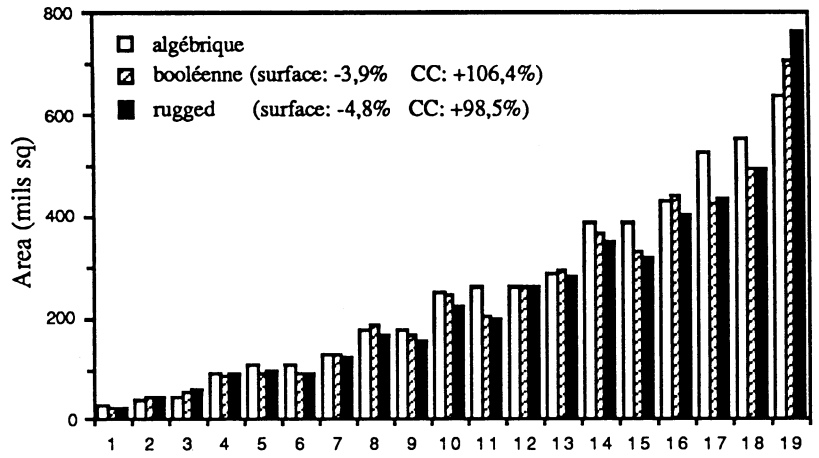


Figure IV.5b. Comparaison en surface des différentes factorisations de SIS pour les circuits séquentiels de faible complexité.

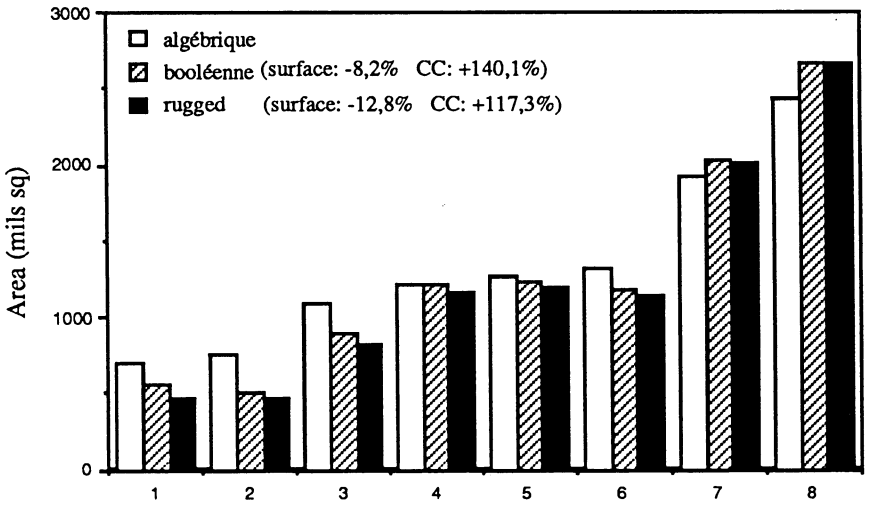


Figure IV.5c. Comparaison en surface des différentes factorisations de SIS pour les circuits séquentiels de moyenne complexité.

IV.5 CONCLUSION

les résultats obtenus pour les trois types de circuits en utilisant les différentes méthodes de factorisation de SIS montrent que la factorisation algébrique reste la solution la plus intéressante, quelque soit le critère d'optimisation. Le gain apporté en surface de la factorisation poussée (rugged) n'est pas très significatif surtout si on tient compte du fait que la perte en chemin critique ainsi que le temps de calcul sont très importants. Cela montre que les efforts faits pour réduire la surface active des circuits en utilisant les techniques Booléennes (extraction des Valeurs Indéfinies) ne sont pas récompensés.

Chapitre V

Résultats et Evaluations des Méthodes Globales de Synthèse

V.1 INTRODUCTION

Comme annoncé au début de cette thèse, ces travaux avaient comme objectif de proposer des techniques de factorisation algébrique restreinte dédiées aux blocs combinatoires complexes. Les trois méthodes retenues sont en résumé, les techniques habituelles de division et substitution algébriques pour les blocs de faible complexité, les techniques de division restreinte par conoyaux pour les circuits de moyenne complexité et les techniques de factorisation lexicographique pour les circuits très complexes. Il est clair que les expériences pratiques ont été faites à l'intérieur du même système de synthèse (ASYL) et dans le même environnement industriel (technologie, outil de placement routage de COMPASS, évaluation de chemin critique par les outils COMPASS). Les expertises de factorisation ont utilisé le même outil de décomposition technologique (sans restriction de niveau). On a donc principalement recherché un environnement expérimental cohérent. Les réserves faites sur les techniques Booléennes se sont appuyées sur les expérimentations entièrement menées dans l'environnement SIS. Dans ce dernier chapitre, des résultats vont être présentés sur un ensemble d'expérimentations menées en utilisant soit le système ASYL, soit le système SIS. Les options, aussi bien de factorisation que de décomposition, seront explorées dans les deux systèmes. Il est donc clair que les résultats obtenus mettent en jeu à la fois les techniques de factorisation et de décomposition. Elle ne permettent pas de conclure de façon précise sur une des techniques, mais donnent une idée de l'efficacité de l'ensemble des approches des deux systèmes. Pour ne pas multiplier les expérimentations, seule la factorisation restreinte (division par conoyaux) est utilisée pour ASYL. On pourra ainsi se faire une opinion de l'intérêt et de la robustesse de cette méthode.

V.2 SYSTEME ASYL

Dans le système ASYL, la méthode de synthèse utilisée est composée de deux étapes: la factorisation et la décomposition technologique.

V.2.1 Méthodes de factorisation

Différentes méthodes de factorisation dans le cadre du système ASYL ont été développées, et cela pour remédier à certains problèmes rencontrés lors de

la phase de synthèse. Ces problèmes sont liés d'une manière générale à la complexité des circuits à synthétiser ou à la nature de la cible technologique choisie. Les différentes options de factorisation sont les suivantes:

• Factorisation algébrique: les deux méthodes de factorisation algébrique détaillées dans les chapitres I et II sont donc implantées dans le système ASYL.

