Synthèse et décomposition technologique sur réseaux programmables et ASICs

Gilles Bosco

THESE

Présentée par

Gilles Bosco

pour obtenir le grade de **DOCTEUR**

De l'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

(Arrêté ministériel du 30 Mars 1992)

(Spécialité: Micro-électronique)

==========

Synthèse et décomposition technologique sur réseaux programmables et ASICs

==========

Date de soutenance : 16 décembre 1996.

Composition du jury:

Pr. Guy Mazaré : Président
Pr. Gabrièle Saucier : Directrice de thèse
Pr. Giovanni De Micheli : Rapporteur
Pr. Wolfgang Rosenstiel : Rapporteur
M. Jean-François Agaesse

Avant-propos

je tiens tout d'abord à remercier Madame Gabrièle Saucier, Professeur à l'ENSIMAG et directrice du Laboratoire de Conception de Systèmes Intégrés, pour m'avoir accueilli dans son laboratoire et avoir guidé mes recherches.

Je remercie également:

Monsieur Guy Mazaré, Professeur et directeur de l'ENSIMAG d'avoir accepté de me faire l'honneur de présider le jury.

Monsieur Giovanni De Micheli, Professeur à l'Université de Stanford, d'avoir accepté d'être rapporteur de cette thèse.

Monsieur Wolfgang Rosenstiel, Professeur à l'Université de Tülingen, d'avoir accepté d'être rapporteur de cette thèse.

Monsieur Jean-François Agaesse, Responsable CAO Thomson TCS, d'avoir accepté d'être membre de ce jury.

Je remercie tous mes collègues du CSI pour leur accueil chaleureux lors de mon arrivée, pour l'aide qu'ils m'ont apporté pour réaliser ce travail ainsi que pour l'agréable ambiance du laboratoire. Je remercie tout particulièrement Mohamed Belrhiti avec qui j'ai eu le plaisir de collaborer durant cette thèse et de partager des travaux, Aadil Amoura pour son aide, Ali, Vincent, Viet, Khalid, Marie-Claude et tous ceux qui ont contribue à l'élaboration de ce travail. Un grand merci a Martine et Pascale pour leur gentillesse et leurs sourires quotidiens.

Je remercie mes parents pour le soutien moral et financier apporté durant ces années de thèse.

Je tiens à remercier tous ceux qui par leur gentillesse et leurs témoignages d'amitié m'ont soutenu, encouragé et accompagné dans la tumultueuse vie du thésard. Mention spéciale pour Pascale qui ne viendra pas à la soutenance, ma chère Behnaz, Pascal qui a su me donner le goût au travail. Merci encore à Laurent et Anne pour leur fidélité, à Eric qui m'a si bien nourri, à Nathalie (qui devrait acheter une table plus solide), à la petite Virginie qui deviendra grande. Une pensée et mille pardons pour mon silence pour Anne-Laure et Claudie. Enfin n'oublions pas les lyricophiles avec qui j'ai partagé de grand moments de passion pour des braillardes dilatées.

A Marius et Lucien

Il n'en croyait pas ses yeux, ne concevait pas comment il avait pu se tromper ainsi. Il lui sembla en cet instant que la Dame de Pique le regardait d'un air moqueur.

Pouchkine

Résumé

Cette thèse s'intéresse d'une part au problème de décomposition technologique orienté surface sur des réseaux programmables de type FPGAs (Field Programmable Gate Arrays) et d'autre part à la synthèse des macro-générateurs sur ASICs et plus précisément de la synthèse des additionneurs.

La décomposition s'articule autour de deux axes essentiels: tout d'abord, il s'agit d'optimiser la taille de la représentation des fonctions booléennes. Les représentations de base choisies ici sont les ROBDDs (Reduced Ordered Binary Decision Diagrams) ainsi qu'une structure dérivée, les ITE (If Then Else). La deuxième étape concerne la décomposition proprement dite. Les technologies cibles sont ici des FPGAs à base de LUT-k (Look Up Table), en particulier les FPGAs XC5200 de Xilinx et Orca de AT&T.

Les deux méthodes de décomposition technologique orienté surface proposées permettent une décomposition hétérogène en prenant en compte non pas une seule configuration mais un ensemble de configurations possibles de la cellule cible. la première méthode est fondée sur un parcours descendant et optimisé du ROBDD. La seconde méthode s'appuie sur une modélisation en recouvrement d'hypergraphe du problème de décomposition technologique. Dans les deux méthodes, le coût exact en terme de surface finale du circuit est pris en compte à chaque étape de la décomposition.

L'étude menée dan la deuxieme partie de la thèse sur la macro-génération conduit dans un premier temps à l'exploration de l'espace des solutions possibles puis à l'optimisation d'une solution sélectionnée par un algorithme de dérivation discrète. L'utilisation d'un filtre permet la restriction de l'espace des solutions à explorer et d'autre part guide le processus de dérivation en éliminant les solutions trivialement médiocres. La combinaison des processus d'exploration et de dérivations permet la génération de macros dont les caractéristiques physiques sont les plus proches possibles de celles fixées par un utilisateur potentiel.

Ces méthodes ont été intégrées au sein d'un outil universitaire ASYL+ développé au laboratoire CSI.

Abstract

This thesis deals with the mapping problem targeting LUTk-based FPGAs and the synthesis of macrogenerators for ASICS.

Two methods are proposed to solve the heterogeneous area-oriented mapping problem from an ROBDD representation of the boolean functions. Both algorithms can take into account a set of configurations of a LUT-based FPGAs, and optimize the decomposition by keeping the mapping closer to the target technology than classical mappers on LUTs. This is done by taking into account specific costs for each configuration.

The first algorithm presented is based on the Roth&Karp decomposition. The first step of the procedure consists in unifying the outputs of the ROBDD for a global processing. Then the top-bottom mapping takes place by applying recursively the Roth&Karp decomposition . The main goal of the procedure is to take into account all the ROBDD and all the possible configurations of the FPGA. This leads to the computation of a shortest path problem in a 'decomposition graph' that is defined in this thesis.

The second method consists in modelling the mapping problem as an hypergraph covering problem. The classical approaches bypass the complexity of the problem by using an heuristic. In this approach, the size of the problem is reduced, next is applied an algorithm that remains exponential.

To solve the hypergraph covering problem, the model is turned into an ILP (Inter Linear Problem). After the relaxation of the variables, the so called 'Proximal decomposition algorithm' is applied to solve the ILP. This algorithm has been chosen because it is less sensitive to the constraints than algorithms such as the Simplex method.

The objective of study on the macrogeneration is to propose an innovative Solution Space Exploration method in terms of Speed/Area. In fact, in recent synthesis tools, emphasis is placed on obtaining temporary optimization synthesis constraints which are sophisticated and precise. It is therefore important to be able to propose macroblocks that correspond exactly to specific requirements. A specialized environment for the generation and filtering of efficient solutions is proposed. We proposed an innovative solution space exploration based on the filtering of intermediate solutions. From a given solution, we proposed a procedure of derivation based on Tabu approach. Our approach can be efficiently applied in constraint timing macrogeneration.

Introduction

L'explosion de la taille des circuits intégrés a suscité depuis quelques années le développement intensif d'outils informatiques destinés à la conception des circuits. Ces outils étaient tout d'abord destinés aux ASICs (circuit intégré pour une application) puis se sont intéressés aux réseaux programmables.

La place des réseaux programmable sur le marché est, depuis leur apparition, en constante augmentation. Les premiers réseaux programmables furent les PAL ou CPLD, dont la caractéristique principale est la forte granularité. Une seconde génération de réseaux programmables, à savoir les FPGAs (Field Programmables Gate Arrays), présentent le double avantage d'être de faible granularité et d'être intégralement reprogrammables. Les outils destinés à la synthèse logique se sont adaptés tant a l'évolution des technologies cibles qu'aux exigences des concepteurs. L'évolution des outils fut aussi marquée par les différents langages utilisés pour la description des circuits à synthétiser. Les langages dits de bas niveau (Palasm EQN) ont été remplacés par des langages de haut niveau tels que VHDL ou Verilog. La croissance de la taille des circuits à synthétiser a également amené les concepteurs a utiliser des "macro-blocks" ou parties de circuit pre-synthétisées, optimisées et pré-stockées.

Deux critères majeurs guident la conception des circuits et conditionnent la phase de décomposition technologique : La minimisation de la surface et la performance en terme de délai du circuit. Plus récemment, on s'est intéressé à la minimisation de la puissance dissipée par les circuits.

Cette thèse s'intéresse aux réseaux programmables de type FPGAs constitués de cellules de bases appelées LUT (Look Up Table) dont la particularité est de pouvoir implémenter n'importe quelle fonction de k variables. Le travail va porter plus précisément sur la minimisation de la surface du circuit au niveau de la décomposition technologique. Les FPGAs pouvant être configurés suivant un ensemble de modes prédéfinis, on s'attachera à utiliser le plus de modes possibles pour la minimisation. Dans une dernière partie, cette thèse s'est également interessée à la génération de macros, cette fois plus précisément des additionneurs dans le domaine des ASICs.

La thèse comprend les chapitres décrits ici:

• Le premier chapitre a pour but essentiel de décrire les FPGAs qui sont pris comme technologies cible de la décomposition technologique et d'en extraire tous les modes utilisables.

• Le second chapitre présente les notions fondamentales de l'algèbre de Boole ainsi que les techniques majeures utilisées au cours de la factorisation d'un ensemble de fonctions booléennes.

• Le troisième chapitre s'intéresse aux différents modes de représentation des fonction booléennes. La représentation sous forme de BDDs (Graphes de Décision Binaires) puis des ROBDDs (Graphes de Décision Binaires Ordonnés et Réduits) y est tout particulièrement développée car elle constitue la représentation sur laquelle sont fondées les techniques de décomposition technologique présentées plus loin. Dans ce chapitre est également menée l'étude de la taille de la représentation en ROBDD et sont décrites les méthodes permettant la réduction de cette taille qui conditionne le résultat de la décomposition technologique.

• Le quatrième chapitre s'intéresse à la décomposition technologique sur FPGAs. Les deux méthodes qui y sont proposées permettent des décompositions technologiques hétérogènes, c'est à dire une décomposition sur non pas une configuration de cellule mais sur un ensemble de configurations. La première méthode s'appuie sur la décomposition de Roth et Karp. Il y est décrit une modélisation de la décomposition technologique en problème de plus court chemin dans un graphe orienté. La seconde méthode s'appuie sur une modélisation en hypergraphe; le problème revient alors a chercher une couvrture de cet hypergraphe.

• Le cinquième chapitre présente une méthode puis un outil permettant d'une part une représentation simple et compacte d'additionneurs et d'autre part une optimisation des performances des additionneurs. Le travail porte ici sur l'exploration de l'espace des additionneurs et sur un processus de dérivation permettant des amélioration locales. La représentation des additionneur est réalisé à l'aide d'arbres de coupe qui en décrivent la structure et la parallélisation des calculs. Un ensemble d'opérations éléméntaires déstinées à modifier la structure des addiionneurs et l'utilisation d'un filtre permettent la dérivation et l'optimisation des performances des additionneurs.

I- Introduction aux Réseaux Programmables

Les circuits programmables remportent depuis leur mise sur le marché un très grand succès. Ils permettent de réduire la taille des cartes électroniques de façon considérable. Cette miniaturisation apporte des gains tant en volume, en poids et en consommation des circuits. Les circuits spécifiques (ASICs) présentent la capacité d'intégrer l'ensemble des fonctions d'une carte électronique dans un seul circuit. Mais ils sont fabriqués par un fondeur et dédiés à une application précise. Les inconvénients sont multiples: les coûts élevés de fabrication, les délais d'obtention, et surtout le risque technique lié à d'éventuelles erreurs tant au niveau du design que de la fabrication. Le circuit ne peut être testé qu'après sa fabrication et toute erreur se traduit par une nouvelle étude et une nouvelle construction du prototype.

Les circuits programmables ne présentent pas tous ces inconvénients. Leur coût est moindre, et le risque quasi nul. Les circuits ainsi conçus sont immédiatement fabriqués et testés. Des erreurs du design n'entraînent qu'une modification locale du circuit.

I-1 Les CPLDs (Complex Programmable Logic Devices)

Cette catégorie de réseaux programmables est caracterisée d'une part par sa forte granularité et d'autre part par sa structure composée de deux étages ET et OU. Le premier etage (nommé "étage produit") réalise le ET logique (.) entre les entrées pour former un ensemble de monômes. Le second (appelé "étage somme") réalise un OU logique (+) entre les monômes pour former des fonctions booléennes. Toute variable est présente en entrée sous sa forme normale a ou sous sa forme complementée $\overline{a}$. En sortie, nous obtenons des fonctions booléennes de la forme $f = \sum_i m_i$ où m_i est un monôme.

Cette implémentation nécessite une minimisation globale des fonctions booléennes si la structure autorise le partage de monômes.

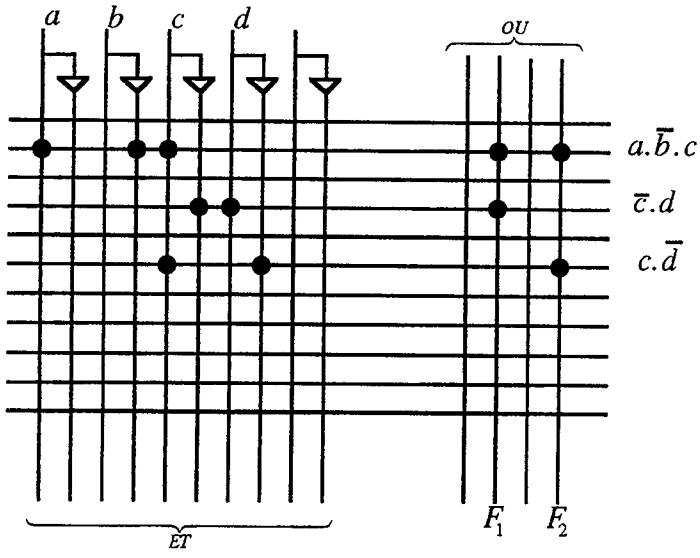

Figure I-1: Réseau à deux plans.

Le réseau à deux plans décrit figure I-1 permet de réaliser le couple de fonctions suivant:

$$\begin{cases} F_1 = a.\overline{b}.c + \overline{c}.d \\ F_2 = a.\overline{b}.c + c.\overline{d} \end{cases}$$

La programmation de ce type de réseau est réalisée à l'intersection des lignes verticales et horizontales de chaque plan. Dans le cas de réseaux non reprogrammables, on utilise des fusibles qui sont brulés. Dans le cas de réseaux reprogrammables, on utilise des cellules "EPROM" chargées pour assurer la connection, et dechargées pour rompre la connection.

La figure I-1 décrit un bloc qui autorise le partage de monômes ($a.\overline{b}.c$ etant partagé par les deux sorties). Certains boitiers (par exemple PAL22V10) n'autorisent pas ce partage de monôme et chaque sortie possède son ensemble de monômes propres.

I-2 LES FPGAs (Field Programmable Gate Arrays)

Cette famille est caracterisée par un ensemble de petites cellules assurant la partie logique et séquentielle réparties uniformément dans le boitier et par des canaux de connections permettant de relier les cellules entre elles. Ces technologies de faible granularité nécessite la décomposition des fonctions booléennes en un ensemble de sous-fonctions pouvant être contenues par une cellule. L'implémentation finale est dite "multi-couche" par opposition à l'implémentation à deux niveaux réalisée sur les CPLDs.

La famille des FPGAs peut être divisée en deux groupes:

- Les FPGAs à base de LUTs (Look Up Tables).
- Les FPGAs à base de multiplexeurs.

I-2-1 Les FPGAs à base de LUTs

Le LUT est une mémoire qui stocke le tableau de verité de n'importe quelle fonction de k variables. On parle alors de LUT-k. La fonctionnalité des cellules est déterminée par les LUT. Les FPGAs de Xilinx (XC3000, XC4000, XC5200), Orca de AT&T ainsi que Flex de ALTERA sont basés sur les LUTs.
Diverses contraintes peuvent apparaître au niveau de ces cellules, comme le partage des entrées (XC5200 de Xilinx) ou la restriction du nombre de sorties possibles. Ces contraintes amènent des traitements différents pour la synthèse des fonctions booléennes.

I-2-2 LES FPGAS à base de multiplexeurs

Les multiplexeurs sont des micro-cellules à trois entrées capables de réaliser la fonction:

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

En d'autre terme, le multiplexeur permet de choisir entre les deux entrées b et c en fonction de la valeur de a.

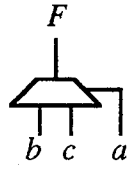

Figure I-2: Représentation
d'un multiplexeur.

La cellule à base de multiplexeurs (ou MUXs) est composé d'un arbre de MUXs dont la logique des entrées varie suivant le FPGA considéré (ACT1 et ACT2 de Actel par exemple [1]).

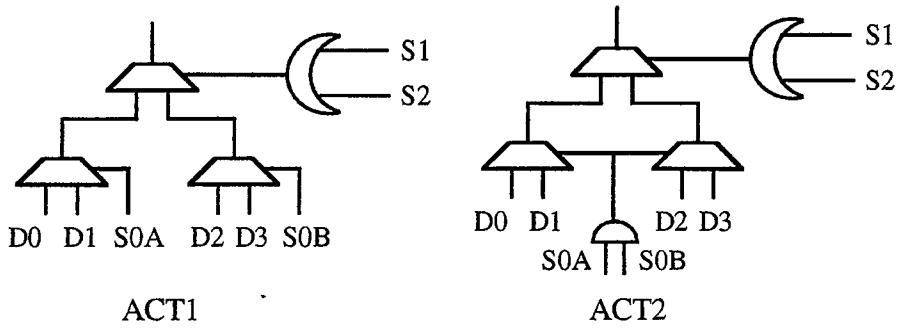

Figure I-4: Cellules à base de multiplexeurs.

I-3 Les FPGAs de Xilinx et de AT&T

On s'intéressera plus particulièrement à la cellule Xilinx 5200 et à la cellule Orca de AT&T. Nous ne nous intéresserons qu'à la partie logique des cellules. La cellule XC5200 de la famille Xilinx comme la cellule Orca sont toutes deux formées à base de LUTs de quatre entrées. Les différences entre ces cellules résident dans les contraintes au niveaux des entrées des LUTs, ainsi que dans le traitement des sorties de ces LUTs.

I-3-1 Xilinx 5200

Cette cellule comprend une série de quatre sous-cellules identiques et composées d'un LUT4 et d'un ensemble de trois multiplexeurs (figure I-2) [2]. Pour le mapping sur ROBDDs ou ITEs, on ne considèrera pas la totalité de la cellule, mais

on va simplement prendre en compte deux configurations possibles. Dans un premier cas, on peut considérer la cellule comme un ensemble de quatre LUT4. Dans le second cas, on va la considérer comme une union de deux LUT4 dont les sorties sont reliées par des multiplexeurs (Figure I-5).

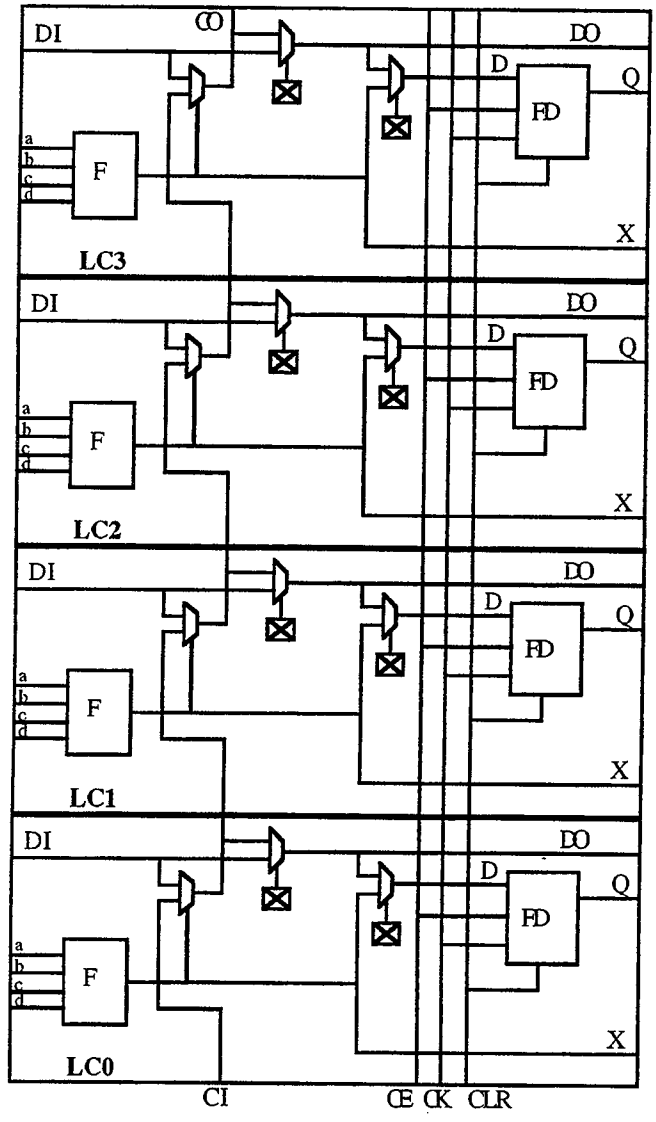

Figure I-4: Bloc Logique Configurable (CLB) de Xilinx 5200

Il est possible d'extraire d'autres configurations de cette cellule, notamment, on peut l'utiliser en tant qu'un couple de LUT5s, ou comme un LUT6. Ces configurations utilisent respectivement deux et quatre LUT4s. On se restreint a ces deux configurations car pour une décomposition technologique orientée surface, il est trivial que ces configurations sont moins intéressantes que les deux choisies. En effet la configuration LUT5 est incluse dans la configuration formée d'un couple de LUT4s reliés par un multiplexeur et occupe la même surface. La même remarque

s'applique aux LUT6s dont la surface est équivalente à celle de deux LUT5 ou quatre LUT4s.

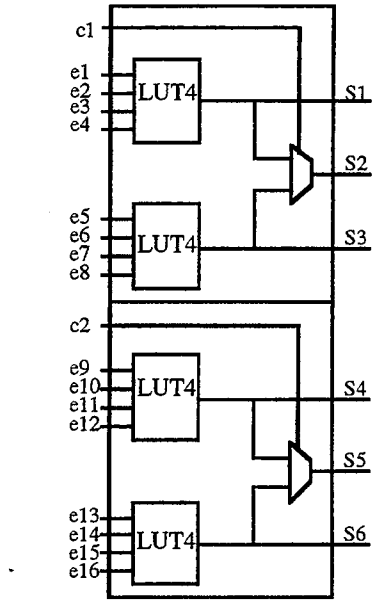

Figure 1-5: Configuration simplifiée d'un CLB XC5200

I-3-2 La cellule Orca de AT&T

Cette cellule [3] est composée de deux couples de LUT4s dont trois des entrées sont identiques. On a donc cinq entrées pour chaque couple de LUT4s. Grâce au "Program Controlled Logic" chaque cellule peut être configurée suivant un ensemble de modes dont nous sélectionnons un ensemble pour réaliser la décomposition technologique.

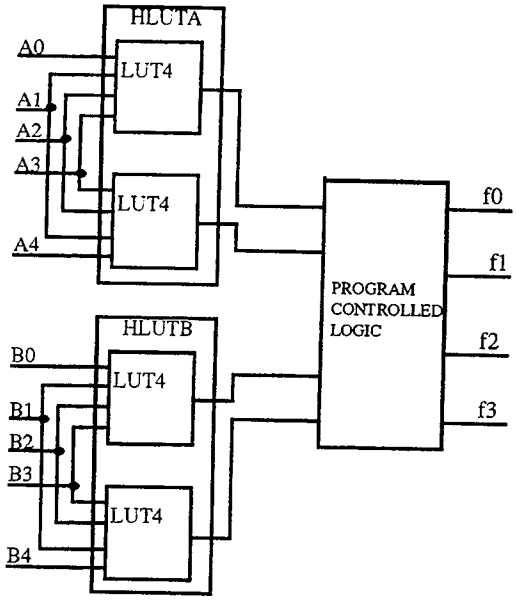

Figure I-6: Partie Combinatoire de la cellule AT&T Orca.

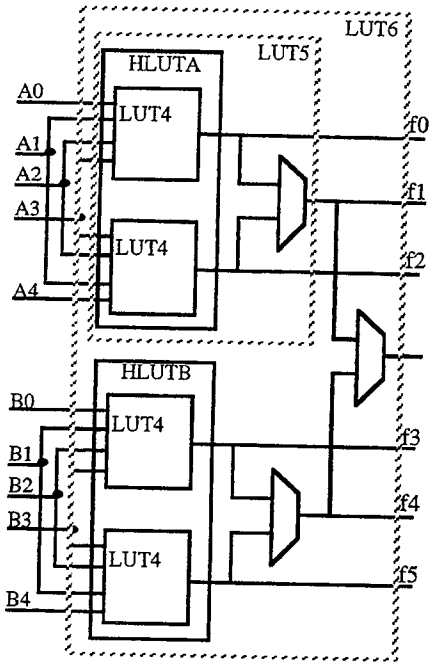

Figure I-7: Configuration utilisée

Comme dans le cas de la Cellule Xilinx 5200 (XC5200) on peut utiliser la cellule comme un ensemble de LUT4. Les contraintes liées aux entrées des LUTs rendent leur utilisation moins flexible; de plus, (cf. figure I-7) dans les deux couples de trois sorties (f0,f1,f2) et (f3,f4,f5) on ne peut utiliser que deux sorties parmi les trois. Celles-ci engendrent bien sur des restrictions au niveau des possibilités, mais nous verrons comment utiliser ces LUTs de façon à respecter ces contraintes. La cellule peut être utilisée également comme un couple de LUT5 ou un LUT6. Ces configurations seront ici prises en compte.

I-4 Le flot de conception sur FPGAs

Le flot de conception sur FPGAs, présenté sur la figure I-8 prend en entrée une description de haut ou de bas niveau, pour aboutir à une description du circuit, prêt alors à être physiquement réalisé. La saisie est réalisée sur une bibliothèque virtuelle constituée de portes logiques et séquentielles simples. Cette traduction servira de base à la synthèse. La phase de placement et routage a pour but d'affecter aux cellules issues de la synthèse des positions physiques dans le circuit, puis de déterminer les ressources qui interviennent dans le routage de chaque signal du

circuit. Le routage de la totalité du circuit n'est pas garanti car les ressources de connections sont limitées.

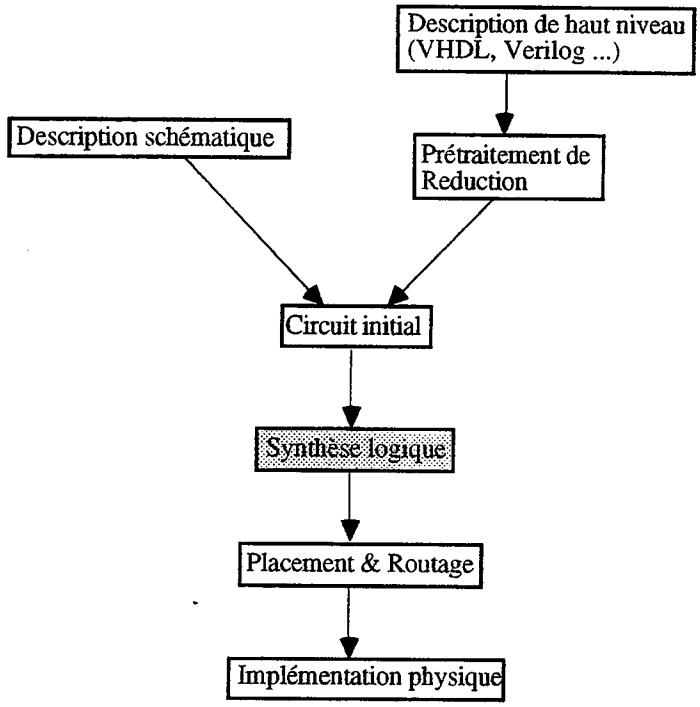

Figure I-8: Flot de conception.

II- Algèbre de Boole

II-1 Généralités

Le mathématicien anglais Georges Boole introduisit cette algèbre afin de formaliser le raisonnement portant sur les valeurs logiques vrai faux et les conjonctions non, et, ou.

- **Définition 1: Algèbre de Boole**

L'ensemble $B = \{0,1\}$muni de l'opération unaire non, et des deux opérations binaires et, ou est une algèbre appelée algèbre de Boole.

- **Définition 2: Variable booléenne**

Soit $B = \{0,1\}$ l'espace de Boole; une variable booléenne générale est un n-uplet $(a_1, a_2, ..., a_n)$ de B^n. Une Variable booléenne simple représente une coordonnée dans B^n. Une variable générale peut prendre 2^n valeurs .

- **Définition 3: Littéral**

Variable booléenne simple dans sa forme directe ou complementée (exemple a ou $\overline{a}$)

- **Définition 4: Fonction booléennes**

Une fonction booléenne est une application de l'ensemble B^n dans l'ensemble B^m.

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

Exemple:

La fonction $F(a,b) = a + \overline{b}$ est une fonction booléenne simple d'une variable générale.

- **Définition 5**

Fonction booléenne incomplète ou (φ-booléenne): une fonction booléenne est dite incomplète si elle est définie par:
$F: B^n \mapsto Y^m$, avec $Y = \{0, 1, \varphi\}$
φ représente la valeur indéfinie (ou Don't care), 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:

- $C_\infty(F)$ représente la couverture à 1 d'une fonction, c'est à dire l'ensemble des points 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 points pour lesquels la valeur de la fonction est 0.
- $C_\varphi(F)$ représente la couverture à φ d'une fonction, c'est à dire l'ensemble des points 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) et correspond à $C_\infty(F)$.
La borne inférieure de F est obtenue en remplaçant tous les φ par des 0. Elle est notée INF(F) et correspond à $C_0(F)$.

- **Définition 6: Monôme**

Un monôme (ou cube) est un produit de n variables simples distinctes sous la forme normale (a) ou complementée ($\overline{a}$); n est appelé le degré ou la longueur du monôme. La taille du monôme est le nombre de points qu'il couvre. Un monôme couvre un point si la valeur de ce monôme en ce point est égale à 1.

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

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

On dit qu'un monôme m1 est inférieur à un monôme m2 (noté $m1 \prec m2$), si tous les points couverts par m1 sont couverts par m2. On dit aussi que m1 est un multiple de m2

Exemple:

$m1 = a.\overline{b}.\overline{c}.d, m2 = a.\overline{c} \Rightarrow m1 \prec m2.$

- **Définition 8: Expression booléenne**

Une expression booléenne est construite à partir des variables booléennes simples de la fonction et de l'ensemble des opérateurs logiques:

L'union (ou +)

L'intersection (et .)

La négation (non -)

- **Définition 9: Monômes premiers**

Un monôme m est dit premier dans une fonction booléenne F si et seulement si

$\begin{cases} m \leq f \\ \text{il n'existe pas } m' \neq m \,/\, m' \leq f \text{ et } m \leq m' \end{cases}$

Exemple:

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

$\overline{a}.\overline{c}.\overline{d}$ est un monôme premier de F tandis que $a.c.\overline{d}$ n'est pas premier car $a.c.\overline{d} \leq a.c$

- **Définition 10: Base d'une fonction booléenne**

Une base d'une fonction est une somme de monômes premiers égale à la fonction. On appelle base complète la somme de tous les monômes premiers. Une base est dite irrédondante si elle cesse d'être une base quand on lui enléve un de ses monômes.

Exemple:

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

$\overline{a}.\overline{c}.\overline{d} + \overline{a}.b.\overline{c} + b.\overline{c}.d + a.c$ est une base de F (non irrédondante).

$\overline{a}.\overline{c}.\overline{d} + b.\overline{c}.d + a.c$ est une base irrédondante de F.

- **Définition 11: Expression booléenne polynomiale**

Une expression booléenne 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:
$F = a.b + \overline{c}.d + g$ est une expression booléenne polynomiale.

- **Définition 12: 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: $F \leq G \Leftrightarrow \begin{cases} F.G = F \\ \quad et \\ F + G = G \end{cases}$

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

Exemple:
$F = a.b.c$
$G = a.b$
$\Rightarrow F \leq G$

- **Définition 13: Support d'une expression**

Le support d'une expression d'une fonction booléenne F noté $SUPP(F)$ est définie par:
$SUPP(F) = \{a / \exists m \in M_F / a(\overline{a}) \in m\}$

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

- **Définition 14: Produit algébrique et booléen**

Soient $F = \sum_{i=1}^{n} m_i$ et $G = \sum_{j=1}^{p} m_j$ deux expressions polynomiales de deux fonctions booléennes. Le produit algébrique $F.G$ existe si F et G dépendent de deux ensembles de variables disjoints; il est défini par:
$F.G = \sum_{i=1..n, j=1..p} m_i.m_j$

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 ensembles 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:

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

$G = a.e$

$\Rightarrow$

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

- **Définition 15: 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 $H' / H < H'$ et $F = D.H' + R$, R étant une expression polynomiale.

Le quotient H et le reste R sont uniques.

Exemple:

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

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

$F / D = a.b$ 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 solution contenant la solution algébrique. Le quotient et le reste booléens ne sont pas uniques.

Exemple:

$F = a.d + c + d$

$D = a + c$ est un diviseur booléen de F, car F peut être écrit sous la forme :

$F = D.(a + b) + d$

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

- **Définition 16: Facteur algébrique et booléen**

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

G est un facteur booléen de F si $F = G.H$, tel que H est une expression booléenne et $G.H$ le produit booléen de G par H.

- **Définition 17: 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 dans 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:

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

$G = a + b + e$

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

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

- **Définition 18: Expression libre**

Une expression booléenne polynomiale F est libre s'il n'existe pas de monômes m (avec $m \neq 1$) qui soit un facteur algébrique de F.

Exemple:

$a.b + a.c$ 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.

Un monôme n'est pas une expression libre.

- **Définition 19: Noyaux algébriques**

K est un noyau algébrique d'une expression polynomiale F si K est le quotient algébrique de F par C, où C est un monôme et K 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 de degré 0 s'il n'accepte comme noyau que lui-même.
- Un noyau K est de degré n s'il y 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 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:

$K(F) = \{K_1(F), K_2(F), \ldots, K_{n-1}(F), K_n(F)\}$

Remarque:

- 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.
- 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 $a.b$ pour F_1 et a pour F_2; inversement, le noyau $b.c + b.d + e.f$ est un noyau local de degré 1, son conoyau est a pour F_2.

- **Définition 20: Portée d'un noyau**

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

K est un noyau de l'expression F,

C est le conoyau associé au noyau K,

E est l'ensemble des monômes résultant 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 F,

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

$\mathrm{E} = a.b.c + a.b.d$ est l'ensemble des monômes de $C.K$ écrit sous forme polynomiale.
(C,K,E) définit la portée de K dans F.

- **Définition 21: Compatibilité algébrique**

Soient deux couples de conoyaux/noyaux (C,K) et (C',K') d'une expression booléenne F. Soit E (resp. E') l'ensemble des monômes de $C.K$ (resp. $C'.K'$) écrit sous forme polynomiale.
On dit que (C,K) et (C',K') sont compatibles algébriquement si et seulement si:

$\mathrm{E} \subset \mathrm{E}'$ ou $\mathrm{E}' \subset \mathrm{E}$ ou $\mathrm{E} \cap \mathrm{E}' = \varnothing$

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',
$\mathrm{E} = \{a.b.c, a.b.d\}$
$\mathrm{E}' = \{a.b.d, a.c.d\}$
(C,K) et (C',K') sont incompatibles algébriquement.

II-2 La factorisation algébrique.

La factorisation est un processus qui part d'un ensemble de fonctions booléennes exprimées par des sommes de monômes (description 2-couches) et le transforme en un ensemble d'expressions factorisées. La complexité d'un circuit est étroitement liée au nombre de littéraux des formes factorisées représentant l'ensemble des fonctions booléennes. La factorisation a pour but de minimiser la complexité des formes factorisées et par conséquent la logique qui sert à les réaliser.

On peut associer de façon virtuelle un schéma de portes et/ou à des expressions en sommes de monômes et des expression factorisées.

La première implémentation s'appelle une implémentation 2-couches. 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 fonctions ou.

Une implémentation plus générale utilise plusieurs couches de logique. Elle repose sur des expressions booléennes factorisées. Cette forme factorisée est ensuite décomposée en sous-fonctions correspondant à des éléments de bibliothèque appelés "Cellules standards".

Exemple:
Soient les fonctions F_1 et F_2:
$F_1 = a.c + a.d.e + b.c + b.d.e$
$F_2 = a.f.g + b.f.g + e.f.g$

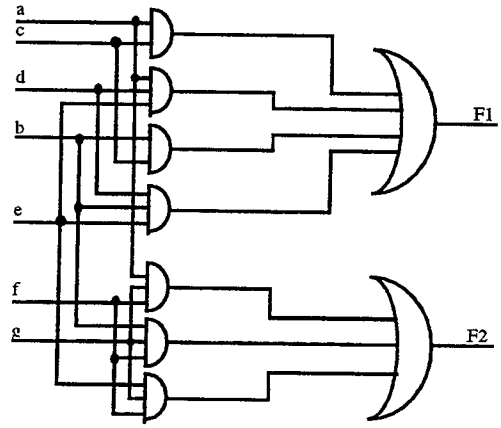

Figure II-1: Implémentation 2-couches

Après factorisation, l'expression des deux fonctions booléennes F_1 et F_2 devient:
$SF = a + b$
$F_1 = SF.(c + d.e)$
$F_2 = f.g.(SF + e)$

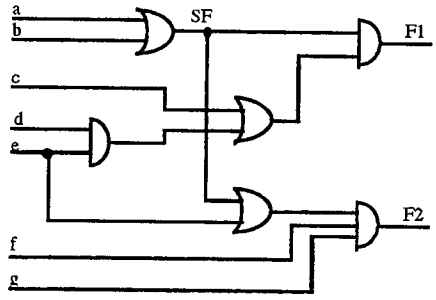

Figure II-2: Implémentation multi-couches

II-2-1 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 candidats peuvent appartenir à une seule fonction booléenne ou à plusieurs fonctions. Cela permet la diminution de la complexité des expressions, exprimées en terme d'occurrences de littéraux.

Il existe deux factorisations majeures:

- La factorisation algébrique qui utilise des noyaux algébriques.
- La factorisation booléenne qui utilise des noyaux (ou facteurs) booléens.

L'exemple suivant illustre la différence entre les deux types de noyaux. Soient les deux fonctions booléennes:

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

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

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

Les noyaux algébriques sont:

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

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

L'expression des deux fonctions booléennes F_1 et F_2 devient alors:

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

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

Soit K le noyau booléen :

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

L'expression K étant commune aux deux fonctions F_1 et F_2 , elle devient une sous-fonction commune notée SF. On peut donc exprimer les fonctions F_1 et F_2 de la forme:

$F_1 = c.d.SF$

$F_2 = b.f.SF$

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

Le nombre de littéraux dans le cas de la factorisation algébrique est de 12; tandis que dans le cas de la factorisation booléenne, on obtient un total de 11 littéraux. 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 avant la factorisation, moins le nombre de littéraux après la factorisation par ce noyau.

Une approche gloutonne (ou greedy) de la factorisation consiste à réaliser l'algorithme suivant:

```
Generation des candidats diviseurs
tant qu' il existe des candidats diviseurs
    Selection des candidats de gain maximal
    Division par ces candidats
Fin tant que
```

Le problème central revient ici à trouver un sous-ensemble de candidats diviseurs algébriquement compatibles 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 sélection gloutonne est utilisée.

II-2-2 Génération des candidats diviseurs

Nous allons nous intéresser aux deux types de candidats suivants:

- Noyaux algébriques et leur intersection.

- Monômes communs ou parties de monômes communes pour un ensemble de fonctions.

En fait, la recherche de monômes ou de parties de monômes communs se ramène à une recherche de noyaux, les monômes communs apparaîtront comme les conoyaux associés à ces noyaux .

Recherches de monômes ou parties de monômes communs par la recherche de noyaux.

Soit un ensemble de fonctions $\{F_1, F_2, ..., F_n\}$ dont on va chercher les monômes communs. On utilise pour ce faire l'algorithme suivant:

Posons: $F = \sum_{i=1}^{n} F_i . v_i$

Calcul des conoyaux / noyaux associés à F

Soit C un conoyau

Si $\forall i \; v_i \notin C$ alors C est un monôme commun

Exemple:

$$F_1 = a.b.c.d + d.e + h$$
$$F_2 = a.b.c.e + d.e + h$$
$$\Rightarrow$$
$$F = a.b.c.d.v_1 + d.e.v_1 + h.v_1 + a.b.c.e.v_2 + d.e.v_2 + h.v_2$$

On obtient donc l'ensemble des couples conoyaux/noyaux suivant:

$$\{(a.b.c, d.v_1 + e.v_2), (d.e, v_1 + v_2), (h, v_1 + v_2)\}$$

Donc $a.b.c$ 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 signifie que $d.e + h$ est une somme de monômes communs.

L'algorithme proposé pour la génération des noyaux est celui présenté dans [4] . Cet algorithme est fondé sur l'ordonnancement des littéraux d'une fonction booléenne, puis sur la mise en facteur des littéraux un par un, afin de trouver les différents couples conoyaux/noyaux.

Noyaux(F)

$m \leftarrow$ le monôme de plus grand degré, facteur algébrique de F

$List\_Noyaux \leftarrow \text{RNoyaux}\left(0, F/m\right)$

si $m = 1$, alors $List\_Noyaux \leftarrow List\_Noyaux + F$

RNoyaux(j, F)

$List\_Noyaux \leftarrow \{F\}$

pour $i = j + 1$ jusqu'à n faire

si l_i apparait dans plus d'un monôme de F alors

$m \leftarrow$ le monôme de plus grand degré, facteur algèbrique de F/l_i

si $l_k \notin m, \forall k \leq i$ alors $List\_Noyaux \leftarrow List\_Noyaux + \text{RNoyaux}\left(i, F/l_i.m\right)$

Retourner($List\_Noyaux$)

Explication de l'algorithme

Dans cet algorithme de génération de l'ensemble des noyaux algébriques, on commence par ordonnancer les littéraux de la fonction booléenne $\{l_1, l_2, ..., l_n\}$. Ensuite, on met en facteur le plus grand monôme m facteur algébrique de la fonction F, et on appelle enfin la procédure $RNoyaux\left(0, F/m\right)$.

Dans la procédure $RNoyaux(j, F)$, on divise l'expression F par tous les littéraux l_k pour $j < k \leq n$. Pour chaque division par l_k, et par le facteur algébrique m de $F/\left(l_{j+1}.l_{j+2}...l_k\right)$,on appelle récursivement $RNoyaux(k, F/\left(l_{j+1}.l_{j+2}...l_k.m\right)$ pour extraire tous les noyaux de $F/\left(l_{j+1}.l_{j+2}...l_k.m\right)$. 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.

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 produit par l'algorithme est le suivant:

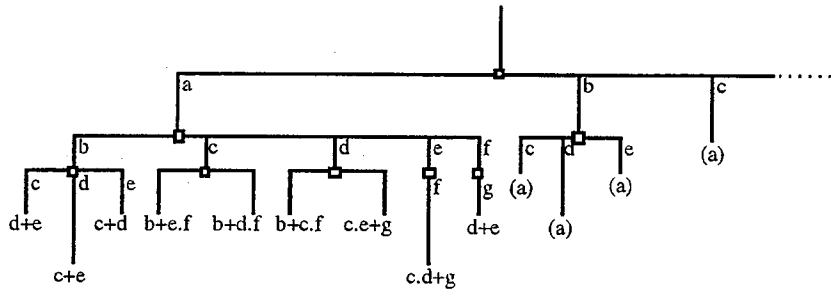

Figure II-3: Représentation des noyaux générés par l'algorithme

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. A chaque niveau L de l'arbre, nous avons les noyaux de degré $D-L$, où D est la profondeur de l'arbre. Le lettres affectées aux branches correspondent aux littéraux éliminés de l'expression à chaque étape. On remarque que ces littéraux sont ordonnés de la même façon en largeur et en profondeur dans l'arbre. (a) dans la branche (bac) signifie qu'on est ramené à la branche initiée par a (première en partant de la gauche) pour laquelle tous les noyaux ont été générés et par suite ceux de (a.b.c).