• Factorisation lexicographique, détaillée dans le chapitre III

• Factorisation fondée sur les Diagrammes de Décision Binaire: cette factorisation est utilisée d'une manière générale pour les circuits symétriques, et pour des cibles technologiques spécifiques (cible Actel à base de multiplexeurs).

Comme on l'a signalé en introduction, pour les comparaisons avec le système SIS, on utilise uniquement la factorisation algébrique fondée sur la division par les conoyaux. Par contre des techniques de filtrage ont été utilisée pour l'optimisation de chemin critique. En effet, les chemins critiques dans les circuits dépendent directement du nombre de portes logiques que doivent traverser les signaux. Sur une représentation sous forme d'arbre, le chemin critique sera donc proportionnel à la profondeur de l'arbre. On cherchera par conséquent à créer des arbres les moins profonds possibles.

La profondeur de l'arbre dépend directement des candidats diviseurs (noyaux, monômes et sous-monômes communs) retenus lors de la factorisation. Par conséquent, nous avons utilisé une méthode de filtrage qui consiste à éliminer les candidats diviseurs qui introduisent des chemins très longs (en nombre de couches logiques) dans le réseau Booléen favorisant ainsi des chemins critiques importants. On choisira donc en priorité les candidats qui augmentent la profondeur de l'arbre le moins possible [Sak93].

V.2.2 Les options de décomposition technologique

Dans le cas du système ASYL, deux méthodes sont proposées. La première est utilisée pour une optimisation de la surface du circuit, il s'agit dans ce cas de la surface active du circuit avant placement routage. Dans la deuxième méthode, l'optimisation de chemin critique utilise les méthodes classiques d'optimisation, telles que la duplication de portes logiques sur les chemins les plus longs ainsi que l'utilisation de portes rapides et de buffers [Sak90,Cra91].

V.2.3 Résultats et évaluations

Pour le système SIS, à la lueur des expériences faites dans le chapitre précédent, on retiendra uniquement deux méthodes, la première consiste en une factorisation algébrique suivie d'une décomposition technologique orientée délai (SIS.alg.del) et la deuxième en une factorisation poussée suivie d'une décomposition technologique orientée surface (SIS.rug.S)

Pour le système ASYL, deux solutions sont données. Elle sont fondées uniquement sur la division restreinte par les conoyaux:

- ASY.S.S: optimisation en surface
- ASY.del.del: optimisation en délai ou chemin critique

V.2.3.1 Comparaison en chemin critique

Les résultats de comparaison entre les deux systèmes SIS et ASYL sont montrés sur les figures V.1(a,b). Pour les circuits combinatoires, les techniques de factorisations algébriques de SIS sont légèrement meilleures que celles du système ASYL (3% de gain pour le chemin critique et 9% de gain en surface). Pour les circuits séquentiels de faible complexité, les résultats sont comparables. Par contre pour les circuits séquentiels de moyenne complexité les techniques de factorisation restreinte du système ASYL sont meilleures que celles de SIS (5% de gain en chemin critique et 22% de gain en surface).

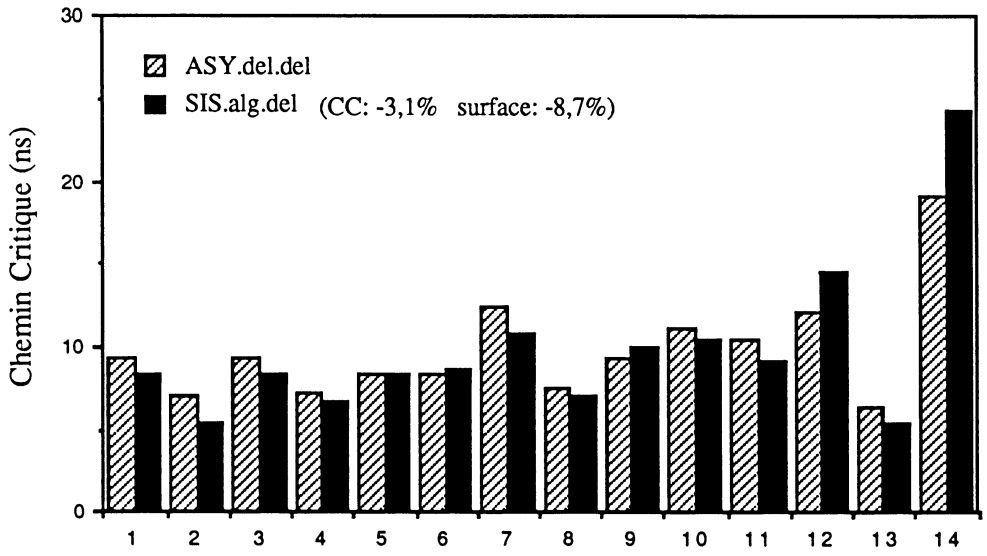


Figure V.1a. Comparaison SIS / ASYL en délai (circuits combinatoires).

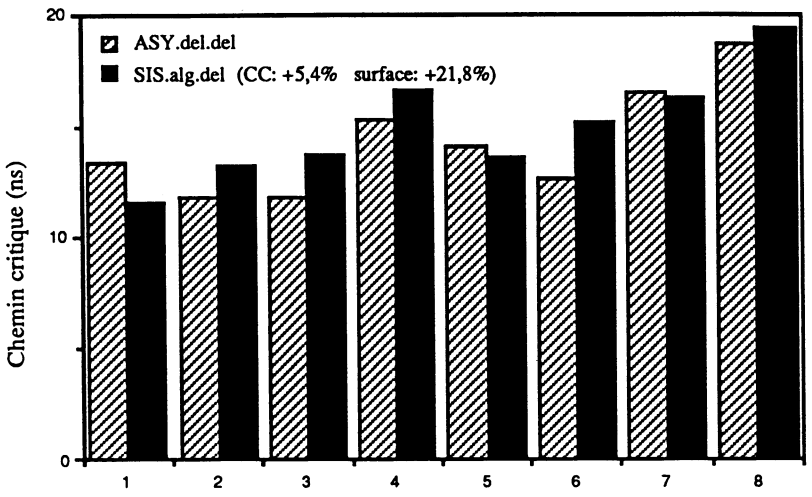


Figure V.1b. Comparaison SIS / ASYL en délai (circuits séquentiels de moyenne complexité).