II-2-3 Calcul du gain d'un candidat diviseur

Calcul du gain en littéraux pour les noyaux locaux.

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 noyaux K.

- **Définition 22: Gain pour les noyaux locaux**

Le noyau apparaît dans une seule fonction booléenne, le gain est donc:

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

où $C_i, 1 \le i \le m$ sont les conoyaux associés au noyau K.

- **Définition 23: Gain pour les noyaux globaux**

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

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

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

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

Exemple:
$F_1 = a.b.c + a.b.d + a.e.f + g$
$F_2 = a.c + a.d + g.h$
Nous obtenons:
$K_1 = c + d$: noyau global; conoyaux associés: $a.b$ pour F_1 et a pour F_2
$K_2 = b.c + b.d + e.f$:noyau local; conoyau associé:a pour F_1
Gains associés:
$Gain(K_0) = ((2-1).2 + (2-1).1) + 1.(2) - 2 = 3$
$Gain(K_1) = (3-1).1 = 2$

- **Définition 24: Gain des monômes ou parties de monômes communs:**

Soient $NbLit(m)$ le nombre de littéraux du monôme m, et $NbOcc(m)$ le nombre d'occurrences du monôme m. La formule du gain associé à un monôme est la suivante:
$Gain(m) = (NbOcc(m) - 1).(NbLit(m) - 1) - 1$

Exemple:
$F_1 = a.b.c.d + d.e + h$
$F_2 = a.b.c.e + d.e + h$
$m_1 = a.b.c$ est une partie commune de monôme dont le gain associé est 1
$m_2 = d.e$ est un monôme commun dont le gain est 0

II-2-4 Sélection des candidats diviseurs

Cette étape permet de choisir dans l'ensemble des candidats diviseurs, déterminés lors de la phase de génération, celui qui impliquera un gain maximal de logique, exprimé en nombre de litteraux. Mais ce choix doit tenir compte de l'incompatibilité algébrique des noyaux.

Une première méthode considère globalement l'ensemble des candidats diviseurs pour tenir compte de l'impact créé par le choix de l'un d'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 suivant:

On construit le graphe non orienté $G(X,E)$ tel que:

- chaque sommet x_i de X correspond à un diviseur d_i;
- $e=(x_i,x_j)\in E$ si et sulement si les candidats diviseurs correspondant aux sommets x_i et x_j sont incompatibles.

Trouver l'ensemble de gain maximal de candidats diviseurs revient exactement à résoudre le problème de la recherche du stable de poids maximum dans $G(X,E)$. Ce problème est NP-complet, et donc la recherche d'un ensemble de candidats diviseurs compatibles deux à deux ne peut être résolue exactement.

Une heuristique, appelée principe de la couverture rectangulaire et proposée dans [5] est fondée sur la couverture disjointe des cellules d'une matrice. Cette méthode consiste à construire une matrice $G(X,E)$ 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, chaque cellule de la matrice représente un monôme de la fonction.

Bien que cette formulation paraisse intéressante, il s'avère que la complexité actuelle des circuits ne permet pas toujours de l'appliquer de manière optimale. Par exemple, pour une expression polynomiale qui comporte 100 monômes de 10 variables, on peut générer 1000.000 de rectangle. 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/résultat.

La seconde méthode, un algorithme glouton, sélectionne le candidat diviseur permettant d'obtenir un gain local maximal et élimine de la liste des candidats ceux qui ne sont plus compatibles après le choix de ce dernier. Cette mise à jour permet de garder uniquement les candidats diviseurs compatibles avec ceux déjà choisis.

III- Représentation des fonctions booléennes

Nous nous attacherons ici aux modes de représentation canonique des fonctions booléennes. La canonicité d'une représentation permet de comparer l'équivalence de deux fonctions ainsi que la tautologie par simple comparaison syntaxique des représentations. Jusqu'à l'apparition des arbres de décision binaires, on ne connaissait que peu de formes canoniques de la logique booléenne:

- Les tables de vérité et tableaux de Karnaugh.
- La forme canonique de Blake [6] .
- La somme exclusive de produits, ou forme de Reed-Müller [7] [8].

Ces représentation présentent le défaut d'être très peu compactes dans le sens où le nombre de caractères nécessaires pour écrire la forme canonique d'une équation F peut être très grand par rapport au nombre de caractères de F (voire exponentiel). Récemment, de nouvelles formes canoniques ont été définies, les BDDs [9] (arbres de décision binaires), les OBDDs [10], les ROBDDs (typés ou non typés) [11] [12] [13].

Nous présenterons toutes ces formes de représentation et nous comparerons leurs complexités de constructions, ainsi que les avantages et inconvénients qu'elles présentent au niveau de la synthèse et de leurs diverses applications.

III-1 Représentations classiques

Les systèmes de représentation les plus primaires reposent sur les tables de vérité et les tableaux de Karnaugh qui présentent le gros avantage de la canonicité. Néanmoins, ces représentations nécessitent 2^n unités mémoires pour une fonction de n variables. De fait, pour les expressions booléennes de grande taille, ces méthodes se sont avérées inefficaces et ingérables lors de la manipulation des fonctions. On a donc besoin de représentations non exponentielles pour la plupart des fonctions. C'est le cas de la forme de Reed-Müller (considérée comme très puissante) et de la somme réduite de produits (appelée aussi forme disjonctive réduite) que nous décrivons maintenant.

III-1-1 Somme réduite de produits

Cette représentation consiste à exprimer une fonction en terme de somme de produits et de la réduire selon un ensemble de règles. La première étape consiste à développer l'expression booléenne en somme de produits par le système de réécriture suivant:

$F \oplus G = \overline{F}.G + F.\overline{G}$

$F \Leftrightarrow G = F.G + \overline{F}.\overline{G}$

$\overline{F+G} = \overline{F}.\overline{G}$

$\overline{F.G} = \overline{F} + \overline{G}$

$F.(G+H) = F.G + F.H$

$\overline{\overline{F}} = F$

Dans un second temps, on réduit l'expression en appliquant les règles de simplification suivantes:

$a.\overline{a} = 0$

$a + \overline{a} = 1$

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

$a + a.b = a$

La taille d'une telle représentation pour une fonction quelconque sera le nombre d'occurrences de littéraux dans cette fonction. Notons que cette forme n'est pas canonique.

Exemple:

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

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

ces deux représentations sont équivalentes d'un point de vue logique mais syntaxiquement différentes.

Cette représentation trouve son intérêt dans les méthodes de minimisation en vue d'une implementation sur PLA (#). Elle est encore utilisée dans de très nombreux algorithmes de synthèse (factorisation algébrique et booléenne, minimisation 2-couches ...)

III-1-2 Forme de Reed-Müller

Cette représentation consiste à exprimer une fonction booléenne en terme de somme exclusive de produits, puis de la réduire. Comme pour la somme de produits, on a un système de réécriture qui est le suivant:

$F.(G \oplus H) = F.G \oplus F.H$

$F + G = F \oplus G \oplus F.G$

$F \Leftrightarrow G = 1 \oplus F \oplus G$

$\overline{F} = 1 \oplus F$

Puis les simplifications suivantes vont rendre cette forme canonique:

$0.F = 0$

$1.F = F$

$F \oplus F = 0$

$0 \oplus F = F$

Une somme exclusive de produits ou ESOP (Exclusive Summ Of Products) s'écrit:

$F = B \oplus \left(\bigoplus_{i=1}^{n} \left(\prod_{j=1}^{m_i} x_i^j \right) \right)$, avec $B \in \{0,1\}$ et x_i^j représente une variable

Exemple:

Pour une fonction: $F = \overline{a}.\overline{b}.\overline{c}$

on a la représentation: $F = 1 \oplus a \oplus b \oplus c \oplus a.b \oplus b.c \oplus a.c \oplus a.b.c$.

III-2 Graphes de décision binaire

III-2-1 Décomposition de Shannon

Etant donnée une fonction booléenne $F(x_1, x_2, ..., x_i, ..., x_n)$, les cofacteurs positifs et négatifs de F selon la variable x_i sont définis par:

$$F_{x_i} = F(x_1, ..., x_{i-1}, 1, x_{i+1}, ..., x_n)$$

$$F_{\bar{x}_i} = F(x_1, ..., x_{i-1}, 0, x_{i+1}, ..., x_n)$$

La décomposition de Shannon selon la variable x_i devient:

$$F = x_i . F_{x_i} + \bar{x}_i . F_{\bar{x}_i}$$

Théorème 1:

Soit F une fonction booléenne; il existe un couple unique (F_0, F_1) de fonctions booléennes tel que:

$$F(x_1, ..., x_i, ..., x_n) = \bar{x}_i . F_0(x_1, ..., x_{i-1}, x_{i+1}, ..., x_n) + x_i . F_1(x_1, ..., x_{i-1}, x_{i+1}, ..., x_n)$$

Démonstration:

Supposons qu'il existe un second couple (F_0', F_1') possédant la même propriété.

$\forall x_1, ..., x_{i-1}, x_{i+1}, ..., x_n$

on a

$$F(x_1, ..., x_{i-1}, 0, x_{i+1}, ..., x_n) = \bar{0} . F_0(x_1, ..., x_{i-1} x_{i+1}, ..., x_n) + 0 . F_1(x_1, ..., x_{i-1} x_{i+1}, ..., x_n)$$

$$\Rightarrow F(x_1, ..., x_{i-1}, 0, x_{i+1}, ..., x_n) = F_0(x_1, ..., x_{i-1} x_{i+1}, ..., x_n)$$

$$F(x_1, ..., x_{i-1}, 0, x_{i+1}, ..., x_n) = \bar{0} . F_0'(x_1, ..., x_{i-1} x_{i+1}, ..., x_n) + 0 . F_1'(x_1, ..., x_{i-1} x_{i+1}, ..., x_n)$$

$$\Rightarrow F(x_1, ..., x_{i-1}, 0, x_{i+1}, ..., x_n) = F_0'(x_1, ..., x_{i-1} x_{i+1}, ..., x_n)$$

donc

$$F_0(x_1, ..., x_{i-1} x_{i+1}, ..., x_n) = F_0'(x_1, ..., x_{i-1} x_{i+1}, ..., x_n)$$

La même démonstration peut être menée pour prouver que:

$$F_1' = F_1$$

d'où l'unicité du couple (F_0, F_1).

III-2-2 Arbre de Shannon - BDD

Considérons une fonction:

$$F: \{0,1\}^n \mapsto \{0,1\}$$

La décomposition de Shannon par rapport à une variable x_i induit un couple unique de fonctions noté (F_0, F_1):

$$F_0: \{0,1\}^{n-1} \mapsto \{0,1\}$$

$$F_1: \{0,1\}^{n-1} \mapsto \{0,1\}$$

telles que:

$$F(x_1,\ldots,x_i,\ldots,x_n) = \overline{x}_i.F_0(x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_n) + x_i.F_1(x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_n)$$

Si l'on réitère cette décomposition sur les sous-fonctions F_0 et F_1 on obtient un arbre binaire, appelé arbre de Shannon, dont les feuilles sont les constantes 0 et 1.

Le théorème 1 est ici important car il assure la canonicité de la décomposition de Shannon puisque celle-ci est unique pour une fonction booléenne F et une variable x_i donnée. Donc, si un ordre de décomposition est fixé pour chaque chemin de l'arbre de décomposition, alors toute fonction G, fonctionnellement équivalente à F aura le même arbre de Shannon pour les mêmes séquences de décomposition.

- **Définition 25: Graphe de décision binaire**

  Un graphe de décision binaire (ou Binary Decision Diagramm) est un graphe acyclique représentant une fonction booléenne. La construction du BDD est fondée sur la décomposition successive de Shannon selon une séquence de variables donnée.

Le BDD est de taille $2^n - 1$, donc exponentielle en fonction du nombre de variables de la fonction F. Nous verrons des méthodes permettant de réduire cette taille de la représentation.

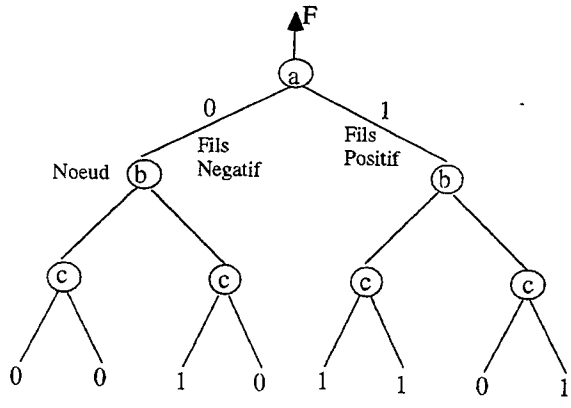

Figure III-1: BDD correspondant à la fonction $F(a,b,c) = a.\overline{b} + a.c + \overline{a}.b.\overline{c}$.

Chaque noeud N non terminal correspond à une variable de décomposition (ou variable de décision) x_i et est à l'origine de deux arcs:

- *Négatif*(N), le fils, ou l'arc pour x_i valant 0.

- *Positif*(N), le fils positif, ou l'arc pour x_i valant 1.

Le graphe au niveau de *Négatif*(N) représente le cofacteur $F_{\bar{x}_i}$, et le graphe au niveau de *Positif*(N) représente le cofacteur positif de F_{x_i}.

III-2-3 RBDD (ou BDD Réduit)

La taille du BDD étant exponentielle en fonction du nombre de variables de la fonction, on va tenter de réduire cette taille. Pour ce faire, on utilisera deux règles de réduction:

- Suppression de noeuds inutiles
- Mise en commun de sous-arbres identiques.

La première règle consiste à supprimer les noeud dont les deux fils sont égaux. En effet, dans ce cas, la variable de décision devient stérile. La seconde consiste à ne pas réécrire un sous-arbre existant, mais à réutiliser le sous-arbre déjà défini. On obtient alors un graphe acyclique orienté (cf figure III-2). On remarquera que ces deux transformations conservent la canonicité de la représentation.

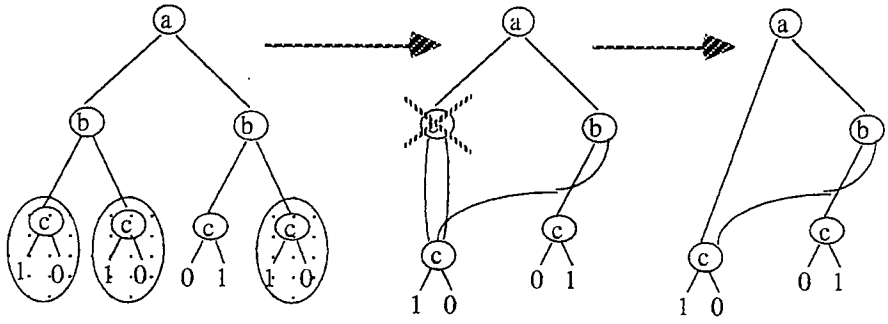

figure III-2 : application des règles de réduction.

III-2-4 ROBDD (BDD réduits et ordonnés)

Un ROBDD est un BDD réduit où un ordre total des variables de décomposition est imposé sur tous les chemins du graphe acyclique orienté; de fait, sur tout chemin de la racine jusqu'aux noeuds, les variables doivent apparaître dans le même ordre. Cet ordre total peut a priori apparaître comme très contraignant, mais il a l'avantage de générer en général un ensemble important de sous-arbres identique, permettant ainsi la mise en commun de sous-BDD.

Etant donné un ordre total des variables, le graphe est unique (au sens isomorphisme de graphes) et constitue une représentation canonique de la fonction.

III-2-5 ROBDD typés

- **Définition 26: Type**

  Un BDD est dit typé si ses arcs peuvent pointer de façon directe ou indirecte sur leurs fils. Autrement dit est introduit l'opération unaire non au niveau des arcs du graphes.

L'intérêt du typage est double:

- Il simplifie l'opération de négation qui ne prend plus qu'un temps constant $O(1)$ sur un ROBDD typé.

- Les sous-arbres existant de façon directes ou inversés peuvent être regroupés en un unique sous-arbre. Cette propriété permet une plus grande réduction de la représentation.

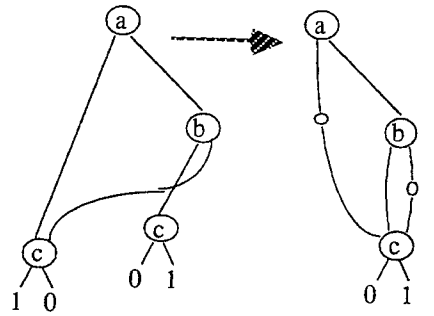

Figure III-3: Introduction du typage dans le ROBDD

L'introduction du typage détruit la canonicité de la représentation. Mais nous verrons plus loin comment nous y ramener.

Opération sur les arcs typés:

Quels que soient le type de l'arc arrivant sur le noeud ainsi que les types des deux fils (fils négatif et fils positif), la fonctionnalité est conservée si l'on inverse les types des trois arcs.

Pour maintenir une forme canonique de la représentation, on peut introduire la règle:

l'arc positif (resp. négatif) d'un noeud doit être de type direct.

Quelle que soit la configuration, il est possible de satisfaire cette contrainte en appliquant sur le ROBDD les règles décrites ci-dessous.

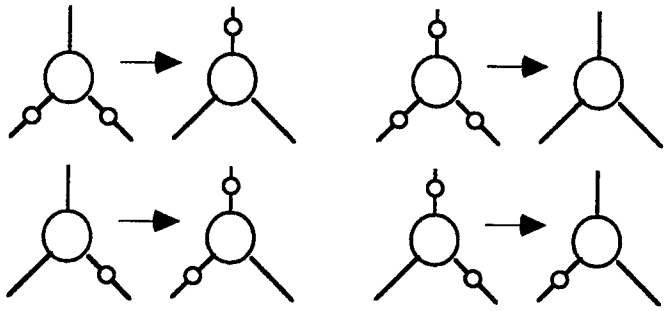

Figure III-4: Règles de transformation

La canonicité est ainsi conservée.

III-2-6 Autres formes de ROBDDs

III-2-6-1 Zero-suppressed BDDs

Lorsque l'on veut représenter un ensemble de combinaisons prises dans un ensemble de n objets, l'une des façons les plus compactes consiste à construire son BDD. La figure III-5 montre la représentation BDD sur un ensemble de 3 puis de 4 variables d'un ensemble de combinaisons.
L'ajout de la variable "a" donne un nouveau noeud sans apporter de nouvelles informations sinon dire que la variable "a" n'est pas dans les combinaisons. Dans un ensemble de combinaisons, les variables faisant défaut sont représentées par un 0 lorsque leur fonction caractéristique est vraie. Ces variables ne peuvent malheureusement pas être supprimées dans la représentation BDD. Pour éviter ces rajouts de noeuds inutiles, [14] a proposé une règle d'élimination de ces noeuds qui engendre un nouveau type de BDDs appelés "Zero-suppressed BDD".

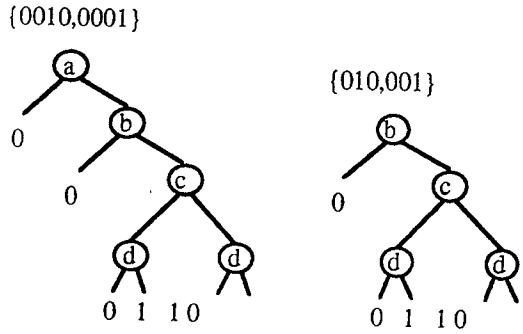

figure III-5: BDD représentant un ensemble de combinaisons.

- **Règle:**

Enlever l'arc à 1 de tous les noeuds dont le fils à 1 correspond au noeud terminal 0. Ensuite connecter cet arc au noeud pointé par l'arc à 0.

Sur la figure III-5, les deux noeuds correspondant aux variables de décision "a" et "b" peuvent être supprimés.

III-2-6-2 Décomposition irredondante

De nombreuses formes de représentations de ROBDDs ont été proposées. Comme pour les ROBDDs, cette structure élimine une variable à chaque étape de la décomposition, mais s'attache ici à éviter la répétition des parties de la fonction qui ne dépendent pas de la variable utilisée comme variable de décision.
La décomposition est réalisée comme suit:

$$F = x.\tilde{F}_x + \bar{x}.\tilde{F}_{\bar{x}} + \tilde{F}_{\neg x}$$

où $\tilde{F}_x$ est la partie de F dépendant uniquement de la variable x, $\tilde{F}_{\bar{x}}$ la partie de F dépendant uniquement de $\bar{x}$ et $\tilde{F}_{\neg x}$ la partie de F ne dépendant ni de x ni de $\bar{x}$.

Si l'on rapporte cette équation à celle correspondant à la décomposition de Shannon, on peut en déduire l'égalité suivante:

$$F = x.F_x + \bar{x}.F_{\bar{x}} = x.\tilde{F}_x + \bar{x}.\tilde{F}_{\bar{x}} + (x + \bar{x}).\tilde{F}_{\neg x}$$

Compte tenu de l'unicité de la décomposition de Shannon, on peut identifier les termes suivants:

$$F_x = \tilde{F}_x + \tilde{F}_{\neg x}$$

$$F_{\bar{x}} = \tilde{F}_{\bar{x}} + \tilde{F}_{\neg x}$$

Le principal atout de cette décomposition réside dans le fait que le terme $\tilde{F}_{\neg x}$ n'est pas répété deux fois, contrairement à la décomposition de Shannon. Grâce à cette propriété, on peut espérer diminuer la taille de la représentation dans la décomposition irredondante. Un noeud possède trois fils.

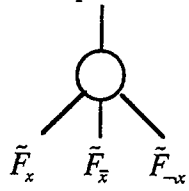

Figure III-6: Noeud tertiaire
de l'arbre de décomposition.