V.2.3.2 Comparaison en surface

Dans ce cas, les résultats obtenus montrent une supériorité de la factorisation algébrique du système ASYL par rapport aux techniques Booléennes utilisées dans le système SIS et cela, quelque soit le type de circuits choisis (figures V.2a,b,c). Le cas le plus favorable aux techniques Booléennes est celui des circuits combinatoires où le gain en surface par rapport à la factorisation restreinte du système ASYL est de l'ordre de 1% avec une perte en chemin critique de 35% seulement. Pour les circuits séquentiels la perte en surface est de l'ordre de 10% avec une perte en chemin critique supérieure à 50%.

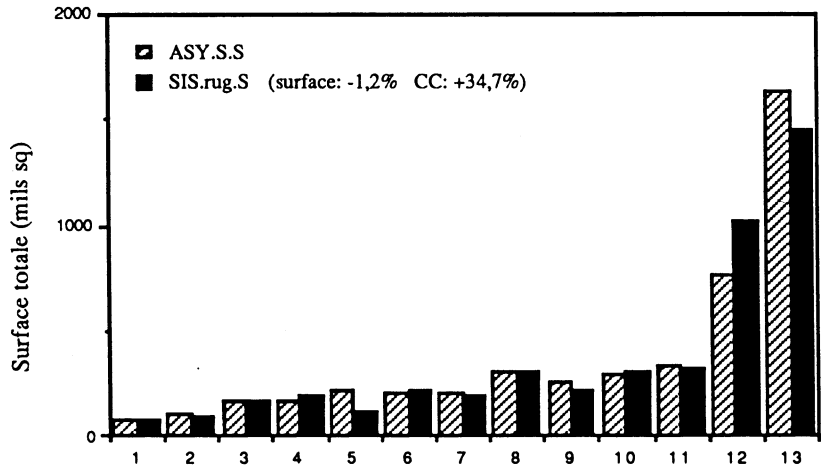


Figure V.2a. Comparaison SIS / ASYL en surface (circuits combinatoires).

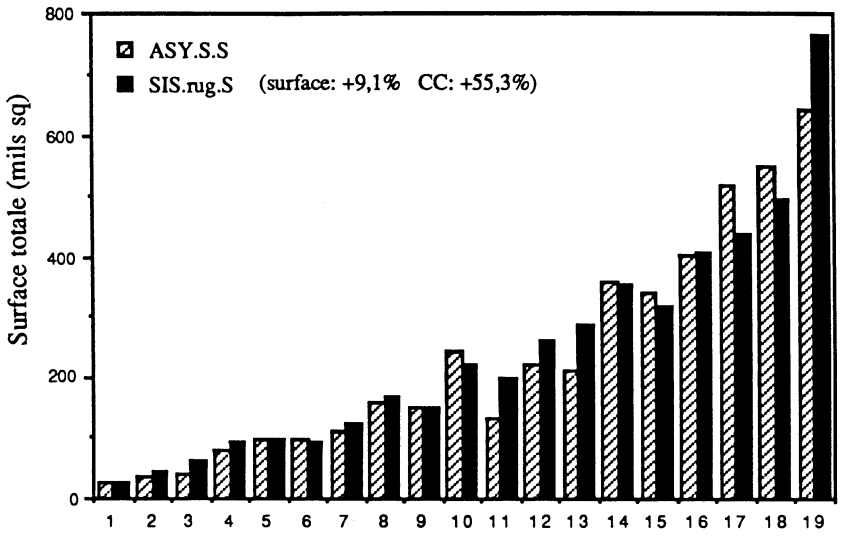


Figure V.2b. Comparaison SIS / ASYL en surface (circuits séquentiels de faible complexité).

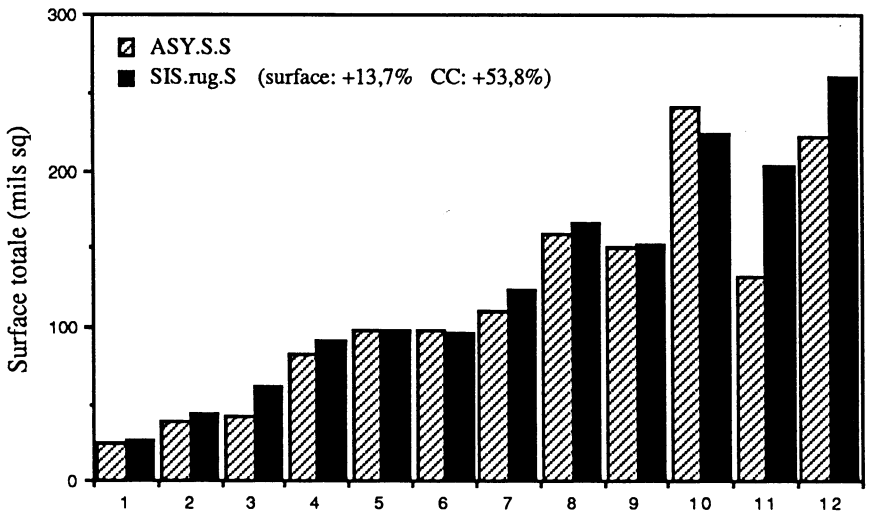


Figure V.2c. Comparaison SIS / ASYL en surface (circuits séquentiels de moyenne complexité).

V.3 CONCLUSION

Les résultats obtenus suite à cette expertise montrent que, si on tient compte des contraintes de conception et de la complexité des circuits dès le début de la synthèse, cela permet d'obtenir de meilleures solutions. Les résultats des techniques Booléennes utilisées dans SIS, qui ont pour but de minimiser la surface, sont équivalent à ceux du système ASYL pour les circuits de faible complexité. Mais pour les grands circuits, les résultats sont assez décevants surtout si on tient compte du fait que ces techniques sont très couteuses en temps de calcul. Par contre quelque soit la complexité et le type de circuit utilisé, le chemin critique est toujours largement supérieur à celui obtenu par le système ASYL.

CONCLUSION GENERALE

Cette thèse a eu comme objectif d'explorer les limites des techniques de factorisation algébrique classiquement proposées ces dernières années face à des blocs combinatoires de haute complexité. De tels blocs existent en particulier pour la synthèse de très gros contrôleurs apparaissant dans les systèmes à contrôle dominant (télécommunications, robotique, automatique, traitement du signal, etc...).

Cette thèse a proposé de réduire la factorisation algébrique à une division par les conoyaux dans un premier temps, puis en allant plus loin, elle a proposé une réduction plus stricte en recherchant une structuration de la logique en couches avec une connectique simplifiée en particulier par rapport aux entrées. Ces méthodes ont été expérimentées de façon très soigneuse en extrayant des grandeurs caractéristiques précises (surface, chemin critique) après placement, routage. Il semble que ceci ait permis d'étendre l'ensemble des méthodes de synthèse logique efficaces que l'on connait aujourd'hui. En effet, si beaucoup de travaux ont concerné des techniques de plus en plus sophistiquées cherchant à réduire la surface de petits blocs combinatoires, peu de travaux à notre connaissance ont concerné l'extension de méthodes aux blocs de très grande complexité et au problème de la préparation de la connectique. Il semble raisonnable de considérer que l'on dispose cette fois d'une panoplie de méthodes réalistes et que face à un bloc combinatoire et dans un ordre de complexité croissante, il convient d'utiliser les méthodes suivantes: factorisation Booléenne, factorisation algébrique, factorisation algébrique restreinte, factorisation lexicographique.

Il est à noter que l'intérêt de la factorisation lexicographique s'est avéré plus large sur d'autres cibles technologiques à granularité plus importante que les cellules standards et ayant des ressources limitées de routage (telle que les cellules XILINX 4000) [Bab92,Cra92,Abo93]. Il est noté également que l'approche lexicographique est fondée sur la recherche d'un ordre des variables d'entrée traduisant les propriétés des fonctions Booléennes

[Sau91,Bou93] et amenant à une représentation optimisée. Cet ordre s'est avéré un très bon ordre pour la représentation des fonctions Booléennes sous forme de graphes de décision binaire ordonnés et a permis une synthèse optimisée sur des cibles technologiques particulières à base de cellules à multiplexeurs (ACTEL) [Bes92a,Bes92b]. Il semble donc, dans un second temps, que cette approche cherchant à extraire un ordre optimisé des entrées a des retombées intéressantes en ce qui concerne la synthèse logique.

BIBLIOGRAPHIE

[Abo90a] P. Abouzeid, K. Sakouti, G. Saucier, F. Poirot "*Multilevel synthesis minimizing the routing factor*", 27th DAC, 24-27 Juin 1990.

[Abo90b] P. Abouzeid, L. Bouchet, K. Sakouti, P. Sicard, G. Saucier, "*Lexicographical Expression of Boolean Function for Multilevel Synthesis of High Speed Circuits*", Sasimi'90, Kyoto, Japan, Octobre 1990.

[Abo91] P. Abouzeid, T. Besson, E. Chotin, M. Crastes, J. Fron, K. Sakouti, G. Saucier, "*Lexicographical Expression of Boolean Functions and Applications to Logic Synthesis*", International Workshop on Logic Synthesis, Research triangle Park, North Carolina (MCNC), USA, Mai 1991.

[Abo92] P. Abouzeid, T. Besson, M. Crastes, J. Fron, K. Sakouti, G. Saucier, "*A global predictive approach to the optimized multilevel synthesis problem*", Sasimi'92, Kobe, Japan, Avril 1992.

[Abo93] P. Abouzeid, B. Babba, M. Crastes, G. Saucier, "*Input Driven Partitioning Methods and Application to Synthesis on PLDs and PGAs*", IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems, à paraître, Février 1993.

[Bab92] B. Babba, M. Crastes, "*Automatic Synthesis on Table Lookup-based PGAs*", EUROASIC'92, pp 25-31, Juin 1992.

[Bar86] K. Bartlett, R. Brayton, R. Jacobi, G. Hachtel, R. Rudell, A. Sangiovanni-Vincentelli and A. Wang, "*Multiple-Level Minimization using implicit Don't Cares*", ICCD'86, pp 552-557, Octobre 1986.

[Bes92a] T. Besson, H. Bouzouzou, M. Crastes, G. saucier,"*Synthesis on Multiplexer-based Programmable Devices using (Ordered) Binary Decision Diagrams*", EUROASIC'92, pp 8-13, Juin 1992.

[Bes92b] T. Besson, H. Bouzouzou, M. Crastes, I. Floricica, G. saucier, "*Synthesis on Multiplexer-based FPGA using Binary Decision Diagrams*", ICCD'92, pp 163-167, Octobre 1992.

[Bos87] D. Bostick, G. Hachtel, R. Jacoby, M.R. Lightner, P. Moceyunas, C. Morisson and D. Ravenscroft, "*The Boulder Optimal Logic Design System*", ICCAD'87, pp62-65, Novembre 1987.

[Bou93] H. Bouzouzou, T. Besson, R. Roane, G. Saucier, "*Input order for ROBDDs based on kernel analysis*", EDAC'93, Février 1993.

[Bra82] R. Brayton, C. T. McMullen. "*The decomposition and factorization of boolean expressions*", ISCAS Proceeding pp 49-54, Mai 1982.

[Bra86] R. Brayton, E. Detjeus, S. Krishna, T. Ma, P. McGeer, L. Pei, N. Phillipps, R. Rudell, R. Segal, A. Wang, R.Yung and A. Sangiovanni-Vincentelli, "*Multiple-level logic optimization system*", pp 356-359, ICCAD 1986, Santa Clara, California, USA..

[Bra87a] R. Brayton, "*Factoring Logic Functions*",IBM J. Research, vol 31, pp 187-198, Mars 1987.

[Bra87b] 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.

[Bra87c] 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.

[Bra90a] R. Brayton, "*Tutorial: Low Level Synthesis of Multi-Level logic*", IFIP Working Conference on Logic and Architecture Synthesis, Paris, France, Juin 1990.

[Bra90b] K. Brace, R. Rudell, R. Bryant, "*Efficient Implementation of a BDD Package*", 27th DAC, Juin 1990.

[Bry86] R. Bryant, "*Graph-based algorithms for Boolean function manipulation*", IEEE trans. on Computers, Vol C35, n° 8, Aout 1986, pp. 677-691.