III-2-6-3 If-Then-Else DAG

Cette représentation est une généralisation des ROBDDs. On ne va plus se limiter à des entrées primaires comme variables de décision, mais admettre des sous-fonctions (ou sous-arbre) comme variables de décision d'un noeud. Cela va correspondre a une décomposition de la fonction de la forme:

$$F = \bar{s}.\hat{F}_0 + s.\hat{F}_1$$

La canonicité de la structure est ici aussi perdue. On pourra dans la représentation ou bien utiliser des identificateurs désignant des sous-fonctions, ou bien encore conserver intégralement la structure du ROBDD en donnant la sortie d'un ROBDD en variables de décision d'un autre noeud.

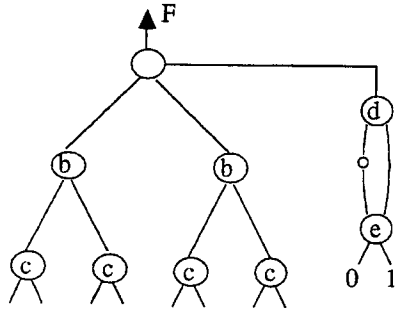

Figure III-7: Exemple de représentation d'un ITE.

Nous verrons plus loin comment peut être utilisée cette structure pour la synthèse et les avantages qu'elle présente.

III-3 Ordonnancement des variables du ROBDD

Nous avons vu les avantages présentés par la structure des ROBDDs, et notamment la canonicité de cette représentation pour un ordre de variables fixé. Cette propriété est particulièrement importante dans le domaine de la vérification logique lorsque le problème consiste à vérifier que la spécification de haut niveau d'un circuit est équivalente à sa réalisation technologique. Dans beaucoup de cas (et notamment pour les fonctions de petite ou de moyenne taille) la représentation ROBDD correspond à une représentation des équations booléennes beaucoup plus compacte que les autres représentations classiques. Néanmoins, la complexité de cette représentation est très fortement liée à l'ordre d'entrée des variables primaires. Le problème consiste donc à trouver l'ordre qui donne le ROBDD le plus petit en terme de nombre de noeuds. Ce problème d'ordonnancement est NP-complet, [15] [16] [17] et il existe diverses méthodes qui tentent de résoudre ce problème. Les principales sont :

- Des heuristiques basées sur des informations issues de la description multi-couche réalisant les fonctions booléennes [14] [18].

- Des méthodes d'amélioration itératives basées sur des échanges de variables (ou swapping) [19] [20] [21] [22].[23].[24].

- Des méthodes analysant la structure de la description en utilisant des propriétés spécifiques [25] [26] [27].

- Des méthodes exactes [17] .

III-3-1 Complexité du problème

Pour illustrer l'importance du problème de l'ordre des variables dans la construction du ROBDD, considérons la fonction;

$F = x_1.x_2+...+x_{2i-1}.x_{2i}+...+x_{2n-1}.x_{2n}$

En choisissant l'ordre:

$x_1 \prec x_2 \prec ... \prec x_{2i-1} \prec x_{2i} \prec ... \prec x_{2n-1} \prec x_{2n}$,

on obtient un ROBDD de taille linéaire (en $O(2n)$);

alors qu'avec l'ordre:

$x_1 \prec ... \prec x_{2i-1} \prec ... \prec x_{2n-1} \prec x_2 \prec ... \prec x_{2i} \prec ... \prec x_{2n}$,

on obtient un ROBDD de taille exponentielle (en $O(2^n)$).

Exemple:

considérons la fonction $F = x_1.x_2 + ... + x_{2i-1}.x_{2i} + ... + x_{2n-1}.x_{2n}$ avec $n = 3$

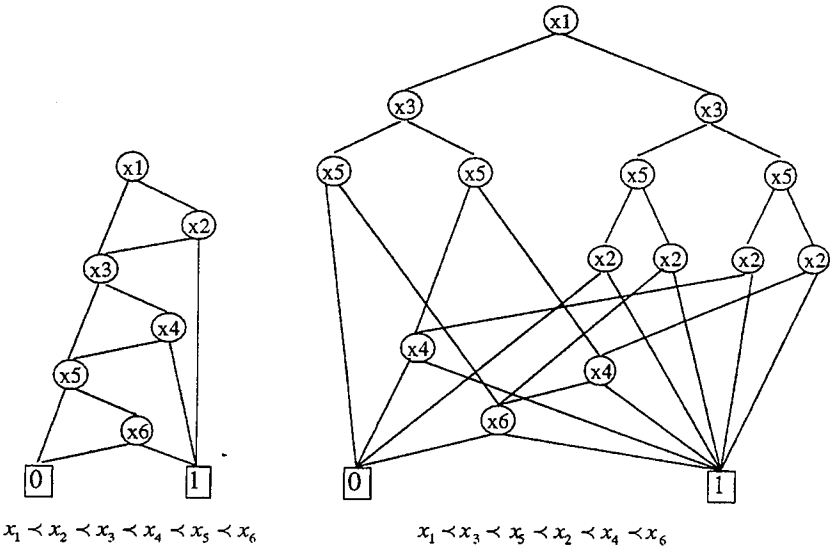

figure III-8: ROBDDs de la même fonction avec deux ordres différents.

Pour cet exemple particulier, le premier ROBDD comporte 6 noeuds, contre 14 pour le second.

L'obtention d'un ordre optimal est, on l'a vu, un problème NP-complet.. Concernant ce problème, nous pouvons rappeler quelques résultats connus:

- Etant donnée une fonction F, il existe un algorithme permettant de déterminer l'ordre qui minimise la taille du ROBDD en terme de nombre de noeuds. Cet algorithme est de complexité $O(n^2 3^n)$ [17] .

- Il existe des fonctions booléennes qui n'admettent pas d'ordre permettant d'obtenir un graphe de taille polynomiale. L'exemple type sont les multiplieurs [16] .

- La taille maximale d'un ROBDD sur n variables est en $O(2^n/n)$ [28] .

III-3-2 Méthodes d'ordonnancement existantes

III-3-2-1 Méthodes exactes

Il existe deux méthodes permettant de calculer un ordre minimisant la taille d'un ROBDD. La première est la recherche exhaustive, qui réalise les $n!$ permutations correspondant à l'ensemble des ordres possibles, et qui construit partiellement le ROBDD pour chacun de ces ordres. Son coût est en $O\left((n!)^2 2^{2n}\right)$. La seconde méthode [17] , restreint la recherche par une élimination de certains cas en utilisant des propriétés que nous verrons plus loin. Sa complexité tombe alors à $O(n^2 3^n)$.

- **Théorème 2:**

Soit F une fonction dépendant des variables $x_1, x_2, \ldots, x_n$. Soit π l'ordre sur les indices $1,2,\ldots,n$, et $Niv(x_i)$ la position de la variable x_i dans π.
Soient i et j, avec $i \prec j$
et notons: $Y = \{y / i \prec Niv(y) \prec j\}$, l'ensemble des variables de la région (ou fenêtre) qui va être réordonnancée.
Tout réordonnacement de Y ne change pas les fonctions booléennes associées aux noeuds de ROBDDs pour les variables x ssi:
$Niv(x) \prec i$ ou $Niv(x) \succ y$.

- Corollaire:

Soit V la variation du nombre de noeuds sur le ROBDD global après réordonnancement de la fenêtre Y, et soit v la variation du nombre de noeuds dans la fenêtre Y, alors :
$V = v$

Cela signifie qu'en diminuant le nombre de noeuds dans une fenêtre fixée Y, on diminue du même nombre de noeuds le ROBDD global sans affecter les noeuds extérieurs à la fenêtre. Cette propriété servira de support à la méthode exhaustive améliorée, ainsi qu'à des heuristiques basées sur le fenêtrage que nous verrons plus loin.

L'intérêt de ce théorème réside dans le fait que l'on va pouvoir améliorer l'algorithme de parcours exhaustif des ordres en éliminant certaines branches de

l'arbre de parcours. Prenons le cas d'une fonction à quatre variables et représentons l'arbre de recherche exhaustif. Cet arbre comprend 4! soit 24 ordres à tester pour obtenir l'ordre optimal.

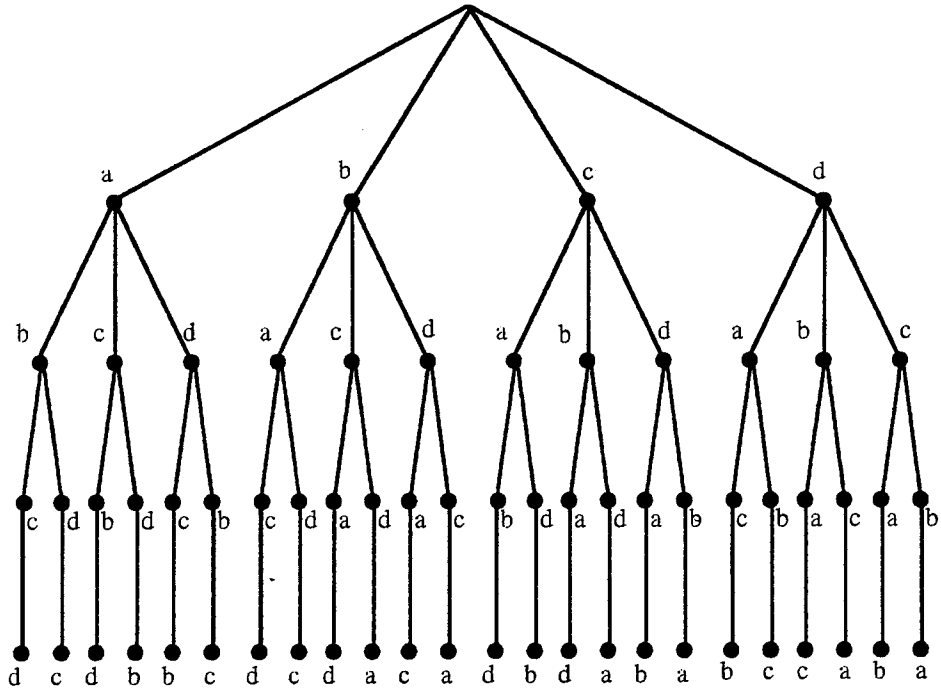

Figure III-10: Arbre décrivant tous les ordres.

En utilisant le théorème vu précédemment, on va pouvoir éviter de parcourir (ou couper) des branches de l'arbre. Considérons la première branche des appels, on construit successivement les deux ordres $a \prec b \prec c \prec d$ et $a \prec b \prec d \prec c$. Supposons maintenant que l'ordre $a \prec b \prec c \prec d$ soit meilleur que le second $a \prec b \prec d \prec c$. Cela signifie que dans le cas ou l'on sépare les variables en deux fenêtres distinctes (a,b) et (c,d), le meilleur ordre pour la seconde fenêtre est $c \prec d$;on sait donc déjà que $b \prec a \prec c \prec d$ est meilleur que $b \prec a \prec d \prec c$.

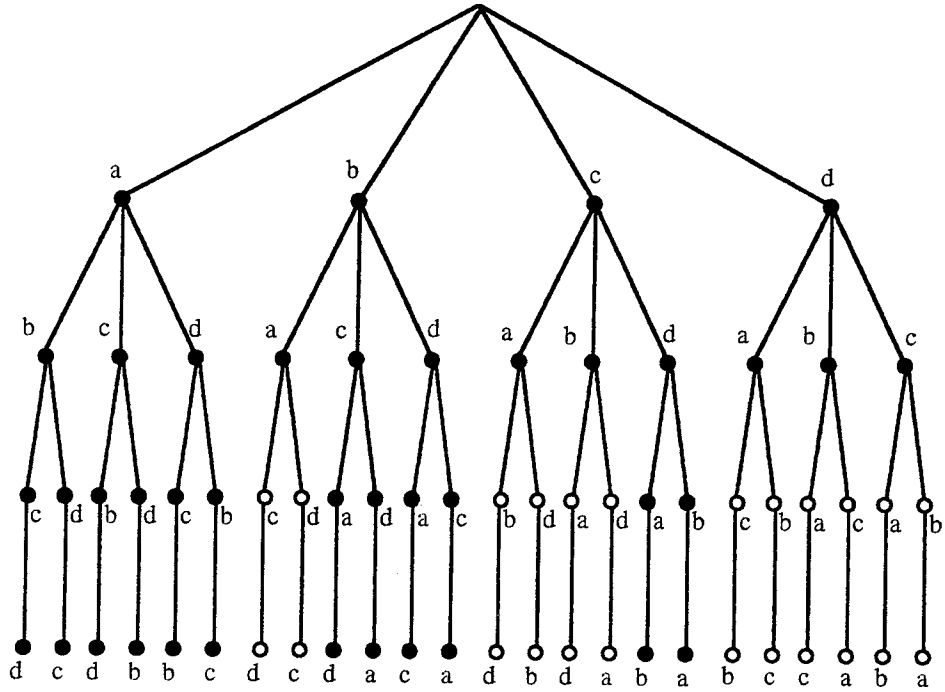

Figure III-11: Arbre réduit de génération des ordres.

La figure III-11 présente l'arbre minimal des appels nécessaires et suffisants à l'obtention du ROBDD de taille minimale.

- **Théorème 3:**

Le nombre de générations d'ordres dans l'arbre restreint est de $(n-2).2^{n-1}+2$

Ces résultats, quoiqu'intéressants, ne peuvent guère être mis en pratique, compte tenu de la complexité des algorithmes que nous venons de voir. Nous allons maintenant voir une série d'heuristiques permettant de trouver des ordres acceptables, ainsi que des combinaisons de ces heuristiques qui peuvent être utilisées.

III-3-2-2 Méthodes heuristiques

Occurrence des variables.

Cette méthode ne peut être utilisée qu'à base de fonctions 2-couches, ou fonctions sous forme de somme de produits. Son principe est très naturel: plus une variable apparaît de fois dans la description, plus elle a de chances d'être une bonne variable de décision.

Les variables seront donc ordonnées suivant leur nombre d'occurrences (sous forme directe ou complémentée) $Nb(x)$ donné par:

$$Nb(x) = OCC(x) + OCC(\bar{x})$$

où $OCC(x)$ est le nombre d'occurrences du littéral x dans l'ensemble des fonctions mises sous forme de somme de produits. Dans le cas où les variables ont le même nombre d'occurrences, on sélectionne celle qui minimise le terme $BAL(x) = |OCC(x) - OCC(\bar{x})|$. Formellement, la relation $\prec$ est construite de la façon suivante:

$$(y \prec x) \Leftrightarrow \big((N(x) < N(y)) \vee \big((N(x) = N(y)) \wedge (BAL(x) < BAL(y))\big)\big)$$

Il est à remarquer que la relation $\prec$ n'est pas totale, puisque x et y peuvent ne pas être ordonnés si et seulement si on a:

$$\big((N(x) = N(y)) \vee (BAL(x) = BAL(y))\big).$$

L'idée de cette méthode des occurrences est que si $F = x.F_x + \bar{x}.F_{\bar{x}}$, après sélection de x, les sous-graphes correspondant à F_x et $F_{\bar{x}}$ restant à construire sont d'autant plus petits que le nombre d'occurrences de x est grand.
Cette méthode n'a de sens que si l'on part d'une décomposition deux-couches de la fonction. Sur une forme multi-couches, il n'est pas toujours possible de connaître l'occurrence exacte d'une variable, car celle-ci peut apparaître dans une sous-fonction utilisée plusieurs fois. Une réinjection des sous-fonctions impliquerait une simplification des expressions booléennes. Cette méthode n'est donc utilisable que dans le cas d'exemples 2-couches.

Exemple:
Soit la fonction vérifiant:
$F = a.\bar{b}.c.d + a.b.$
L'ordre de décomposition sera:
$(b \prec a \prec c \prec d)$ ou $(b \prec a \prec d \prec c)$.

Le problème de cette méthode est que la représentation 2-couches n'étant pas canonique, F peut avoir plusieurs représentations possibles (minimisées ou non). Cela peut influencer fortement le résultat.

Méthode basée sur le nombre inverse des occurrences de variables.

Cette méthode ordonne les variables suivant l'ordre inverse induit par la méthode des occurrences. Cette méthode trouve son intérêt dans des cas très particuliers comme dans le cas des équations itératives, fréquemment utilisées dans les blocs arithmétiques. Ainsi, un additionneur n-bits a comme équation:

$$\begin{cases} c_n = \left(\bigoplus_{i=1}^{n} (a_i \oplus b_i) \right) \oplus c_0 \\ z_n = a_n . b_n + (a_n + b_n) . c_{n-1} \end{cases} \quad \text{avec} \quad \begin{cases} c_i : \text{retenue de la } i^{eme} \text{ cellule} \\ z_i : \text{sortie de la } i^{eme} \text{ cellule} \end{cases}$$

En considérant l'ordre inverse des variables, nous obtenons l'ordre:

$(a_n b_n) \prec (a_{n-1} b_{n-1}) \prec \ldots \prec (a_2 b_2) \prec (a_1 b_1 c_0)$

Cet ordre correspond au flot naturel des variables et décrit un additionneur de type carry-ripple (cf chapitre IV).

Exemple:

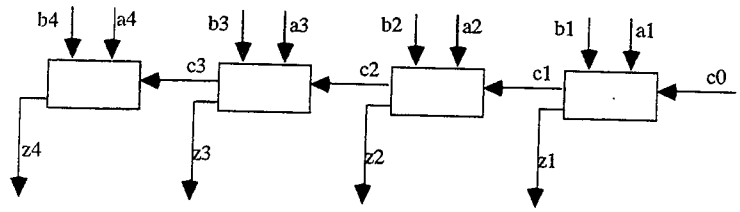

Figure III-12: Exemple d'additionneur 4-bit.

Méthodes utilisant la structure des réseaux de portes.

Pour les gros exemples multi-couches (ou réseaux de portes), d'autres heuristiques sont utilisées [14] , basées sur l'analyse topologique du circuit. Certaines heuristiques peuvent être meilleures que d'autres dans une majorité de cas, mais il n'existe pas de meilleure heuristique dans le cas général.

Méthode de plus petite profondeur.

Cette méthode repose sur des constatations empiriques et fut l'une des premières à être proposée [29] . Soit n_i un noeud du réseau booléen; alors on définit le niveau du noeud par $N(n_i)$ vérifiant:

$$N(n_i) = \underset{n_j \in sort(n_i)}{Max} \left(N(n_j) + 1 \right)$$

où $sort(n_i)$ est l'ensemble des noeuds connectés à la sortie de n_i. Ensuite, chaque noeud est ordonné dans l'ordre décroissant des niveaux. L'ordre final est obtenu en

prenant toutes les entrées primaires de chaque noeud dans l'ordre où ils ont été classées.

Exemple:

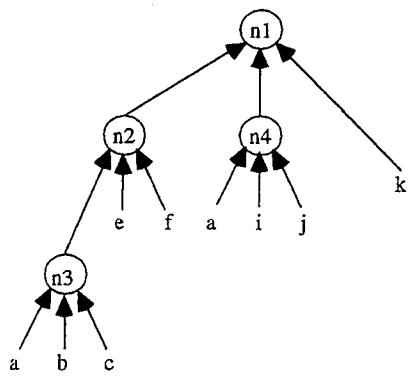

Figure III-13: Calcul de profondeur
et ordonnancement.

Ce réseau induit les ordres suivant:

pour les noeuds: $n_1 < (n_2 n_4) < n_3$;

pour les variables: $k \prec efaij \prec cb$.

Cette méthode tente de retrouver une décomposition en ROBDDs proche de la structure du réseau de portes. En effet, les variables qui arrivent en premier dans le réseau booléen doivent être aussi celles qui apparaissent en premier dans l'ordre des variables de décision du ROBDD. Cette méthode fournit de bons résultats en général.

Méthode d'assignement de poids statique et dynamique.

L'idée générale [14] dans les deux méthodes est de trouver les variables les plus importantes (comme dans le cas de la méthode basée sur les occurrences), ou celles qui ont le plus de "poids". La méthode d'assignement de poids statique est en réalité un cas particulier de la méthode dynamique.

L'algorithme général se déroule comme suit: une constante (1 en général) est propagée depuis le sommet de la fonction jusqu'à ses feuilles. Comme l'illustre la figure III-14, à un noeud donné n_i, cette constante se partage équitablement entre les fils du noeud n_i, et ainsi de suite jusqu'aux entrées primaires. Dans le cas statique, on ordonne l'ensemble des variables primaires (les feuilles du graphe) dans l'ordre décroissant de leur poids. Cet ordre est celui recherché.

Dans le cas de l'assignement de poids dynamique, on réalise pour la première étape le même partage de la constante jusqu'aux racines. Mais ici, on sélectionne la variable qui a le plus grand poids, on la supprime dans tout le graphe, et on réitère l'opération jusqu'à la dernière variable. La méthode statique revient en fait à arrêter la méthode dynamique à la première étape.

Exemple:

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

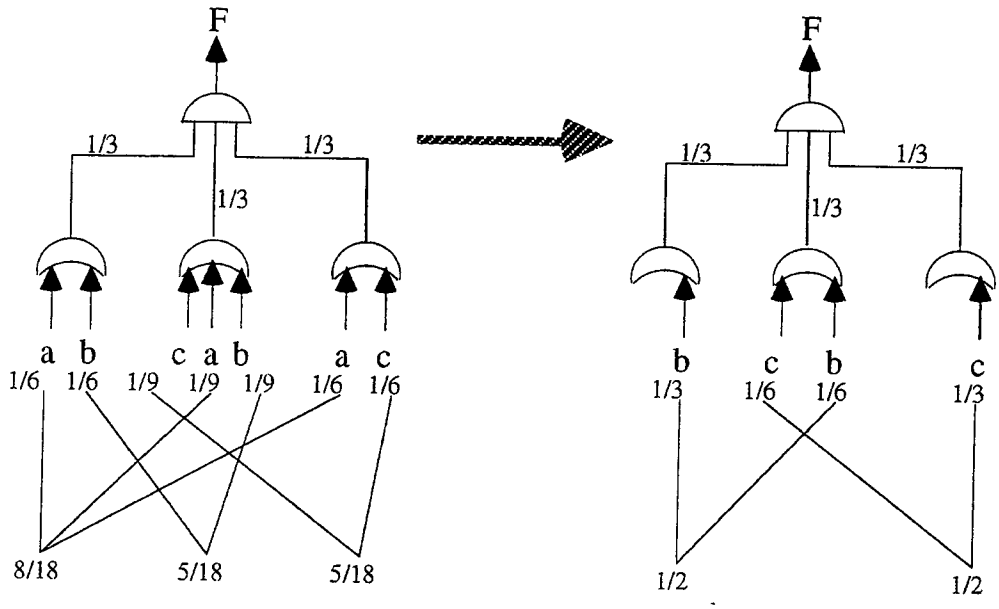

Figure III-14: Assignement de poids dynamique.