[Cal91] N. Calazans, R. Jacobi, Q. Zhang, C. Trullemans, "*Improving Binary Decision Diagrams Through Incremental Reduction and Improved Heuristics*", Custom Integrated Circuits Conference, 1991.

[Car91] G. Caruso, "*Near Optimal Factorization of Boolean Functions*", IEEE Transactions on CAD, Vol 10, No 8, pp 1072-1078, Aout 1991.

[Che90] K.C. Chen and S. Muroga, "*Timing Optimization for Multilevel Combinational Network*", in Proc. of the 27th Design Automation Conference (DAC'90), pp 339-344,.Orlando, USA, Juin 1990.

[Cra89] M. Crastes de Paulet, C. Duff, G. Saucier "*ASYL: a Logic and Architecture Design Automation System*", EURO ASIC 89, pp 183-209, Janvier 1989.

[Cra91] M.Crastes, K. Sakouti, G. Saucier, "*A Technology Mapping method Based On Perfect and Semi-Perfect Matchings*", 28th DAC, pp 17-21 Juin 1991.

[Cra92] M. Crastes, B. Babba, G. Saucier, "*Input driven Synthesis on PLDs and PGAs*", EDAC'92, pp 48-52 Mars 1992.

[Dam90a] M. Damiani, G. De Micheli, "*Efficient Computation of Exact and simplified Observability Don't Care Sets for Multilevel Combinational Networks*", IFIP Working conference on Logic and Architecture Synthesis, Paris, France, Juin1990.

[Dam90b] M. Damiani, G. De Micheli, "*Observability Don't Care Sets and Boolean Relations*", ICCAD'91, pp. 502-505, Santa Clara, USA, Novembre 1990.

[Det87] E. Detjeus, G. Gannot, "*Technology Mapping in MIS*", in Proc. Int. Conference on Computer Aided Design (ICCAD'87), pp. 116-119, Santa Clara, USA, Novembre 1987.

[Fri87] S. Friedman, K. Supowit, "*Finding the Optimal Variable Ordering for Binary Decision Diagrams*", 24th ACM/IEEE Design Automation Conference, Juin 1987.

[Gre86] D. Gregory, K. Bartlett, A. de Gueus, G. Hachtel, "*Socrates: A system automatically synthesizing and optimizing combinational logic*", 23th ACM/IEEE Design Automation Conference, Juin 1986.

[Hsu92] W.J. Hsu and W.Z. Shen, "*Coalgebraic Division for Multilevel Logic Synthesis*", 29th DAC, pp 438-442, Juin 1992.

[Keu87] K. Keutzer, "*Dagon: Technology Binding and Local Optimization*", DAC'87, pp341-347, Juin 1987.

[Kun68] J. Kuntzmann. "*Algèbre de Boole*", Edition Dunod, Paris, 1968.

[Leg88] M.C. Lega, "*Mapping properties of multi-level logic synthesis operations*", ICCD'88, pp 257-261, Octobre 1988.

[Lis87] R. Lisanke, F. Berglez, G. Kedem, "*DECAF: Decomposition and Factoring for Multilevel Logic Synthesis*", Microelectronics Center of North Carolina, MCNC'87, Mai 1987.

[Lis88] R. Lisanke, F. Berglez, G. Kedem, "*McMAP: A Fast Technology Mapping Procedure for Multi-Level Logic Synthesis*", ICCD'88, pp 252-256 Octobre 1988.

[MCN89] Microelectronics Center of North Carolina, Standard Synthesis Benchmarks, International Workshop on Logic Synthesis, Mai 1989.

[Poi90] F. Poirot, P. Abouzeid, G. Saucier, "*Lexicographical factorization minimizing the critical path and the routing factor of multilevel logic*", IFIP Working Conference on Logic and Architecture Synthesis, Paris, France, Juin 1990.

[Raj90] J. Rajski and J. Vasudevamurthy, "*Testability Preserving Transformation in Multi-level Logic Synthesis*", ITC'90, pp 265-273, Septembre 1990.

[Sak90] K. Sakouti, G. Saucier. "*A fast and effective technology mapper on an autodual library of standard cells*", IFIP Working Conference on Logic and Architecture Synthesis, Paris, France, Mai 1990.

[Sak93] K. Sakouti, P. Abouzeid, M. Belrhiti, M. Crastes, G. Saucier, "*Coherent Optimization Strategies for Multilevel Synthesis*", EDAC'93, Février 1993.

[Sal89] A. Saldanha, A. Wang, R. Brayton, A. Sangiovanni-Vincentelli, "*Multilevel Logic Simplification using Don't Cares and Filters*", in Proc. 26th Design Automation Conference (DAC'89), Juin 1989.

[Sat87] H. Sato and al., "*Boolean resubstitution with Permissible functions and Binary Decision Diagrams*", in Proc. 27th Design Automation Conference (DAC'87), pp 284-289, Juin 1987.

[Sau91] G. Saucier, F. Poirot, "*A Good Input Ordering for Circuit verification Based on Binary Decision Diagrams*", EUROASIC'91, pp 385-387, Juin 1991.

[Sau92a] G. Saucier, "*Analysis of the Trends in Logic Synthesis* ", Sasimi'92, Kobe, Japan, Avril 1992.

[Sau92b] G. Saucier, P. Abouzeid, T. Besson, J. Fron, K. Sakouti, "*A coherent Area/Timing Optimization Strategies for Multilevel Logic*", TAU'92,Workshop on Timing Issues in the Specification and Synthesis, Princeton University, USA, Mars 1992.

[Sav90] H. Savoj, R. Brayton, "*The use of Observability and External Don't Cares for the Simplification of Multilevel Networks*", 27th DAC, 24-27 Juin 1990.

[Sav91a] H. Savoj, R. Brayton, H.J. Touati, "*Extracting local don't cares for network optimization*", in Proc. Int. Conference on Computer Aided Design (ICCAD'91), pp. 514-517, Santa Clara, USA, Novembre 1991.

[Sav91b] H. Savoj, R. Brayton, "*Observability Relations and Observability Don't Cares*", ICCAD'91, pp 518-521, Juin 1991.

[Sin88] K.J. Singh, AR. Wang, RK Brayton, A.Sangiovanni-Vincentelli, "*Timing Optimization of Combinational Logic*",ICCAD'88, pp 282-285, Novembre 1988.

[Sin90] K.J. Singh, A.Sangiovanni-Vincentelli, "*A Heuristic Algorithm for the Fanout Problem*", in Proc. 27th Design Automation Conference (DAC'90), pp 357-360,.Orlando, USA, Juin 1990.

[Tou90] H.J.Touati, "Performance-Oriented Technology Mapping", Thèse de Doctorat, université de Berkeley, 28 Novembre 1990.

[Tou91] H.J.Touati, H.Sajov, R.Brayton, "*Delay Optimization of Combinational Logic Circuit by Clustering and Partial Collapsing*", in Proc. Int. Conference on Computer Aided Design (ICCAD'91), pp. 188-191, Santa Clara, USA, Novembre 1991.

[Vas90] J. Vasudevamurthy and J. Rajski, "*A Method for Concurrent Decomposition and Factorization of Booleans Expressions*", ICCAD'90, pp 510-513, Novembre 1990.

[Yos91] K. Yoshikawa & all, "*Timing optimization on mapped circuits*", 28th DAC, pp 112-117, Juin 1991.

Annexe

Tableaux de Comparaisons entre la Méthode de Division Restreinte et la Méthode de Division et Substitution Algébriques

	Meth2	Meth1		Meth2	Meth1	
Name	S_T	S_T	G(%)	CC	CC	G(%)
a5xp1	200	209	4,07	11,20	12,90	13,18
bbsse	233	247	5,63	12,00	11,00	-9,09
bbtas	38	40	3,54	5,20	5,10	-1,96
bw	339	327	-3,82	12,10	12,70	4,72
dk15	117	118	0,43	7,20	8,90	19,10
dk17	102	106	4,33	7,80	7,30	-6,85
donfile	180	177	-1,35	10,00	11,40	12,28
f51m	227	230	1,09	10,90	10,70	-1,87
frg1	240	236	-1,99	11,20	14,90	24,83
kirkman	236	262	10,08	11,10	12,30	9,76
misex2	168	169	0,30	7,00	7,20	2,78
rd53	76	79	4,16	7,40	8,80	15,91
rd73	231	240	4,04	11,50	12,20	5,74
sao2	248	288	13,94	12,80	15,20	15,79
vg2	151	155	3,15	9,40	8,80	-6,82
z4ml	75	76	1,45	9,10	9,30	2,15
MOY			**3,07**			**6,23**

Tableau A.1. Résultats comparatifs entre les méthodes 1 et 2 de la factorisation algébrique appliquées aux circuits de faible complexité (décomposition orientée surface).

Meth1: méthode de division par les conoyaux
Meth2: méthode de division et de substitution algébriques.
S_T : surface totale du circuit (en mils sq)
CC : chemin critique (en ns)
G(%) : gain de la méthode 2 par rapport à la méthode 1

$$\text{Gain } \% = 100 * \left(\frac{\text{résultat de (1) - Résultat de (2)}}{\text{Résultat de (1)}} \right)$$

	Meth2	Meth1		Meth2	Meth1		Meth2	Meth1	
Name	F_r	F_r	G(%)	S_r	S_r	G(%)	S_a	S_a	G(%)
a5xp1	0,94	0,86	-9,30	97	97	-0,52	103	112	8,02
bbsse	1,05	1,08	2,78	119	128	6,94	114	119	4,30
bbtas	0,61	0,66	7,58	15	16	7,64	24	24	0,00
bw	1,23	1,15	-6,96	187	175	-7,03	152	152	-0,07
dk15	0,85	0,88	3,41	54	55	2,18	63	63	-1,28
dk17	0,74	0,85	12,94	43	49	11,66	58	58	-1,57
donfile	0,86	0,86	0,00	83	82	-1,47	97	95	-1,36
f51m	0,91	0,91	0,00	108	110	1,10	119	120	1,08
frg1	1,02	0,94	-8,51	121	114	-6,30	119	121	1,98
kirkman	0,94	0,96	2,08	114	128	10,99	121	134	9,20
misex2	0,70	0,72	2,78	69	71	1,98	99	98	-0,92
rd53	0,60	0,64	6,25	29	31	8,06	48	48	1,86
rd73	0,94	1,02	7,84	112	121	7,99	119	119	0,17
sao2	1,02	1,18	13,56	125	156	19,73	123	132	7,11
vg2	0,88	0,76	-15,79	71	67	-5,07	80	88	9,29
z4ml	0,64	0,56	-14,29	29	27	-6,99	46	49	6,38
MOY			**0,27**			**3,18**			**2,76**

Tableau A.2. Résultats comparatifs entre les méthodes 1 et 2 de la factorisation algébrique appliquées aux circuits de faible complexité (décomposition orientée surface).

Meth1: méthode de division par les conoyaux
Meth2: méthode de division et de substitution algébriques.
S_r : surface de routage du circuit (en mils sq)
S_a : surface active du circuit (en mils sq)
F_r : facteur de routage
G(%) : gain de la méthode 2 par rapport à la méthode 1

	Meth2	Meth1		Meth2	Meth1	
Name	S_T	S_T	G(%)	CC	CC	G(%)
a5xp1	236	251	5,78	8,50	9,30	8,60
bbsse	263	294	10,42	7,90	7,30	-8,22
bbtas	43	44	3,17	4,30	4,30	0,00
bw	360	412	12,49	7,10	6,70	-5,97
dk15	125	128	2,66	5,50	6,80	19,12
dk17	115	108	-6,86	6,30	5,80	-8,62
donfile	189	172	-10,06	8,60	10,00	14,00
f51m	279	274	-1,86	8,60	8,40	-2,38
frg1	281	244	-15,20	10,20	15,00	32,00
kirkman	248	270	7,97	9,30	10,00	7,00
misex2	218	227	3,58	5,40	5,10	-5,88
rd53	97	104	6,94	6,20	7,10	12,68
rd73	241	240	-0,58	9,90	10,50	5,71
sao2	283	304	6,98	9,30	11,50	19,13
vg2	160	188	14,92	8,30	8,10	-2,47
z4ml	86	98	12,05	7,80	7,80	0,00
MOY			**3,27**			**5,29**

Tableau A.3. Résultats comparatifs entre les méthodes 1 et 2 de la factorisation algébrique appliquées aux circuits de faible complexité (décomposition orientée vitesse).