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

$a \prec (b,c)$.

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

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

En effet, l'ordre trouvé sera:

$x_1 \prec x_2 \prec ... \prec x_n$

Cet ordre correspond à l'ordre optimal. Avec l'assignement statique, on n'aurait pas pu trouver cet ordre, car le poids de toutes les variables est le même à la première

étape de l'assignement. La méthode des occurrences aurait aussi échoué, la relation d'ordre n'étant que partielle pour cet exemple.

Méthodes par amélioration incrémentale

L'idée de base de cette méthode est d'échanger deux variables adjacentes du ROBDD, afin de produire un nouvel ordre qui conduise à un nouveau ROBDD (Figure III-15). Dans la suite, nous appellerons *SWITCH*(i) la fonction qui échange deux variables adjacentes aux niveaux i et $i+1$.
L'échange de l'ordre des variables implique la mise à jour du ROBDD. Cette opération n'impliquera pas la reconstruction complète du ROBDD, mais une modification locale de sa structure, et le cas échéant des simplifications et réductions de la représentation. Comme démontré dans le théorème prédédent, si l'on considère une fenêtre de taille 2, seuls les noeuds présents dans cette fenêtre sont susceptibles d'être modifiés. De fait, l'échange de deux variables n'obligera pas à reconstruire tout le ROBDD.

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

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

donc

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

Cette équation implique suivant la configuration de départ, l'ensemble des modifications décrites figure III-15.

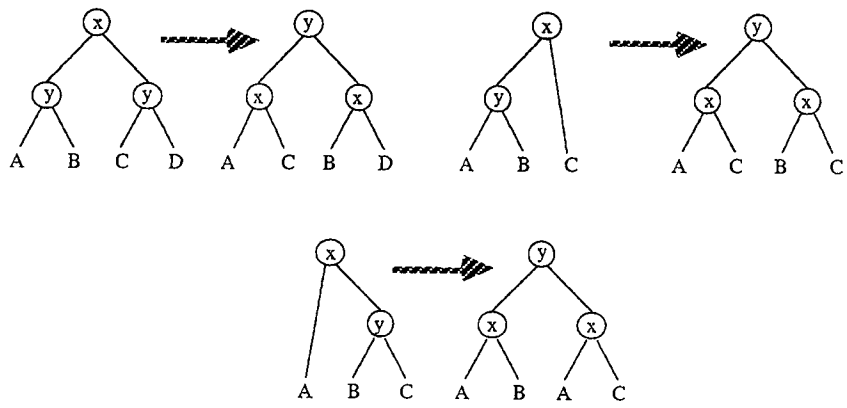

Figure III-15: opération *SWITCH*(i).

L'échange de deux variables adjacentes [30] a donné lieu à l'élaboration de plusieurs méthodes de minimisation de la taille d'un ROBDD. Une extension a été proposée [17] en généralisant la méthode à m variables adjacentes au lieu des deux classiques. La stratégie consiste à échanger les variables au niveau de la variable étiquetant le plus grand nombre de noeuds.

Exemple:

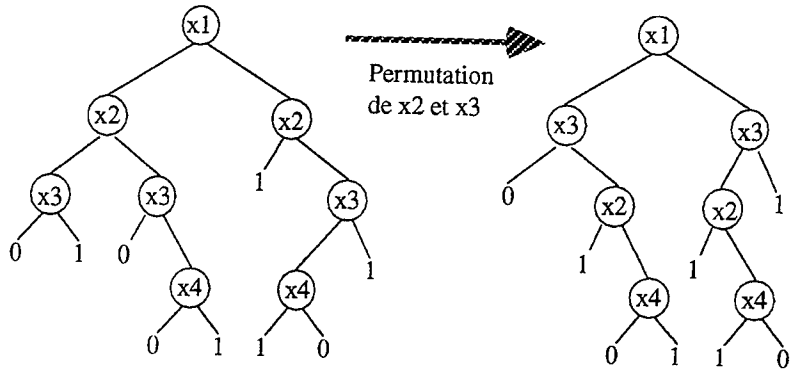

Figure III-16: réduction de la taille par échange de noeuds.

Approche par fenêtre glissante

Une autre approche consiste à réaliser l'opération sur une fenêtre glissante de taille W qui parcourt l'ensemble des variables en permettant l'échange de la variable en position i avec toutes les variables de position allant de $i+1$ à $i+W$. L'algorithme consiste à calculer la taille du ROBDD lorsque sont échangées deux variables comme décrit dans la figure III-16. Si la taille est plus faible, on conserve cet ordre et l'on réitère l'opération pour toutes les variables de la fenêtre avant de la faire glisser.

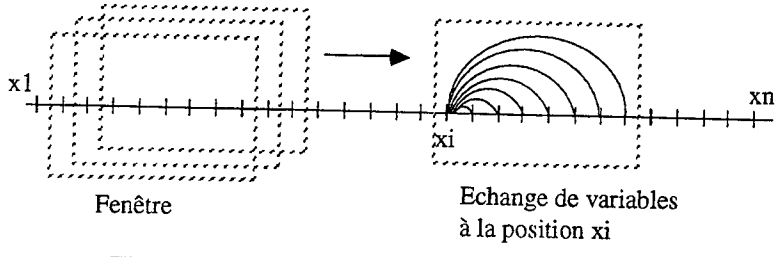

Figure III-17: Approche par fenêtre glissante.

III-3-2-3 Recherche exhaustive améliorée par l'opérateur SWITCH

Comme nous l'avons vu, la complexité de la recherche d'un ordre optimal dépend de deux facteurs qui sont:

- Le parcours de l'ensemble des ordres (ou de l'ensemble réduit des ordres)
- La construction des ROBDDs pour chacun de ces ordres.

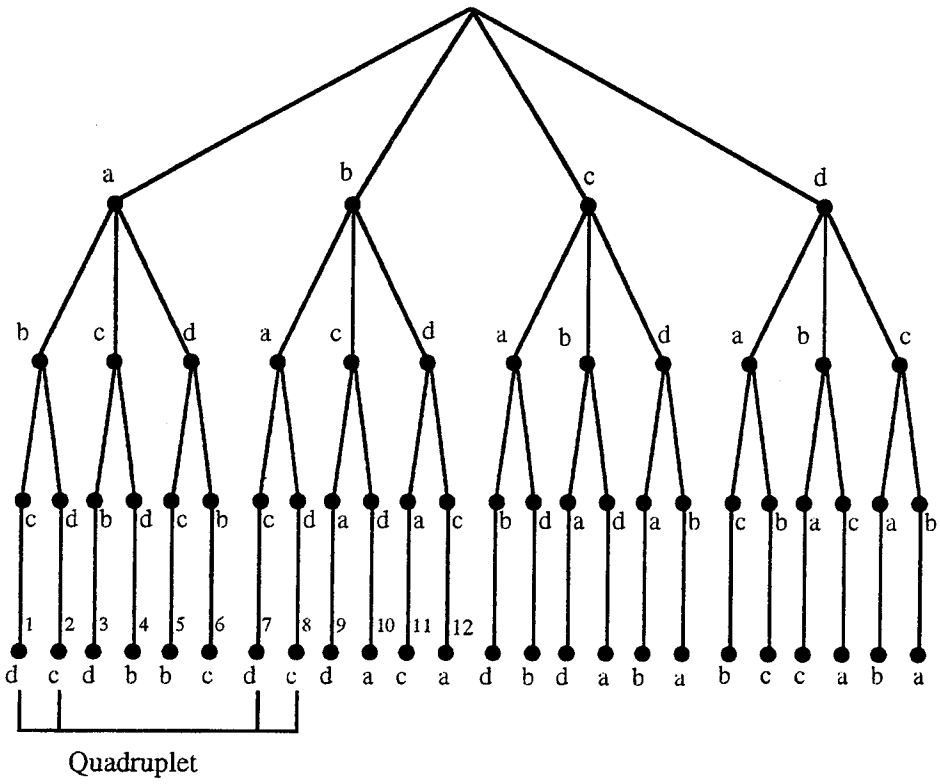

Figure III-18: Arbre exhaustif de génération des ordres

Il est possible de réduire l'espace des ordres à parcourir en utilisant des propriétés découlant du théorème 2. Le nombre d'ordres à visiter passe de $n!$ à $(n-2).2^{n-1}+2$. Sachant que l'opérateur SWITCH permet d'économiser des constructions de ROBDDs en ne modifiant qu'un nombre limité de noeuds, est-il possible de parcourir l'ensemble restreint des ordres avec cette opération?

Considérons le cas de quatre variables (cf Figure III-18); seul ce cas sera étudié, et nous ne nous intéresserons pas au cas général. Comme nous l'avons vu, il est possible de ne visiter que trois des noeuds parmi les six quadruplets suivants:
$\{1,2,7,8\},\{9,10,15,16\},\{4,3,14,13,\},\{12,11,22,21\},\{18,17,24,23\},\{5,6,19,20\}$.

Parcourir l'ensemble des ordres en utilisant l'opérateur SWITCH revient à trouver un chemin hamiltonien dans le graphe d'adjacence des ordres. C'est trivialement possible lorsque l'on veut parcourir l'ensemble exhaustif des ordres, sans tenter de réduire le nombre d'ordres parcourus en éliminant des ordres (on peut en éliminer un par quadruplet). Il n'est par contre pas possible de trouver un chemin hamiltonien dans le graphe si l'on élimine un élément pour chaque quadruplet.

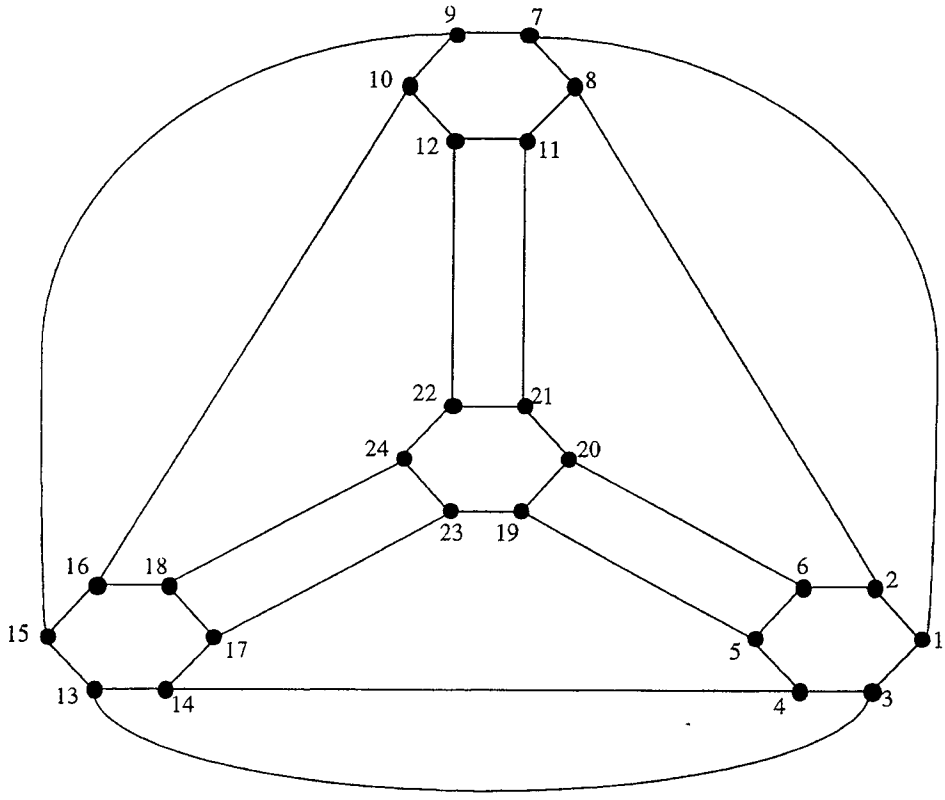

Figure III-19: Graphe d'adjacence

Une solution intermédiaire consiste à éliminer des ordres tant qu'il demeure possible de parcourir tout le graphe sans redondance. On ne peut parcourir un ensemble suffisant d'ordres si l'on élimine 6 ou 5 noeuds du graphe d'adjacence. Il existe toujours un circuit hamiltonien si on élimine les sommets {11,16,18,19} ; il est donc possible de parcourir un nombre suffisant de sommets du graphe pour obtenir le meilleur ordre. La figure III-19 présente la solution qui permet de minimiser le nombre d'opérations SWITCH pour résoudre le problème. La solution consiste à parcourir le chemin décrit dans la figure III-20, sur le sous-graphe du graphe d'adjacence dont ont été éliminés les sommets {11,19,15,16}

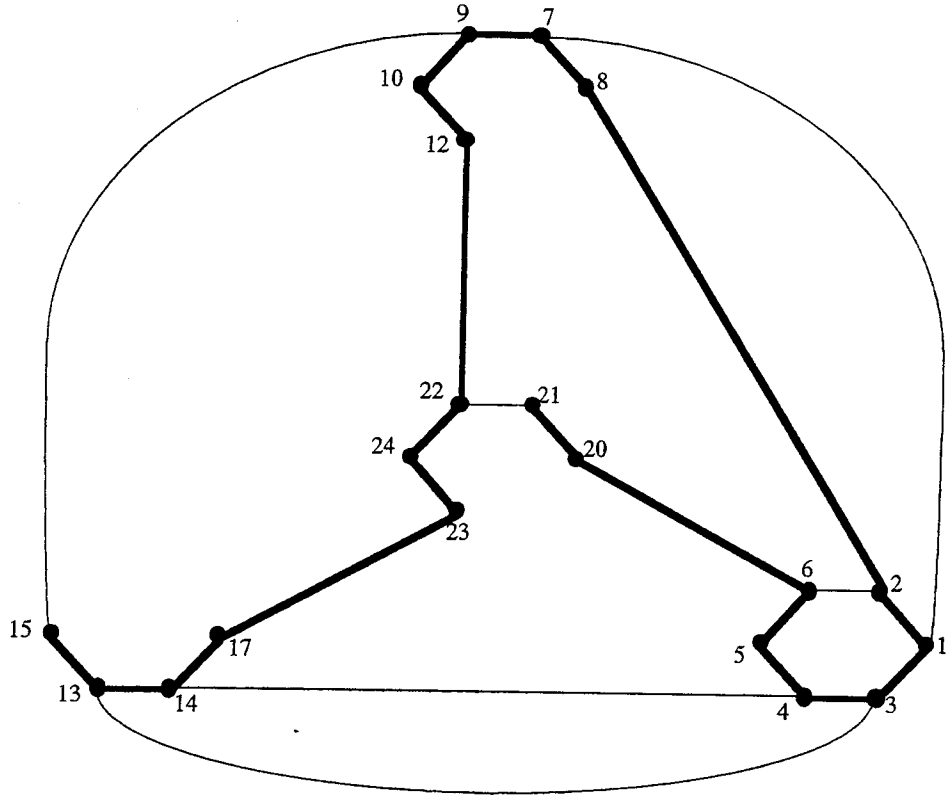

Figure III-20; Parcours optimal des ordres

L'utilisation de l'opérateur SWITCH combiné avec le parcours minimal du nombre d'ordres implique le calcul d'un chemin hamiltonien dans le graphe d'adjacence, ce qui est un problème NP-complet. Ce processus ne semble donc guère envisageable pour des fonctions dont le nombre d'entrées est trop élevé.

III-3-3 Résultats comparatifs

Les évaluatations suivantes sont réalisées sur des benchmarks MCNC. Dans les cas où une structure multi-couches est nécessaire au calcul du nombre de noeuds du ROBDD, on réalise une factorisation algébrique [32] et une procedure de minimisation des fonctions.

Exemples	NAT	OCC	OCC_I	MIN_P	PDS_S	PDS_D	FEN
alu4	1275	1156	1565	1045	1164	1004	582
apex1	3067	2034	4567	2455	2345	2607	1379
apex2	1980	1985	1234	1067	987	934	581
apex3	3234	1123	2897	1358	1209	1098	821
apex4	1188	1345	2399	921	1732	1802	712
apex5	1425	2780	1463	1609	1232	992	1120
bw	102	101	104	100	99	100	95
duke2	1073	805	1230	727	723	658	377
e64	1879	1675	545	396	128423	481	190
ex1010	1067	1071	1055	1070	1065	1064	1025
ex4	987	570	1123	1082	1184	1398	486
ex5	798	456	345	961	370	344	191
misex3	2471	995	1340	1104	1201	1106	382
pdc	875	611	690	496	521	575	415
spla	2433	1230	745	1319	1147	867	598
vg2	855	907	1209	408	398	501	94

NAT : Ordre naturel donné.
OCC : Méthode du nombre d'occurences
OCC_I : Méthode du nombre d'occurences inverse
MIN_P : Méthode de la profondeur minimale
PDS_S : Assignement statique de poids
PDS_D : Assignement dynamique de poids
FEN : Méthode de la fenêtre glissante

L'anarchie des résultats ne permet pas de déterminer quelle heuristique est la meilleure; Neanmoins, La mèthode par fenêtre glissante semble donner sur cet ensemble de benchmarks les résultats les plus satisfaisants.

IV- Décomposition technologique hétérogène sur ROBDDs.

Un réseau de portes (ou netlist) est constitué d'une partie purement combinatoire et d'une partie séquentielle constituée de points mémoires (ou bascules). La partie séquentielle étant fixée et définie, les techniques d'optimisation ne peuvent s'appliquer qu'à la partie combinatoire du réseau. Le but de l'optimisation spatiale est la minimisation de la surface utilisée pour l'implémentation de la partie combinatoire, et ce problème est de taille exponentielle [33] . Ceci est montré en se ramenant aux problème de 3-satisfaisablitè qu nous savons NP-Complet [34].
La taille des blocs logiques étant fixe, le but devient alors de minimiser le nombre de cellules utilisées.
Plusieurs méthodes existent pour répondre à ce problème [35] [36] [37] [38] [39] [40]. Certaines s'appliquent sur une représentation en arbres factorisés (DAG) des fonctions booléennes, d'autres sur des représentations des fonctions en ROBDDs ou ITE. C'est sur cette seconde représentation que va porter le travail .
Deux méthodes de décomposition seront présentées ici. L'une fondée sur le problème de recouvrement et la seconde fondée sur une amélioration de la décomposition de Roth&Karp. Les deux methodes permettent de réaliser des décompositions technologiques hétérogènes [41] [42]. En effet, ces deux methodes peuvent prendre en compte non plus un LUT de taille fixé, mais un ensemble de LUTs de taille différentes et (ou) un ensemble de configurations mélangeant par exemple des LUTs avc des multipleurs.

IV-1 Le flot de synthèse sur les FPGAs Xilinx et AT&T Orca

Les FPGAs sur lesquels va porter le travail sur la projection technologique sont les FPGAs XC5200 de Xilinx et Orca de AT&T décrits dans le chapitre I.

La phase de synthèse se décompose en diverses factorisations et décompositions technologiques. Un ensemble de décompositions est réalisé, et le meilleur est sélectionné. Tout d'abord on va utiliser deux types de représentation des fonctions booléennes: la representation en ROBDDs, et la représentation sous forme d'arbres factorisés ET/OU (Direct Acyclic Graphs). On va plus particulièrement s'intéresser à la décomposition sur les ROBDDs

IV-1-1 Décompositions sur les arbres factorisés ET/OU (DAG)

Cette partie sera mentionnée ici car elle fait partie du flot général de la synthèse. On va procéder à une série de mapping sur les DAGs. Le même algorithme de décomposition est utilisé sur divers graphes. Dans la figure IV-1, le coefficient de reinjection signifie que l'on réinjecte toutes les fonctions possibles, de telle sorte que les supports des fonctions et sous-fonctions créées ne soient pas de taille supérieure à ce coefficient.

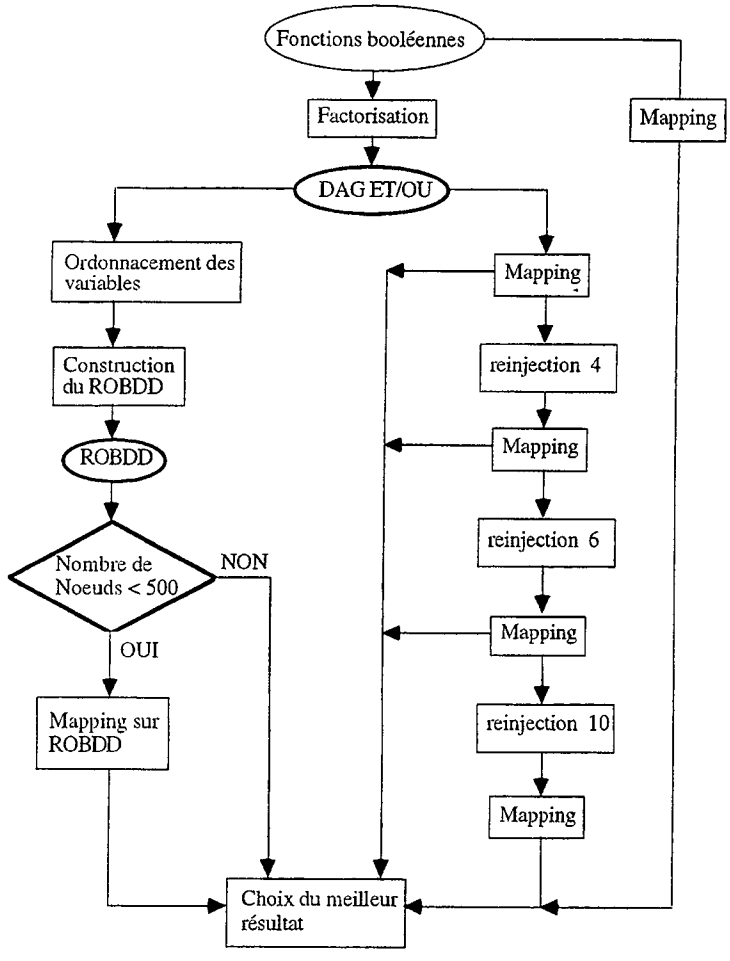

figure IV-1: Organigramme de la synthèse logique.