Meth1: méthode de division par les conoyaux
Meth2: méthode de division et de substitution algébriques.
S_T : surface totale du circuit (en mils sq)
CC : chemin critique (en ns)
G(%) : gain de la méthode 2 par rapport à la méthode 1

	Meth2	Meth1		Meth2	Meth1		Meth2	Meth1	
Name	F_r	F_r	G(%)	S_r	S_r	G(%)	S_a	S_a	G(%)
a5xp1	0,97	0,83	-16,87	116	114	-2,28	120	137	12,47
bbsse	1,07	0,93	-15,05	136	142	3,89	127	152	16,49
bbtas	0,66	0,72	8,33	17	19	8,11	26	26	-0,39
bw	0,93	1,06	12,26	174	212	18,04	187	200	6,56
dk15	0,77	0,84	8,33	54	58	7,19	70	70	-1,15
dk17	0,73	0,70	-4,29	49	44	-9,46	67	63	-5,05
donfile	0,86	0,81	-6,17	88	77	-13,64	102	95	-7,16
f51m	0,93	0,93	0,00	134	132	-1,82	144	142	-1,83
frg1	0,99	0,96	-3,13	140	120	-17,07	141	125	-13,49
kirkman	0,94	0,96	2,08	120	132	8,93	128	138	6,98
misex2	0,86	0,88	2,27	101	106	4,72	117	121	2,57
rd53	0,74	0,78	5,13	41	46	9,67	56	58	4,80
rd73	0,88	0,91	3,30	113	114	1,14	128	126	-2,23
sao2	0,96	1,07	10,28	138	157	11,85	144	147	1,77
vg2	0,81	0,81	0,00	72	84	14,95	89	104	14,99
z4ml	0,67	0,70	4,29	35	40	14,39	52	58	10,59
MOY			**0,67**			**3,66**			**2,87**

Tableau A.4. Résultats comparatifs entre les méthodes 1 et 2 de la factorisation algébrique appliquées aux circuits de faible complexité (décomposition orientée vitesse).

Meth1: méthode de division par les conoyaux
Meth2: méthode de division et de substitution algébriques.
S_r : surface de routage du circuit (en mils sq)
S_a : surface active du circuit (en mils sq)
F_r : facteur de routage
G(%) : gain de la méthode 2 par rapport à la méthode 1

	Meth2	Meth1		Meth2	Meth1	
Name	S_T	S_T	G(%)	CC	CC	G(%)
cache_c	1876	1851	-1,35	28,50	26,70	-6,74
jay	727	695	-4,63	18,80	16,30	-15,30
planet1	1120	1094	-2,39	22,80	21,50	-6,05
planet	1089	1071	-1,74	22,50	18,80	-19,70
platcos	799	793	-0,73	12,60	12,20	-3,28
s1	717	719	0,21	15,50	14,20	-9,15
sand	1326	1224	-8,29	22,70	19,30	-17,60
scf	2049	1892	-8,27	35,30	28,10	-25,60
styr	941	956	1,56	19,70	20,00	1,50
tbk	784	780	-0,50	18,80	19,00	1,05
zeegers	4048	3766	-7,47	52,00	37,30	-39,40
MOY			**-3,06**			**-12,75**

Tableau A.5. Résultats comparatifs entre les méthodes 1 et 2 de la factorisation algébrique appliquées aux circuits de moyenne complexité (décomposition orientée surface).

Meth1: méthode de division par les conoyaux
Meth2: méthode de division et de substitution algébriques.
S_T : surface totale du circuit (en mils sq)
CC : chemin critique (en ns)
G(%) : gain de la méthode 2 par rapport à la méthode 1

	Meth2	Meth1		Meth2	Meth1		Meth2	Meth1	
Name	F_r	F_r	G(%)	S_r	S_r	G(%)	S_a	S_a	G(%)
cache_c	1,52	1,29	-17,83	1131	1042	-8,52	744	808	7,91
jay	1,27	1,13	-12,39	407	369	-10,30	320	326	1,81
planet1	1,25	1,09	-14,68	622	571	-9,06	498	524	4,89
planet	1,23	1,13	-8,85	601	568	-5,76	488	503	2,83
platcos	1,26	1,24	-1,61	445	439	-1,46	354	354	0,17
s1	1,25	1,21	-3,31	398	393	-1,27	319	325	1,97
sand	1,47	1,19	-23,53	789	665	-18,60	537	559	3,99
scf	1,53	1,23	-24,39	1239	1044	-18,70	810	849	4,56
styr	1,16	1,16	0,00	505	513	1,56	436	443	1,56
tbk	1,21	1,12	-8,04	429	412	-4,15	355	368	3,59
zeegers	1,98	1,56	-26,92	2689	2295	-17,20	1358	1471	7,68
MOY			**-12,87**			**-8,50**			**3,72**

Tableau A.6. Résultats comparatifs entre les méthodes 1 et 2 de la factorisation algébrique appliquées aux circuits de moyenne complexité (décomposition orientée surface).

Meth1: méthode de division par les conoyaux

Meth2: méthode de division et de substitution algébriques.

S_r : surface de routage du circuit (en mils sq)

S_a : surface active du circuit (en mils sq)

F_r : facteur de routage

G(%) : gain de la méthode 2 par rapport à la méthode 1

	Meth2	Meth1		Meth2	Meth1	
Name	S_T	S_T	G(%)	CC	CC	G(%)
cache_c	1963	1997	1,69	18,60	15,40	-20,78
jay	771	698	-10,44	10,80	10,70	-0,93
planet1	1161	1134	-2,31	12,80	12,00	-6,67
planet	1220	1106	-10,35	11,80	11,40	-3,51
platcos	847	790	-7,28	9,90	8,20	-20,73
s1	740	741	0,09	11,10	12,60	11,90
sand	1384	1284	-7,77	13,80	19,10	27,75
scf	2243	1986	-12,92	15,30	14,90	-2,68
styr	987	984	-0,32	14,70	13,20	-11,36
tbk	837	798	-4,91	15,90	15,80	-0,63
zeegers	4061	3641	-11,54	20,90	18,20	-14,84
MOY			**-6,01**			**-3,86**

Tableau A.7. Résultats comparatifs entre les méthodes 1 et 2 de la factorisation algébrique appliquées aux circuits de moyenne complexité (décomposition orientée vitesse).