IV-1-2 Décomposition sur les ROBDDs

L'algorithme consiste à itérer sur un ensemble de ROBDDs (successivement obtenus par modification de l'ordre des entrées) sur lesquels seront réalisées les décompositions technologiques. L'algorithme (glouton) va rester le même, mais l'ordre des variables va évoluer en fonction des résultats des décompositions technologiques. En fait, on va optimiser le ROBDD non plus en fonction du nombre de noeuds, mais en fonction d'une évaluation du nombre de cellules de la technologie-cible issues de la décomposition. Le nombre d'ordre n'excède pas n^2, où n désigne le nombre de variables de la fonction traitée. Cette procédure est rendue possible par la taille relativement faible des fonctions traitées avec les ROBDDs. Une borne supèrieure de 500 noeuds de ROBDD est fixée afin de conserver un temps de calcul acceptable pour tous les passages.

Algorithme glouton de décomposition technologique sur ROBDD.

Il s'agit d'un parcours ascendant du ROBDD avec un marquage glouton des noeuds parcourus en fonction du support de ce noeud. Cet algorithme réalise une décompsotion sur LUT4.

```
classer les noeuds par ordre croissant de profondeur
        ⇒ Liste L
tant que L ≠ ∅
    prendre le premier noeud n de L
    si n est une sortie alors marquer(n)
    si support(n) > 4 alors
        marquer(fils_droit(n))
        marquer(fils_gauche(n))
        marquer(n)
    Si support(n) = 4
        marquer(n)
    L ← L − {n}
```

Explication de l'algorithme:
Le marquage signifie qu'une sous-fonction est créée au niveau du noeud. Le support est calculé au niveau de chaque noeud et donne le nombre de variables et de sous-fonctions dont dépendrait une sous-fonction créée au niveau de ce noeud.

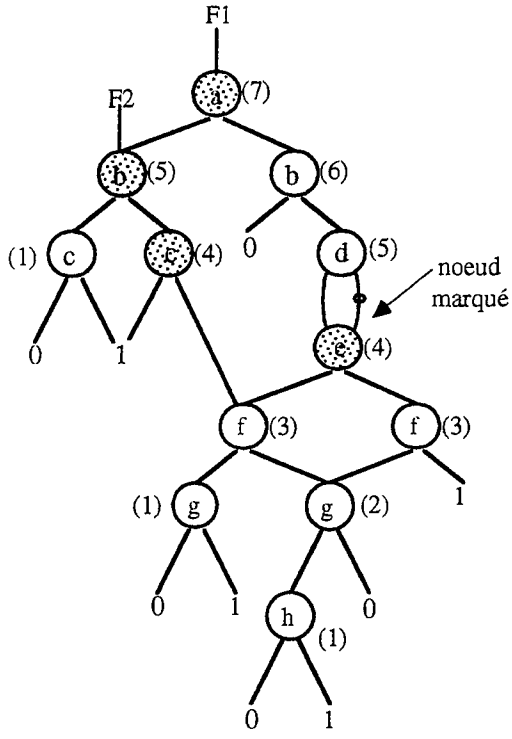

figure IV-2: Marquage des noeuds suivant l'algorithme glouton.

La figure IV-2 montre le résultat obtenu avec cet algorithme. Les chiffres affectés aux noeuds correspondent à leur profondeur et conditionnent le classement dans la liste L et donc le parcours des noeuds.

Cet algorithme de coût linéaire a l'avantage d'être très rapide et de permettre un nombre très élevé de passages. Néanmoins, il ne procède en réalité qu'à une décomposition LUT4 et ne peut prendre en compte des configurations de la cellule plus intéressantes.

IV-2 Méthode de Roth&Karp optimisée pour la décomposition technologique sur ROBDDs

Nous allons ici voir un type de décomposition sur les ROBDDs qui permet de réduire tant la surface que la profondeur du réseau après décomposition, même si ce second élément n'est pas le critère principal. Comme dans la décomposition sur une représentation polynomiale des fonctions booléennes, il est nécessaire de prendre en compte dans cette décomposition les contraintes liées à la cible technologique, ainsi

que le (ou les) critère(s) d'optimisation (surface dans le cas présent). La décomposition génère un ensemble de sous-fonctions qu'il est possible de recouvrir par une cellule ou une partie de la cellule de la technolie-cible.

En 1992, Roth et Karp [43] introduisent une méthode destinée à découper l'ensemble des variables d'une fonction booléenne en sous-ensembles disjoints. Cette procédure permet de traiter ces sous-ensembles de façon indépendante.

Cette méthode va être ensuite utilisée pour réduire la taille des représentations des fonctions booléennes et notamment les BDDs et ROBDDs. [44] utilise ce principe pour créer dans un ROBDD des "supers noeuds". Une approche gloutonne permet d'une part la réduction de la représentation mais aussi une réduction du nombre de cellules nécessaires à la décomposition technologique.

La procédure n'est utilisable que pour des fonctions ne possédant qu'une seule sortie. [45] et [46] vont tenter de généraliser la procédure à des fonctions à sorties multiples. Les fonctions à sorties multiples seront ici traitées grâce à une phase d'unification de l'ensemble des sorties à l'aide d'un BDD couvrant l'ensemble des sorties.

Nous nous attacherons ici à intégrer de façon très précise le coût en terme de cellules de chaque décomposition et à voir cette décomposition au niveau du ROBDD total et non plus de façon locale. Cela va nous amener à parcourir l'ensemble des décompositions possibles et à élaborer une stratégie permettant de réaliser une suite de décomposition de façon optimisée. La décomposition technologique sera réalisée sur les FPGAs à base de LUTs de type AT&T ORCA.

IV-2-1 Définitions et généralités

- **Définition 27: Bound-set, Free-Set**

Une fonction $F(x_0, x_1, ..., x_{n-1})$ est dite décomposable sous un bound-set (ou un support fixé) $\{x_0, x_1, ..., x_{i-1}\}$, et un free-set (ou support libre) $\{x_{i-k}, ..., x_{n-1}\}$ avec $0 \leq k < i$ si on peut écrire la fonction sous la forme:
$F(x_0, x_1, ..., x_{n-1}) = F'(G_0(x_0, ..., x_{i-1}), ..., G_{j-1}(x_0, ..., x_{i-1}), x_{i-k}, ..., x_{n-1})$,
$0 < j < i-k$, ou les G_i sont des fonctions booléennes.
Les fonctions G_i seront appelées fonctions associées.

Si $k=0$, alors la fonction $F(x_0,x_1,...,x_{n-1})$ est dite disjonctivement décomposée. Autrement dit, cela signifie que les variables dont dépendent les fonction G_i n'apparaissent plus dans la fonction F'.

Si $k>0$, alors la fonction $F(x_0,x_1,...,x_{n-1})$ est dite non-disjonctivement décomposée.

Si $j=1$, alors la fonction est dite simplement décomposée.

Exemple:

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

Décomposition disjonctive:
$$\begin{cases} G_0=a+b \\ G_1=c.\overline{d} \\ F=G_0.G_1.e.f+G_1.e.g \end{cases}$$

Décomposition non-disjonctive:
$$\begin{cases} G_0=a+b \\ G_1=e.f \\ G_2=c.\overline{d} \\ F=G_0.G_1.G_2+G_2.e.g \end{cases}$$

Décomposition simple:
$$\begin{cases} G_0=a+b \\ F=G_0.e.f.c.\overline{d}+c.\overline{d}.e.g \end{cases}$$

- **Définition 28: fonction *eval(v,B)***

Soient le v ROBDD d'une fonction booléenne $F(x_0,x_1,...,x_{n-1})$ et le vecteur:

$B=[b_0,...,b_{i-1}]/\ b_k\in\{0,1\},0\le k\le i-1.$

$v'=eval(v,B)$ est le ROBDD représentant $F(b_0,...,b_{i-1},x_i,...,x_{n-1})$

On utilisera aussi:

$eval(v,j)=eval(v,B)$ où $j=2^{i-1}b_0+...+2^0b_{i-1}$

- **Définition 29: *Cut_set(v,level)*** (ou *Coupe(v,level)*)

Soit v le ROBDD représentant la fonction $F(x_0,x_1,...,x_{n-1})$, on définit pour $0\le level\le n-1$:

$cut\_set(v,level)=\{u/\ u=eval(v,i),0\le i<2^{level+1}\}$,

Autrement dit, c'est l'ensemble des noeuds du ROBDD v de profondeur *level*.

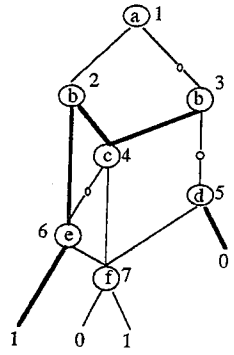

Figure IV-3: Exemple de ROBDD

Dans la figure IV-3, nous pouvons donner:

$$Cut\_set(Noeud1,2) = \{Noeud5, Noeud6, Noeud7\}$$

La notion de Cut-Set ne tient pas compte du type des arcs incidents sur les noeuds. On va introduire la notion de Cut-set typé qui tiendra compte des inverseurs qui modifient la fonctionnalité de la sous-fonction associée à un noeud.
On notera $\overline{v}$ le sous-ROBDD successeur d'un arc indirect.

- **Définition 30: fonction** ***eval_t(v,B)***

Soit le ROBDD v et le vecteur $B = [b_0,...,b_{i-1}] / b_k \in \{0,1\}, 0 \le k \le i-1$.
$eval\_t(v,B) = v'$, si v est le ROBDD représentant $F(b_0,...,b_{i-1},x_i,...,x_{n-1})$ et est atteint par un arc direct;
$eval\_t(v,B) = \overline{v}'$, si v est le ROBDD représentant $F(b_0,...,b_{i-1},x_i,...,x_{n-1})$ et est atteint par un arc indirect.

- **Définition 31:** $Cut\_set\_t(v,level)$

Soit v le ROBDD représentant la fonction $F(x_0,x_1,...,x_{n-1})$, on définit pour $0 \le level \le n-1$:
$cut\_set\_t(v,level) = \{u / u = eval\_t(v,i), 0 \le i < 2^{level-1}\}$.

IV-2-2 Décomposition de Roth&Karp

Cette décomposition conduit à la modification du ROBDD et le transforme en une représentation ITE (If-Then-else, cf. chapitre II). Les fonctions utilisées comme variables de décision des noeuds de l'ITE vont être les sous-fonctions associées à une décomposition disjonctive du ROBDD.

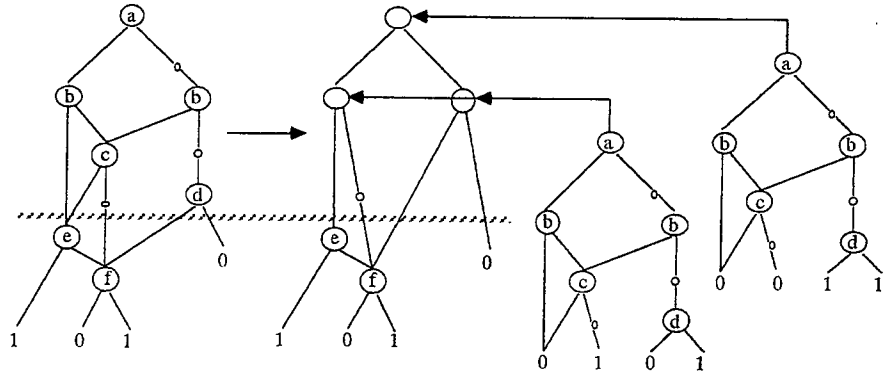

Figure IV-4: Décomposition de Roth&Karp.

Considérons un ROBDD v de racine unique. Par définition même d'un ROBDD, une variable présente dans les k premiers niveaux sera absente du reste du ROBDD. Donc, la décomposition sera forcément disjonctive.

La décomposition de Roth&Karp conduit à la formation d'une structure ITE. Considérons une coupe de l'arbre au niveau k. On a donc le bound_set $\{x_0,\ldots,x_{k-1}\}$ et le free_set $\{x_k,\ldots,x_n\}$. Calculons maintenant le nombre p de sous-fonctions G_j qui seront nécessaires à la décomposition. Ces fonctions doivent permettre de "décider" entre les différentes sous-fonctions représentées par sous-ROBDDs ayant les noeuds de la coupe comme racines. Il faut bien sur tenir compte des inverseurs qui sont susceptibles de changer la fonctionnalité des noeuds. Il faut donc construire un arbre capable de couvrir toutes ces sous-fonctions.
Compte tenu de la structure en ITE, nous avons:

$$2^p = |cut\_set\_t(v,k)|$$

donc:

$$p = \lceil \ln_2 |cut\_set\_t(v,k)| \rceil$$

Lors de cette transformation, nous créons exactement:

$cut\_set\_t(v,k) - 1$ nouveaux noeuds.

IV-2-3 Décomposition récursive hétérogène

La décomposition de Roth&Karp (ou une autre forme de décomposition sur un ROBDD) peut être appliquée récursivement, voire jusqu'aux feuilles du ROBDD. Une première approche gloutonne consiste à réaliser la décomposition tant que celle-ci présente un intérêt en terme de surface et de la stopper lorsque la décomposition détériore le résultat. Une seconde approche tente de prendre en compte la totalité du ROBDD et conduit à l'utilisation d'une heuristique plus

complexe que nous allons développer. Nous supposerons ici qu'il est possible de fournir le coût en terme de cellules LUTs de k variables. Cette supposition s'avère vraie pour des valeurs de k relativement petites, lorsqu'on se place dans le cas où la technologie cible est le FPGA AT&T ORCA. Nous considérerons aussi un coefficient C_n représentant le coût moyen de couverture d'un noeud du ROBDD. Cette approximation, comme nous le verrons, sera peu utilisée et n'interviendra qu'en fin de la décomposition, n'entraînant pas ainsi une trop grande incertitude au niveau du résultat de la factorisation et du mapping qui en découlera.

On notera:

$C_m(k)$	le coût du mapping d'une fonction à k variables.
M	Le nombre de noeuds du ROBDD dont les variables de décisions associées sont incluses dans le bound-set (ou support fixé).

Nous avons vu que pour une décomposition du ROBDD v au rang k, nous générions $p = \lceil \ln_2 \lvert cut\_set\_t(v,k) \rvert \rceil$ sous-fonctions qui servaient de variables de décision pour un ensemble de $cut\_set\_t(v,k) - 1$ noeuds.

Donc le gain en terme de surface de la décomposition est :

$$G(v,k) = C_n.M - C_m(k)\lceil \ln_2 \lvert cut\_set\_t(v,k) \rvert \rceil - C_n.(\lvert cut\_set\_t(v,k) \rvert)$$

Exemple:

Considérons le cas où le ROBDD est décomposé jusqu'au niveau 4.

Nous avons les valeurs suivantes:

$$\begin{cases} Bound\_set(v,4) = \{a,b,c,d\} \\ Free\_set(v,4) = \{e,f,g,h\} \\ \lvert cut\_set(v,4) \rvert = 2 \\ \lvert cut\_set\_t(v,4) \rvert = 4 \end{cases}$$

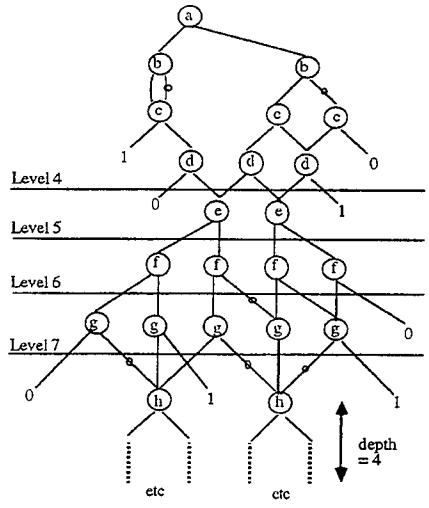

figure IV-5: ROBDD de départ

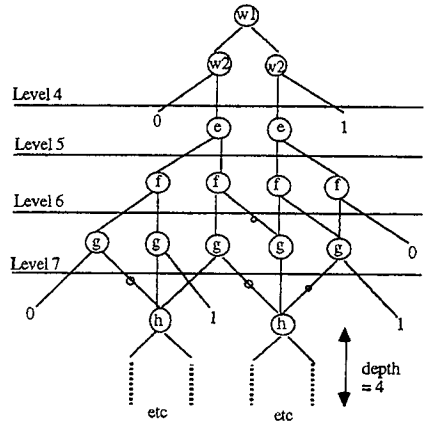

figure IV-6: ITE après décomposition au niveau 4

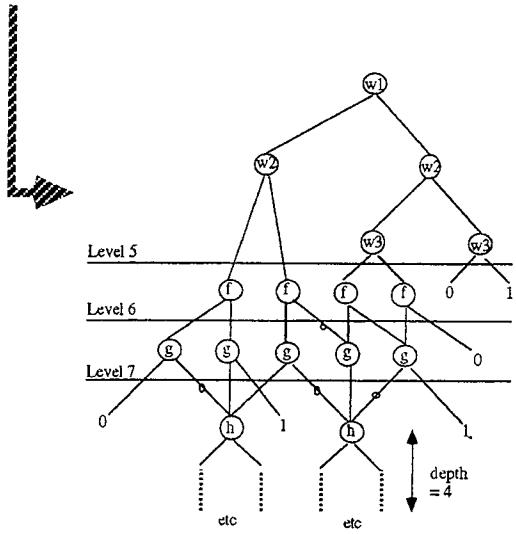

figure IV-7: ITE Après décomposition au niveau 5.

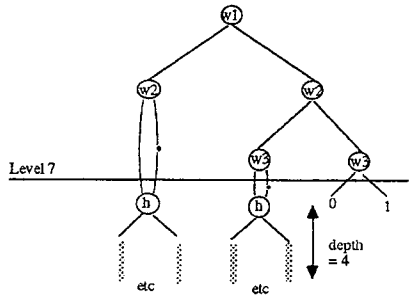

figure IV-8: Décomposition de Roth&Karp jusqu'au niveau 7

A partir du ROBDD de départ (figure IV-5), plusieurs possibilités s'offrent pour réaliser la décomposition. Les figures IV-6 et IV-7 présentent les décompositions réalisées au niveau 4 et au niveau 5. La figure IV-8 présente la décomposition réalisée jusqu'au niveau 7 du ROBDD. Cette décomposition peut être obtenue à partir du ROBDD de départ, mais aussi des deux précédentes décompositions décrites dans les figures IV-6 et IV-7. Il s'agit maintenant d'évaluer les gains associés à ces deux transformations. On prendra comme technologie cible le FPGA AT&T ORCA.

On considère l'évaluation:

$$C_n \cong \frac{1}{6}$$

Cette approximation est obtenue en considérant que chaque LUT4 de la cellule ainsi que les deux multiplexeurs liant les LUTs deux à deux couvrent un noeud du ROBDD (cf. chapitre I).

Lors de l'utilisation des LUT4, il faut tenir compte des contraintes liées aux entrées. Dans le cas où l'on couvre des noeuds de ROBDDs, il suffit de couvrir deux noeuds de même variable.de décision pour satisfaire les contraintes d'entrée. Lors d'une étape de décomposition de Roth&Karp, les sous-fonctions engendrées dependent toutes des mêmes variables. Donc les contraintes liées aux entrées sont satisfaites.

Et nous avons les valeurs:

$$C_m(4) = \frac{1}{4}$$

$$C_m(5) = \frac{1}{2}$$

$$C_m(6) = 1$$

donc, dans le cas $k = 4$:

$$G(v,4) = C_n.M - C_m(4)\lceil \ln_2 |cut\_set\_t(v,4)| \rceil - C_n.(|cut\_set\_t(v,4)|)$$

$$G(v,4) = \frac{1}{6}.9 - \frac{1}{4}\lceil \ln_2(4) \rceil - \frac{1}{6}.(4-1)$$

$$G(v,4) = \frac{1}{2}$$

et pour $k = 5$:

$$G(v,5) = C_n.M - C_m(5)\lceil \ln_2 |cut\_set\_t(v,5)| \rceil - C_n.(|cut\_set\_t(v,5)|)$$

$$G(v,5) = \frac{1}{6}.11 - \frac{1}{2}.\lceil \ln_2(6) \rceil - \frac{1}{6}.(6-1)$$

$$G(v,5) = -\frac{1}{2}$$

Ces calculs de gains nous montrent qu'il est à priori intéressant de procéder à la décomposition au niveau 4 du ROBDD.

IV-2-4 Approche Gloutonne

L'évaluation du gain pour chaque décomposition peut induire un algorithme glouton de décomposition récursive. Dans cette approche, comme dans l'approche plus complexe que nous verrons plus loin, nous fixerons pour le coefficient k une borne supérieure et une borne inférieure $l_{\inf}, l_{\sup}$. Ces valeurs vont dépendre de la technologie cible sur laquelle la décomposition sera réalisée.

Calcul du gain $G(v,k)$ pour $l_{Inf} \le k \le l_{Sup}$
Si $G(v,k) < 0, \forall k \,/\, l_{Inf} \le k \le l_{Sup}$ alors arrêt de la décomposition
Sinon
    Sélectionner $k \,/\, G(v,k) = \underset{l_{\inf} \le k \le l_{\sup}}{Sup} G(v,k)$
    Décomposer le graphe au rang k
    Soit v' le graphe obtenu
    affecter $v = v'$

Cette approche présente l'avantage de la simplicité, mais l'utilisation de la valeur approximative C_n biaise les résultats. De plus cette approche ne permet pas des décompositions susceptibles d'être intéressantes en terme de surface mais qui implique une décomposition préalable à priori dommageable.

IV-2-5 Approche globale

Dans la suite, nous nommerons "position k" noté $pos(k)$ le graphe ITE obtenu après décomposition récursive de Roth&Karp jusqu'au niveau k du ROBDD v. Quelle que soit la manière d'obtenir une décomposition (ou position) à un niveau k quelconque, celle-ci ne dépend que de la valeur $cut\_set\_t(v,k)$ qui détermine le nombre de noeuds et de sous-fonctions créées à ce niveau. Des divers moyens de

parvenir jusqu'au niveau k dépendront les sous-fonctions créées et affectées en tant que variables de décision. On peut donc établir un graphe de décomposition, G(X,E) qui aura pour sommets les diverses positions décrites ci-dessus, et l'on définit un arc entre les positions i et j si et seulement si il existe une décomposition permettant de relier ces deux sommets. Aux arcs est affecté le gain associé à la décomposition correspondante.
Lorsque la décomposition récursive est arrêtée, un ensemble de noeuds reste à la base du ROBDD de départ. De manière gloutonne, les noeuds de profondeur inférieure ou égale à 4 peuvent être mappés sur des LUT-4. Pour les noeuds de profondeur strictement supérieure à 4, on peut utiliser l'approximation C_n.
On peut ainsi couvrir la décomposition jusqu'à la base du ROBDD.

Notons $pos(fin)$ la position correspondante à la fin de la décomposition. Il s'agit maintenant de déterminer le coût associé aux arcs menant à cette position finale. Nous avons vu que pour les noeuds de taille inférieure ou égale à quatre est utilisé un algorithme glouton de décomposition et pour les noeuds de tailles supérieure, est utilisé l'approximation C_n: On considérera donc que les noeuds de profondeur cinq sont les derniers noeuds à mapper et constituent la base du ROBDD que nous traitons.

Le coût associé aux arcs menant à la position $pos(fin)$ est égal à:

$C_n.M_r$, où M_r représente le nombre de noeuds restant de profondeur strictement supérieure à quatre.

Notons $M_{p\to q}$ le nombre de variables contenues dans la partie supérieure du graphe si l'on procède à la décomposition au rang q, sachant que la dernière décomposition réalisée le fut au rang p. La figure IV-9 décrit le graphe obtenu après décomposition au rang p, sachant que le ROBDD était décomposé jusqu'au rang o et montre ainsi le lien qui existe entre les trois valeurs successives $M_{O\to P}$, $M_{O\to Q}$, et $M_{P\to Q}$:

$$M_{O\to Q} = M_{O\to P} + M_{P\to Q} - \left(|cut\_set\_t(v,p)| - 1\right)$$

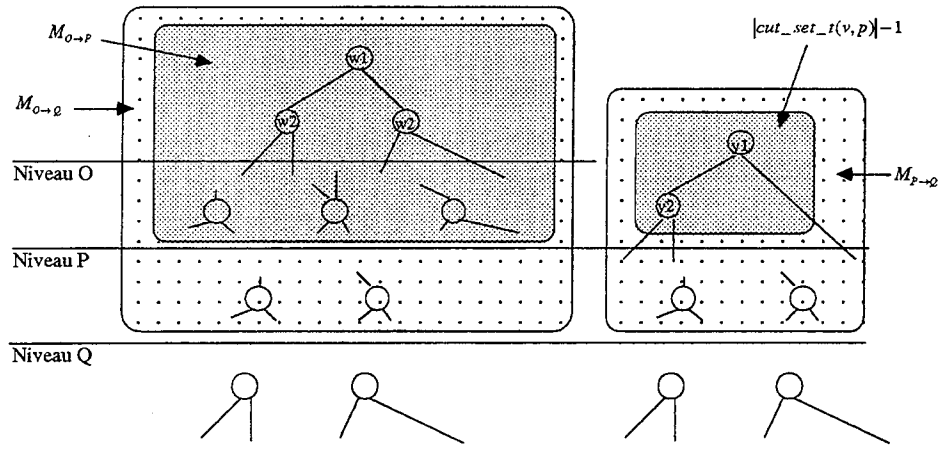

figure IV-9: Relation entre les valeurs $M_{O\to P}$, $M_{O\to Q}$, et $M_{P\to Q}$.

Calculons maintenant le coût associé à la décomposition successive du ROBDD aux niveaux p et q, sachant que le ROBDD est déjà décomposé jusqu'à niveau o.

$$G = G(v,p) + G(v',q)$$

$$G = C_n.M_{o\to p} - C_m(k_p).\lceil \ln_2|cut\_set\_t(v,p)|\rceil - C_n.(|cut\_set\_t(v,p)|-1)$$

$$+C_n.M_{p\to q} - C_m(k_q).\lceil \ln_2|cut\_set\_t(v,q)|\rceil - C_n.(|cut\_set\_t(v,q)|-1)$$

donc

$$G = C_n.(M_{o\to p} + M_{p\to q} - (|cut\_set\_t(v,p)|-1)) - C_m(k_p).\lceil \ln_2|cut\_set\_t(v,p)|\rceil$$

$$-C_m(k_q).\lceil \ln_2|cut\_set\_t(v,q)|\rceil - C_n.(|cut\_set\_t(v,q)|-1)$$

d' où

$$G = C_n.M_{o\to q} - C_m(k_p).\lceil \ln_2|cut\_set\_t(v,p)|\rceil - C_m(k_q).\lceil \ln_2|cut\_set\_t(v,q)|\rceil$$

$$-C_n.(|cut\_set\_t(v,q)|-1)$$

En calculant le gain obtenu entre la position de départ et la dernière position où est effectuée la décomposition, on obtient:

$$G = C_n.M_{deb\to i_n} - \sum_{p=i_1,\ldots,i_n} C_m(k_p).\lceil \ln_2|cut\_set\_t(v,p)|\rceil - C_n.(|cut\_set\_t(v,i_n)|-1)$$

où:

$i_0,\ldots,i_n$ représentent les niveaux où furent réalisées les décompositions

Or d'après la relation décrite figure IV-9, nous avons:

$$M_{deb\to fin} = M_{deb\to i_n} + M_{i_n\to fin} - (|cut\_set\_t(v,i_n)|-1)$$

donc:

$$G = C_n.(M_{deb\to fin} - M_{i_n\to fin}) - \sum_{p=i_1,\ldots,i_n} C_m(k_p).\lceil \ln_2|cut\_set\_t(v,p)|\rceil$$

Maximiser ce gain revient à minimiser son opposé. Le terme $M_{deb \to fin}$ représentant une constante, il peut être supprimé lors de la minimisation. On obtient donc le suivant:

$$Min\left(C_n.M_{i_n \to fin} + \sum_{p=i_1,\ldots,i_n} C_m(k_p).\left\lceil \ln_2 |cut\_set\_t(v,p)| \right\rceil \right)$$

Sur le graphe défini précédemment, aux arcs menant de la position p à la position q,on affectera le coût:

$$Coût_{p \to q} = C_m\left(q - p + \left\lceil \ln_2\left(|cut\_set\_t(v,p)|\right)\right\rceil\right).\left\lceil \ln_2\left(|cut\_set\_t(v,q)|\right)\right\rceil,$$

car $\left(q - p + \left\lceil \ln_2\left(|cut\_set\_t(v,p)|\right)\right\rceil\right)$ représente le nombre de variables présentes dans la partie supérieure du ROBDD lorsqu'est réalisée la décomposition au niveau q.

Et aux arcs menant d'une position q à la position *fin*:

$$C_n.M_{q \to fin}$$

On obtient ainsi un problème de plus court chemin sur le graphe de décomposition défini précédemment. L'approximation C_n n'apparaît plus que dans les arcs ayant pour extrémité la position finale. En fait, $C_n.M_{i \to fin}$ représente le coût du mapping des noeuds lorsqu'est arrêtée la décomposition. Si l'on décide de s'affranchir de l'approximation C_n, on peut calculer la valeur réelle de ce coût. Ces décompositions technologiques étant réalisées sur des ROBDDs de taille réduite, leurs coûts en temps CPU en sera d'autant plus réduit.

Exemple:

Considérons le ROBDD décrit figure IV-5. On va calculer le coût associé aux arcs entre la position 0 et 5, et entre la position 4 et 6.

Cas $pos(0) \to pos(5)$:

On a les valeurs: $\begin{cases} p = 0 \\ q = 5 \end{cases}$ donc $\begin{cases} |cut\_set\_t(v,0)| = 1 \\ |cut\_set\_t(v,5)| = 6 \end{cases}$

Donc:

$$Coût_{0 \to 5} = C_m(5).3 = \frac{1}{2}.3 = \frac{3}{2}$$

Cas $pos(4) \to pos(6)$:

On a les valeurs: $\begin{cases} p = 4 \\ q = 6 \end{cases}$ donc $\begin{cases} |cut\_set\_t(v,4)| = 4 \\ |cut\_set\_t(v,6)| = 8 \end{cases}$

Donc:

$$Coût_{4\to 6} = C_m(4).3 = \frac{1}{4}.3 = \frac{3}{4}$$

on peut en déduire le graphe de décomposition suivant:

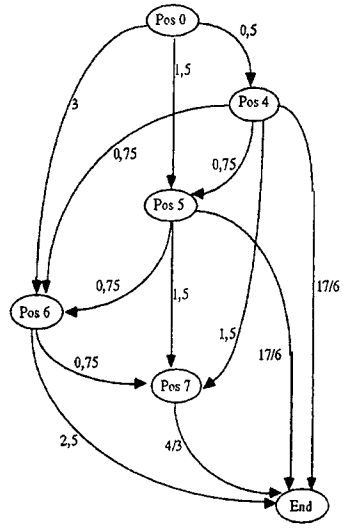

figure IV-10: Graphe de décomposition.

La résolution du probleme de plus court chemin dans le graphe de décomposition [47] [48] nous donne la (ou les) meilleure(s) solution(s). Dans ce cas, nous avons deux chemins donnant la même valeur de $10/3$:

$$\begin{cases} 0 \to 4 \to fin \\ 0 \to 4 \to 7 \to fin \end{cases}$$

La décomposition est donc réalisable avec $\lceil 10/3 \rceil = 4$ cellules AT&T ORCA.

IV-2-6 Procédure d'unification

Nous avons vu que la décomposition de Roth&Karp du ROBDD s'effectue sur un ROBDD de racine unique. Dans la majorité des cas, on est en présence d'un ensemble de ROBDDs, dont les sommets représentent des sortie primaires de la fonction booléenne que l'on traite. Plusieurs solutions ont été proposées pour la prise en compte de fonctions à sorties multiples. [46] propose la notion de *Cut_set _multi_outputs*, qui réalise l'union des Cut_sets pour l'ensemble des sorties. Ici, nous allons tenter d'étendre notre approche par le rajout d'un ensemble de variables de décision au niveau de l'ensemble de ROBDDs. Un BDD va être rajouté sur la

partie supérieure du vecteur de ROBDDs, de taille suffisante pour que ses feuilles puissent couvrir l'ensemble des sorties.

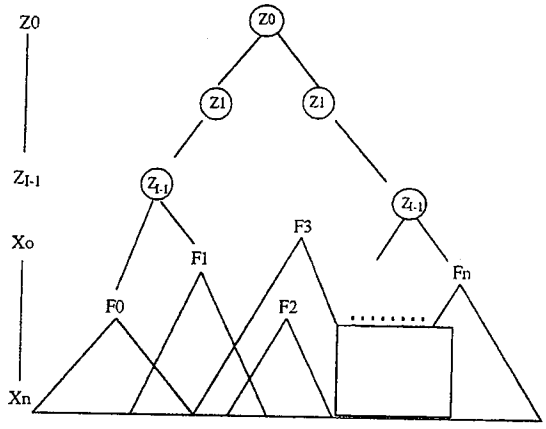

figure IV-11: Ajout de variables de décision.

Notons Z_i les variables de décision du BDD rajouté. Le BDD devant couvrir l'ensemble des sorties de la fonction booléenne, on a besoin de:

$I = \lceil \ln_2(m) \rceil$ variables. Z_i

où m est le nombre de sorties.

A chaque sortie primaire correspond un vecteur de Z_i.

Exemple:

à la sortie F_1 correspond le vecteur $(Z_0, Z_1, ..., Z_{I-2}, Z_{I-1}) = (0,0,...,0,1)$

Après cette opération, les variables de décision Z_i qui ont été rajoutées sont placées en fin de l'ordre des variables d'entrée, et le ROBDD est reconstruit avec ce nouvel ordre. Cette modification est apportée afin que la décomposition touche en priorité les variables réelles et non les variables auxiliaires. En fin de mapping, on extencie les variables Z_i afin de récupérer les valeurs des sorties.

IV-2-7 Résultats expérimentaux

Nous allons maintenant voir les résultats obtenus en utilisant la procédure de décomposition technologique décrite ici. Le point de départ est une fonction booléenne sous la même forme pour les deux décompositions comparées ici. L'algorithme auquel on se compare est l'algorithme glouton décrit en debut de chapitre. Le ROBDD est ici construit en partant de deux ordres différents: la

méthode fondée sur le nombre d'occurence notée "OCC" et la méthode fondée sur le poids des variablesnotée "PDS" (cf chapitre III). Les résultats de la projection ont été placés et routés par l'outil NEOCAD AT&T. Les colonnes "Pfus" donnent le nombre de Cellules AT&T ORCA et "Prof" la profondeur maximale en nombre de cellules.

Benchs	ROTH&KARP				ROBDDs			
	OCC		PDS		OCC		PDS	
	Pfus	Prof	Pfus	Prof	Pfus	Prof	Pfus	Prof
5xp1	7	5	7	4	12	5	10	5
9sym	4	5	4	5	5	6	5	6
alu2	33	8	27	8	40	8	39	8
clip	17	6	14	7	43	7	11	7
count	19	16	19	16	41	12	16	12
cse	42	9	32	9	31	7	27	8
misex1	7	6	7	6	6	3	6	3
rd73	4	4	4	4	5	4	5	4
rd84	4	4	4	4	6	5	6	5
sao2	16	7	15	7	26	8	17	8
z4ml	4	3	2	4	6	4	4	4

La méthode permet une meilleure projection technologique dans une faible majorité de cas parmis ces benchmarks, mais dégrade parfois les resultats. Cette détèrioration est due à l'ajout des variables auxiliaires qui modifient la structure du ROBDD et peuvent augmenter le nombre de noeuds réels. On remarque aussi que la méthode d'ordonnancement fondée sur le poids des variables donne un meilleur résultat. Néanmoins on doit noter que la méthode d'ordonnancement a moins d'influence lors de la décomposition optimisée de Roth&Karp que lors de la décomposition directe sur le ROBDD.

IV-3 Méthode de recouvrement d'hypergraphe

Nous avons vu que le problème de décomposition technologique orienté surface est un problème exponentiel. De nombreuses méthodes utilisent des heuristiques pour contourner la complexité du problème. L'algorithme qui sera proposé ici est de complexité exponentielle. Contrairement aux approches heuristiques, nous nous attacherons à réduire la taille du problème grâce à une série de simplifications. Le résultat ne sera donc pas la solution optimale du problème général.

La résolution du problème général conduit à la minimisation sous contraintes d'une fonction linéaire en nombres entiers (Integer Linear Problem) [48]. La méthode du simplexe ou le Branch&Bound [49] peuvent servir à résoudre ces problèmes. Néanmoins le nombre élevé de contraintes linéaires du système rendent la résolution difficile et ces algorithmes inefficaces. La modélisation proposée ici permettra l'utilisation d'un algorithme général (méthode Proximale) moins sensible aux contraintes du systèmes que les autres algorithmes de résolution d'ILP.

La méthode sera ici appliquée à la technologie-cible XC5200 décrite précédemment.

IV-3-1 Définitions

- **Définition 32: Hypergraphe**

Soit $X=\{x_1,x_2,...,x_n\}$ un ensemble fini, on nomme hypergraphe sur X une famille $\{E_1,E_2,...,E_m\}$ de partie de X avec:

$$\begin{cases} E_i \neq \varnothing, \forall i / 1 \leq i \leq m \\ \bigcup_{i=1}^{m} E_i = X \end{cases}$$

- **Définition 33: Degré**

Soit un hypergraphe $H=(X,E)$, on appelle degré de $x \in X$ le nombre d'arêtes contenant x, on le note $d_H(x)$.
Le degré maximal de l'hypergraphe sera noté: $\Delta(H)=\max_{x\in X}(d_H(x))$.

- **Définition 34: Recouvrement**

Un recouvrement de H est une famille d'arêtes qui couvre tous les sommets de H, c'est à dire l'hypergraphe: H_1 avec: $\min_{x\in X} d_{H_1}(x) \geq 1$.

IV-3-2 Modelisation en hypergraphe

On considère deux types de blocs couvrants:

- Le LUT4 - type 1
- Le couple de LUT4 liés par un multiplexeur (cf. chapitre I) - type2

La première étape va consister à rechercher toutes les parties de ROBDDs susceptibles d'être couvertes par l'un des deux types de blocs. Ces parties vont être des ensembles de noeuds du ROBDDs.

Considérons l'hypergraphe H=(X,E) tel que:

- X: l'ensemble des sommets de l'hypergraphe. Chaque sommet correspond à un noeud du ROBDD.
- E: Les arcs de l'hypergraphe. Chaque arc est un ensemble de sommets dont les noeuds correspondant du ROBDD peuvent être couverts par un bloc de type 1 ou 2.

Le problème va donc consister à trouver un recouvrement de coût minimal (en terme de surface) dans l'hypergraphe défini ci-dessus.

On ne prendra pas en compte tous les arcs possibles. Premièrement, pour la base du ROBDD, on réalisera une projection technologique gloutonne en couvrant tous les noeuds de profondeur quatre (et le sous-ROBDD dont ils sont la racine) par un LUT4. La seconde restriction imposée le sera au niveau des sorties des noeuds couverts par un bloc de type 1 ou 2. On ne prendra pas les parties de ROBDDs dont un des noeuds a un père qui n'est pas compris dans l'arc; hormis bien sur, si c'est une sortie du bloc couvrant permet de lier ce noeud à ses pères extérieurs au bloc (cf. figure IV-12).

On ne prendra pas non plus en compte des arcs constitués d'un seul noeud. En effet, il est trivial de constater qu'un noeud isolé peut être couvert par un LUT4 (bloc de type 1).

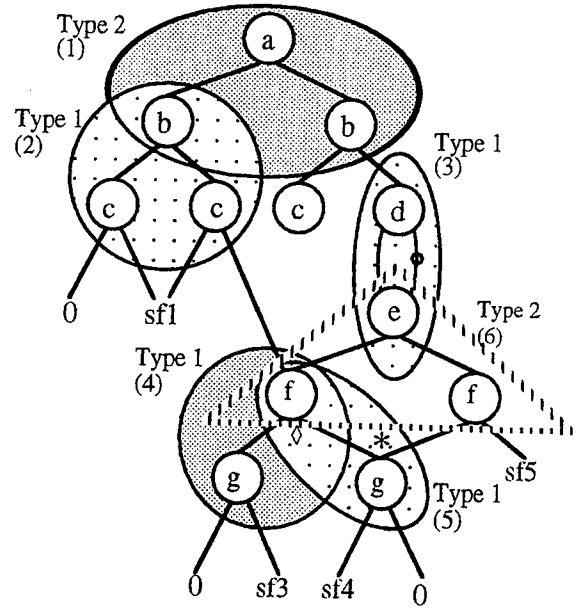

figure IV-12 Construction des arcs de l'hypergraphe.

L'arc 5 de la figure IV-12 ne sera pas sélectionné. cette partie de ROBDD peut être couverte par un LUT4, car un des pères du noeud marqué (*) n'est pas présent dans l'arc. Si cet arc était choisi, la fonctionnalité de ce noeud ne serait plus directement accessible, on serait amener à le dupliquer, afin de pouvoir exprimer la fonctionnalité du père non inclu dans l'arc. L'arc 6 de Type 1 peut en revanche être sélectionné car le noeud marqué (◊) sera couvert par un LUT4 . Il existe donc une sortie pour le père extérieur au bloc.

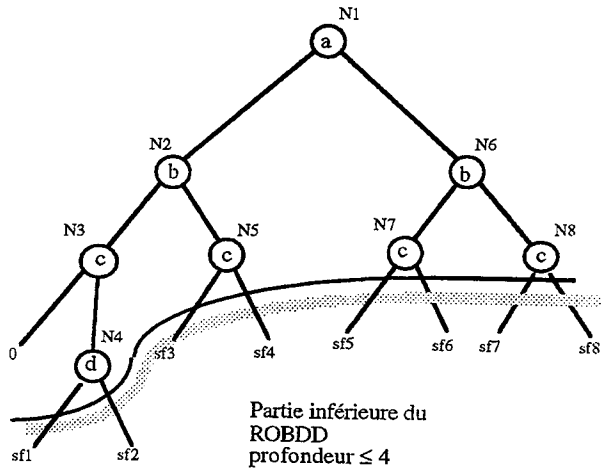

figure IV-13: ROBDD de départ.

Le ROBDD sur lequel on travaille est tronqué au niveau 4 (Figure IV-13) et la partie inférieure est directement mappée grâce à des LUT4. C'est sur ce graphe tronqué que va être calculé l'ensemble des arcs de l'hypergraphe comme décrit figure IV-14.

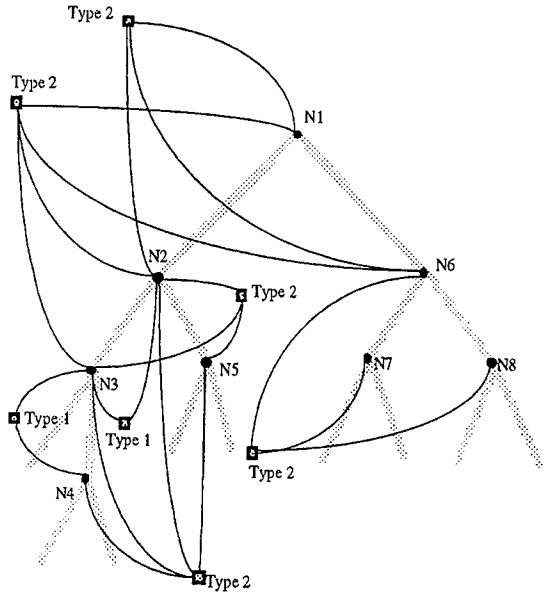

figure IV-14: Calcul des arcs.