Meth1: méthode de division par les conoyaux
Meth2: méthode de division et de substitution algébriques.
S_T : surface totale du circuit (en mils sq)
CC : chemin critique (en ns)
G(%) : gain de la méthode 2 par rapport à la méthode 1

	Meth2	Meth1		Meth2	Meth1		Meth2	Meth1	
Name	F_r	F_r	G(%)	S_r	S_r	G(%)	S_a	S_a	G(%)
cache_c	1,48	1,36	-8,82	1172	1151	-1,81	792	846	6,44
jay	1,15	1,03	-11,65	412	354	-16,43	359	344	-4,27
planet1	1,24	1,10	-12,73	642	594	-8,11	518	540	4,07
planet	1,31	1,15	-13,91	692	591	-16,99	528	514	-2,70
platcos	1,33	1,19	-11,76	484	429	-12,70	364	361	-0,83
s1	1,11	1,17	5,13	389	400	2,53	351	341	-2,75
sand	1,40	1,24	-12,90	807	711	-13,58	577	573	-0,58
scf	1,53	1,28	-19,53	1357	1115	-21,64	887	871	-1,77
styr	1,21	1,15	-5,22	540	526	-2,68	447	458	2,43
tbk	1,29	1,10	-17,27	472	418	-12,82	366	380	3,79
zeegers	1,86	1,42	-30,99	2641	2136	-23,62	1420	1505	5,62
MOY			**-12,70**			**-11,62**			**0,86**

Tableau A.8. Résultats comparatifs entre les méthodes 1 et 2 de la factorisation algébrique appliquées aux circuits de moyenne complexité (décomposition orientée vitesse).

Meth1: méthode de division par les conoyaux.
Meth2: méthode de division et de substitution algébriques.
S_r : surface du routage du circuit (en mils sq)
S_a : surface active du circuit (en mils sq)
F_r : facteur de routage
G(%) : gain de la méthode 2 par rapport à la méthode 1

NAME	$S_T(1)$	$S_T(2)$	G(%)	$F_r(1)$	$F_r(2)$	G(%)	CC(1)	CC(2)	G(%)
planet1	1109	1147	-3,41	1,27	1,08	14,96	25,10	21,20	15,54
planet	1136	1111	2,26	1,34	1,09	18,66	20,80	20,60	0,96
s1	682	650	4,71	1,25	1,01	19,20	14,90	15,00	-0,67
sand	1219	1262	-3,54	1,29	1,17	9,30	21,70	20,40	5,99
scf	1963	1934	1,50	1,48	1,18	20,27	33,10	30,10	9,06
zeegers	3947	3618	8,32	1,95	1,35	30,77	49,10	40,20	18,13
cache_ctr	1873	1966	-4,97	1,55	1,25	19,35	27,80	25,70	7,55
jay	677	664	1,91	1,22	1,01	17,21	17,20	15,30	11,05
platcos	799	799	-0,01	1,26	1,26	0,00	12,80	12,60	1,56
styr	949	969	-2,03	1,24	1,11	10,48	18,30	19,70	-7,65
tbk	800	807	-0,77	1,28	1,09	14,84	21,00	21,50	-2,38
MOY			**0,36**			**15,91**			**5,38**

Tableau A.9. Résultats comparatifs avant et après la prise en compte du facteur de routage appliquée aux circuits de moyenne complexité (décomposition orientée surface).

S_T : surface totale du circuit (en mils sq)
F_r : facteur de routage
CC : chemin critique (en ns)
(1) avant l'application du filtre, (2) aprés
G(%) : gain de (2) par rapport (1)

NAME	$S_r(1)$	$S_r(2)$	G(%)	$S_a(1)$	$S_a(2)$	G(%)
planet1	621	596	4,03	489	551	-12,85
planet	651	579	10,99	486	531	-9,41
s1	379	326	13,81	303	323	-6,67
sand	687	681	0,87	532	582	-9,30
scf	1172	1047	10,66	792	887	-12,05
zeegers	2609	2079	20,32	1338	1540	-15,09
cache_ctr	1138	1092	4,07	734	874	-18,97
jay	372	334	10,27	305	331	-8,40
platcos	445	445	0,00	354	354	0,00
styr	525	510	3,03	424	459	-8,33
tbk	449	421	6,39	351	386	-9,94
MOY			**7,68**			**-10,09**

Tableau A.10. Résultats comparatifs avant et après la prise en compte du facteur de routage appliquée aux circuits de moyenne complexité (décomposition orientée surface).

S_r : surface de routage du circuit (en mils sq)
S_a : surface active du circuit (en mils sq)
(1) avant l'application du filtre, (2) aprés
G(%) : gain de (2) par rapport (1)

Résumé

Cette thèse propose des méthodes de synthèse dédiées aux circuits intégrés complexes. Elle concerne l'étape de factorisation algébrique dont le but est de réduire la complexité des expressions Booléennes évaluées en terme de littéraux. Les méthodes classiques généralement proposées, amènent une bonne minimisation de la surface active mais peuvent entraîner un mauvais contrôle de la connectique. Cette thèse présente d'abord l'état de l'art critique sur les techniques de factorisation algébrique poussées incluant les techniques dites Booléennes. Dans la suite, deux approches alternatives de factorisation plus restreintes sont proposées. La première est réduite à une division par conoyaux et la deuxième concerne une factorisation dite lexicographique encore plus restrictive, dont le but est de préparer une connectique simplifiée. Les résultats expérimentaux ont permis de définir à partir de quel seuil de complexité, il convient d'appliquer ces deux méthodes pour obtenir une bonne surface globale ainsi qu'un bon facteur de routage.

Mots-clés

Synthèse logique multicouche, factorisation algébrique, factorisation Booléenne, synthèse sur cellules standards, factorisation lexicographique.