IV-3-3 Modelisation en ILP

Pour résoudre le problème de recouvrement de l'hypergraphe, on va le reformuler en terme de problème de minimisation en nombre entier. Puis nous verrons quel algorithme a été choisi pour le résoudre, et les raisons qui ont motivé ce choix.

Le coût de chaque bloc couvrant se déduit simplement de la configuration décrite figure IV-12. Lorsqu'est sélectionnée la configuration LUT4 (type 2), on utilise $\frac{1}{4}$ de la cellule XC4000, et pour la configuration composée de deux LUT4 plus un multiplexeur, on utilise $\frac{1}{2}$ de la cellule. Ces deux valeurs vont constituer les coûts associés à chacun des deux types d'arcs.

Les coûts P_i prendront pour valeur:

$$P_i = \begin{cases} \frac{1}{4}, \text{ pour les arcs de type 1} \\ \frac{1}{2}, \text{ pour les arcs de type 2} \end{cases}$$

Nous avons vu que nous ne prenions pas en compte les arcs de type 1 qui ne sont constitués que d'un seul noeud. Néanmoins, ces cas de figure doivent être pris en compte au niveau du programme linéaire. On va donc chercher à minimiser le coût de couverture total, en tenant compte du fait qu'un sommet resté isolé doit être couvert par un arc de type 1.

Variables de l'ILP:

On va prendre trois types de variable:

- y_i sera associée à chaque arc;
- b_j associée à chaque noeud;
- c_{ij} associé à un couple arc/noeud.

Les variables prennent suivant les cas les valeurs suivantes:

$$y_i = \begin{cases} 1 \text{ si l'arc } i \text{ est utilisé} \\ 0 \text{ sinon} \end{cases}$$

$$b_j = \begin{cases} 1 \text{ si le sommet } i \text{ est couvert} \\ 0 \text{ sinon} \end{cases}$$

$$c_{ij} = \begin{cases} 1 \text{ si l'arc } j \text{ couvre le sommet } i \\ 0 \text{ sinon} \end{cases}$$

On notera $Nb(i)$ le nombre de sommets couverts par l'arc i. Cette constante associée a chaque arc est donnée lors de la phase de construction de l'hypergraphe.

Fonction coût:

Le coût total sera donc donné par la somme des coûts associés à l'utilisation des arcs, plus le coût associé à la couverture des noeuds isolés. On a donc la fonction:

$$F = \sum_j y_j . P_j + \sum_i b_i . P_i$$

Contraintes de l'ILP:

- Un noeud est soit isolé, soit couvert par un arc, donc:

$$\sum_j c_{ij} + b_i = 1, \forall i$$

- Tous les sommets couverts par le même arc ont le même coefficient: $c_{ij} = c_{kj}, \forall i, j, k$

- Si un arc est utilisé, il couvre tous les sommets inclus dans cet arc: $\sum_i c_{ij} = Nb(j).y_j, \forall j$

Le problème linéaire en nombres entiers peut donc se mettre sous la forme:

$$\begin{cases} Min\left(\sum_j y_j.P_j + \sum_i b_i.P_i\right) \\ \sum_i c_{ij} = Nb(j).y_j, \forall j \\ \sum_j c_{ij} + b_i = 1, \forall i \\ c_{ij} = c_{kj}, \forall i, j, k \\ 0 \le y_j \le 1, \forall j \quad 0 \le b_i \le 1, \forall i \\ 0 \le c_{ij} \le 1, \forall i, j \end{cases}$$

De la seconde contrainte, nous pouvons déduire l'égalité: $y_j = \frac{\sum_i c_{ij}}{Nb(j)}, \forall j$

Posons: $v_i = 1 - b_i$, après ce changement de variable et le remplacement de y_j dans la fonction coût, nous obtenons le problème:

$$\begin{cases} Min\left(\sum_j \frac{\sum_i c_{ij}}{Nb(j)}.P_j + \sum_i b_i.P_i\right) \\ c_{ij} = c_{kj}, \forall i, j, k \\ 0 \le y_j \le 1, \forall j \quad 0 \le b_i \le 1, \forall i \\ 0 \le c_{ij} \le 1, \forall i, j \end{cases}$$

dans le sous-espace vectoriel A défini par: $\sum_j c_{ij} - v_i = 0;$.

Les contraintes d'égalité du problème se traitent aisément au niveau de la minimisation. Les contraintes de couplage vont être considèrées comme un espace vectoriel dans lequel la solution doit se trouver.

Le problème en nombres entiers peut donc se mettre sous la forme $\begin{cases} Min & F(x) \\ x \in \mathrm{A} \end{cases}$, ou $F(x)$ est une fonction liptsichtzienne et convexe, et A un espace vectoriel. On considèrera pour la résolution du problème, l'équivalence suivante :

$$\begin{cases} Min & F(x) \\ X \in \mathrm{A} \end{cases} \Leftrightarrow \text{trouver } (x,y) \in \left(\mathrm{A}\ , \mathrm{A}^{\perp}\right) / \ y \in \delta F(x).$$

IV-3-4 La méthode proximale.

- **Définition 35: Graphe d'une fonction**

Soit T un opérateur maximal monotone défini sur $D \subset R^n$. On appelle graphe de T noté $Gr(T)$ l'ensemble de couples $Gr(T) = \{(x,y) \in D \times D / \ y \in T(x)\}$.

- **Définition 36: Opérateur monotone**

Un opérateur $T: D \to D^{\perp}$ est dit monotone si pour tout couple (x_i, y_i) et (x_j, y_j) de $Gr(T)$, on a : $\langle x_i - x_j, y_i - y_j \rangle \geq 0$

- **Définition 37: Maximal monotone [50]**

T est dit maximal monotone s'il est maximal dans la famille des opérateurs monotones de D dans $D^{\perp}$ ordonnés par relation d'inclusion.

- **Définition 38: Décompositions proximale [51]**

On dit que (x,y) est une décomposition proximale de z sur T si:

$$\begin{cases} (x,y) \in Gr(T) \\ x + y = z \end{cases}$$

Propriété:

Pour tout $z \in D$, la décomposition proximale (x,y) de z sur $Gr(T)$ est unique et on a:

$$\begin{cases} x \in (I+T)^{-1} z \\ x + y = z \end{cases}.$$

De plus, soit $D = \mathrm{A} \times \mathrm{A}^{\perp}$. On peut décomposer $z = x + y$ avec $x \in \mathrm{A}$ et $y \in \mathrm{A}^{\perp}$ de façon unique.

Propriété :

Pour tout $(x,y) \in \mathrm{A} \times \mathrm{A}^{\perp}$, il existe une décomposition proximale unique (u,v) de (x,y) sur $Gr(T)$ et on a:

$$\begin{cases} u \in (I+T)^{-1}(x+y) \\ u+v=x+y \end{cases}.$$

IV-3-4-1 Algorithme de décomposition proximale:

Soit $(x_k, y_k) \in \mathrm{A} \times \mathrm{A}^{\perp}$
Si $(x_k, y_k) \in Gr(\partial F)$
    arrêt: (x_k, y_k) est optimal.
Sinon
    Calculer la décomposition proximale (u_k, v_k) de $z = x_k + y_k$ sur $Gr(\partial F)$.
    Si $(u_k, v_k) \in A \times A^{\perp}$ arrêt: (u_k, v_k) est optimal.
    Sinon
        $x_{k+1} = proj_{\mathrm{A}}(u_k)$;
        $y_{k+1} = proj_{\mathrm{A}^{\perp}}(v_k)$.
$k = k+1$.

Où $proj_{\mathrm{A}^{\perp}}$ et $proj_{\mathrm{A}}$ désignent respectivement les projections sur les espaces vectoriels A et $\mathrm{A}^{\perp}$.

IV-3-4-2 Justification de l'algorithme

Notation:
Soit (u,v) la décomposition proximale de (x,y) sur $Gr(T)$.avec $x \in \mathrm{A}$ et $y \in \mathrm{A}^{\perp}$.
On notera P l'opération telle que:

$$(u,v) = P(x,y)$$

Propriété:
La suite (u,v) converge vers un point fixe [52].

Preuve:
On peut écrire l'étape Incrémentation sous la forme:
$(x_{k+1}, y_{k+1}) = O(x_k, y_k)$, où $O = proj_{\mathrm{A} \times \mathrm{A}^{\perp}} \circ P$.

Or O est la composition de deux opérateurs contractants, donc il est contractant. La suite engendrée par l'algorithme converge vers un point fixe.

Propriété :

Tout point fixe de l'algorithme est solution du problème.

Preuve:

Un point fixe de l'algorithme vérifie: $O = proj_{\mathbb{A} \times \mathbb{A}^{\perp}} \circ P$.

donc si (u,v) est la décomposition proximale de (x,y), on a;

$$\begin{cases} x = proj_{\mathbb{A}}(u) \\ y = proj_{\mathbb{A}^{\perp}}(v) \end{cases} \quad \text{donc} \quad \begin{cases} x - u \in \mathbb{A}^{\perp} \\ y - v \in \mathbb{A} \end{cases}.$$

Or par construction, nous avons: $u + v = x + y$

L'intersection de deux espaces orthogonaux étant le vecteur nul, on a:

$$x - u = y - v = 0 \quad \text{donc} \quad \begin{cases} x = u \\ y = v \end{cases}$$

On a donc bien trouvé un couple qui vérifie :

$(x,y) \in \left(\mathbb{A}, \mathbb{A}^{\perp}\right) / \; y \in \delta F(x)$

Nous procédons à une relaxation [53] des variables pour utiliser cet algorithme. Les variables sont ensuite serrées et l'algorithme relancé jusqu'à ce que nous obtenons un résultat en nombres entiers.

IV-3-4-3 Complexité de l'algorithme:

- La décomposition proximale: Soit (u,v) la décomposition proximale de (x,y) sur $Gr(\partial F)$. on a:

$$\begin{cases} (I + \partial F)u = x + y \\ u + v = x + y \end{cases} \text{ avec } \begin{cases} F = \sum_j \frac{\sum_i c_{ij}}{Nb(j)} . P_j + \sum_i b_i . P_i \\ c_{ij} = c_{kj}, \forall i, j, k \end{cases}.$$

Ce qui nous donne pour u les composantes:

$$\begin{cases} v_i(u) = v_i(x + y) + P_i \\ C_{ij}(u) = C_{ij}(x + y) - \frac{P_j}{Nb(j)} \end{cases}.$$

Et pour v les composantes:

$$\begin{cases} v_i(v) = -P_i \\ C_{ij}(v) = \frac{P_j}{Nb(j)} \end{cases}$$

- La projection sur A et $A^{\perp}$: Soit le vecteur $X = (x_1,...,x_n)$

La projection de X sur A est le vecteur $(x_1 - \alpha,...,x_{n-1} - \alpha, x_n + \alpha)$
Et la projection de X sur $A^{\perp}$ est le vecteur $(\alpha,...,\alpha,-\alpha)$où:

$$\alpha = \frac{(x_1+...+x_{n-1} - x_n)}{n}.$$

La projection comme la décomposition proximale ont un coût linéaire, ce qui nous garantit un temps de calcul raisonnable. Nous avons vu que dans le cas de l'algorithme glouton, on optimise le mapping en re-ordonnant les variables en fonction du résultat de la décomposition technologique, le nombre d'ordres pouvant aller jusqu'à n^2, où n désigne le nombre de variables de la fonction traitée. L'algorithme de mapping par recouvrement d'hypergraphe étant plus coûteux en temps de calcul, on se limite à 10 passages sur l'ordre des variables pour conserver un temps de calcul comparable avec celui de la méthode gloutonne.

IV-3-5 Résultats expérimentaux

Dans le tableau suivant, nous comparons les résultats obtenus par les deux méthodes de décompositions décrites ici. Les étapes précédant la phase de décompositions sont les mêmes dans les deux cas. Les tests sont réalisés sur des benchmarks internationaux MCNC et placés et routés à l'aide du logiciel de Xilinx PPR 5.0.

Benchmarks	Algorithme glouton		Recouvrement d'hypergraphe	
	#F gen. 4LUT	#M gen MUX	#F gen. 4LUT	#M gen MUX
z4ml	9	1	7	1
b9	26	4	26	7
misex1	14	3	16	5
count	37	0	45	1
alu2	77	15	77	20
5xp1	19	2	19	3
rd84	20	4	23	3
rd73	14	3	13	4
clip	30	6	28	8
bw	51	10	46	12
sao2	42	0	40	6
cse	72	9	69	13
vg2	41	7	40	6
9sym	16	2	16	2
keyb	103	12	95	15
duke2	198	21	188	23
jay	113	14	112	20
planet	201	40	196	30
s1	155	23	145	27
misex2	43	2	40	8
sand	165	21	161	33

Résultats comparatifs.

Les bons résultats obtenus ici sont dus à une augmentation du nombre de Multiplexeurs. En effet, la méthode de recouvrement prend en compte la configuration formée de deux LUT4 et un MUX, ce qui permet d'absorber plus de logique.

IV-4 Conclusion

Tout d'abord, les deux méthodes présentées ici possédent l'avantage d'avoir une vision globale de l'ensemble des equations booléennes à décomposer technologiquement. Neanmoins, cette vue d'ensemble augmente la complexité du problème à résoudre. Le second avantage et point commun de ces deux methodes est de permettre une décomposition technologique hétérogène. Dans les deux cas, la prise en compte de diverses configurations de la cellule de la technologie cible, ou même le changement de configurations possibles ne va induire que des faibles modifications de l'algorithme. La première methode fondée sur la décomposition de Roth&Karp ne donne que des résultats a priori decevant. La procédure d'unification rend l'algorithme difficilement utilisable pour des fonctions booléennes possédant

un nombre important de sorties. Il semble plus interressant d'utiliser la procédure de façon ponctuelle et locale, de manière gloutonne, lorsqu'est détecté une zone d'étranglement sur un ROBDD. La seconde methode basée sur le recouvrement d'hypergraphe est plus satisfaisante. Elle ne demande aucune modification du ROBDD, est directement applicable et insensible au nombre de sorties de la fonction booléenne.

V- Exploration de l'espace des solution lors de la génération de macro-blocs : Application aux additionneurs.

Les macro-blocs peuvent être classifiés en macros dures (hard macro), macros molles (soft macro) et en générateur paramétré.

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

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

Si l'usage des bibliothèques de base (cellules standards) est banal, l'extension à des bibliothèques de macroblocs [54] et à des générateurs de macro-blocs s'est accélérée. Afin de réduire le temps de synthèse des circuits complexes tout en obtenant les meilleurs résultats, on ne va utiliser que des parties déjà synthétisées et optimisées suivant les critères habituels (surface, vitesse). Ces parties pre-synthetisées appelées macros, sont stockées dans une bibliothèque et insérées dans les circuits suivant les besoins. Par exemple, des additionneurs ou des multiplieurs décrits au niveau de la saisie comportementale en langage de haut niveau (IVHDL) seront directement pris dans la bibliothèque sans même repasser par une phase de synthèse logique (figure V-1).

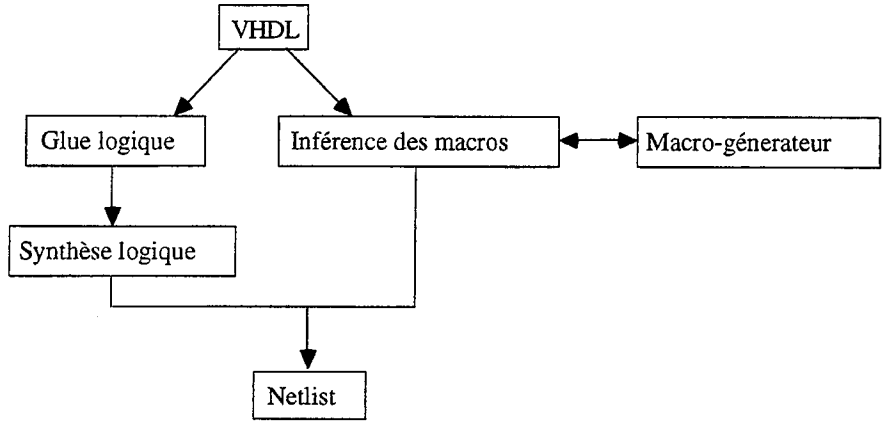

Figure V-1: Flot de conception.

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

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

La litterature est riche en publication traitant de la synthèse des addionneur. Sklanski proposa un algorithme [56] dont le principe est de générer deux ensembles de sortie et de retenues, un ensemble au cas ou la retenue est 1 et un second au cas ou la retenue est 0. Kelliher [57] proposa un addionneur de même structure mais avec un ensemble d'equations differentes. Wei et Thomson [58] modifièrent l'algorithme pour traiter le problème de la sortance qui croit exponentiellement dans l'additionneur de Slansky. Brent et Kung [59] proposèrent une structure qui limite la sortance mais qui est deux fois plus profonde que l'additionneur de Slansky. L'algorithme de Kogge et Stone [60] limite la sortance tout en optimisant la

profondeur de l'additionneur. Enfin, Han et Carlson [61] combinèrent les structures de Brent et Kung et de Kogge et Stone.

V-1 Equations de l'additionneur:

Le problème majeur de l'additionneur réside non pas dans le résultat bit à bit, mais dans la gestion et l'accélération du calcul de la retenue entrante. La retenue est par définition exprimée par des équations récurrentes. Celle-ci peut être exprimée et définie de différentes façons. Brent et Kung [59] [62] [63] [64] introduisirent en 1982 l'opérateur arithmétique Δ, destiné à modéliser son comportement quant à sa propagation et sa génération en fonction des variables d'entrée sortie.

V-1-1 Génération et Propagation

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

Au niveau d'un bit

A chaque position i, la retenue suivante peut être:

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

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

Au niveau d'une tranche de bits:

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

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

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

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

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

$Out\_0 = a_0 + b_0$

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

V-1-2 Définition de La Cellule Δ

On notera:

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

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

Définition d'une Δ-tranche

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

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

Définition d'un module de concaténation

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

Δ(i, j)& Δ(j+1, k)

le module ayant comme entrées les 2 Δ tranches et réalisant en sortie un ensemble de : Δ(i,k).

Le graphe associé à ce module contient trois noeuds; le noeud de la concaténation Δ(i, j, k) représenté par un triplet: les deux bornes séparés par l'indice du bit de concaténation. Les noeuds opérande sont les deux Δ tranches: Δ(i, j) et Δ(j, k) et le noeud sommet la Δ tranche:

Δ(i.k)=Δ(i, j)& Δ(j+1, k)

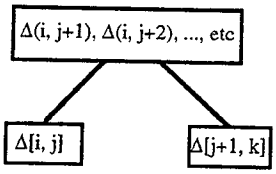

figure V-2:Noeud du graphe
réalisant un module de concaténation

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

V-2 Arbre de Coupe

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

Exemple:
Pour la figure ci-dessous fonctions associées aux cellules Δdans les intervalles de bit I1 = [0, 2], I2 = [3, 4], I3 = [5, 6], I4 = [7, 8], I5 = [9, 11], I6 = [12, 14], I7 = [15, 16] sont calculés en série. Les intervalles I1 et I2 sont calculés en parallèle puis composés à l'aide du module de concaténation C1 (C1 = I1 Δ I2); C2 = I3 Δ I4; C3 = I6 Δ I7; C4 = C1 Δ C2; C5=I5 Δ C4; C6 = C5 Δ C3. Ces opérations de sérialisation/parallélisation est représentées dans le graphe ci-dessous.

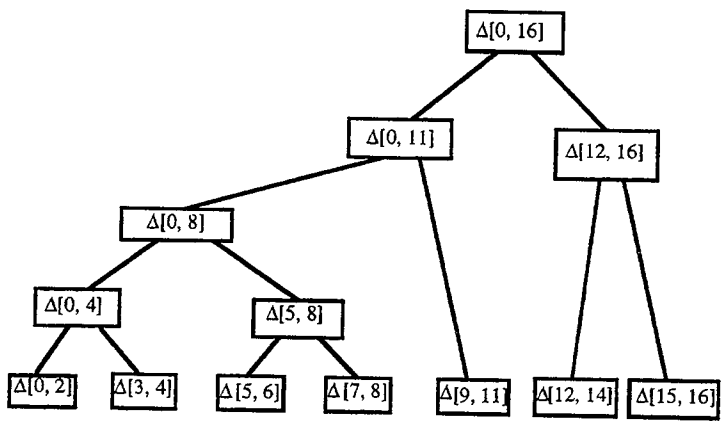

figure V-3: représentation d'un additionneur 16 bits

Ce graphe est appelé arbre de coupe.

On pourra ne représenter que les bits de coupe dans l'arbre dépourvu de ses feuilles [42]. Sachant que les bit contenus dans la partie gauche sont inférieurs aux bits de coupe, et ceux compris dans la partie gauche supérieurs, il est aisé de reconstituer les Δ tranches associées aux feuilles.

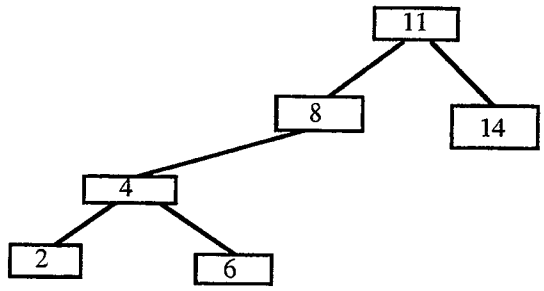

figure IIV-4: représentation minimale d'un additionneur 16 bits

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

(k,(A1)(A2))

où A1 représente le codage de l'arbre de coupe des bits inférieurs à k et A2 le codage de l'arbre de coupe des bits supérieurs à k.

Pour l'additionneur de 16 bit, le codage est:

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

V-3 Additionneurs classiques:

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

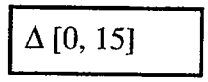

figure V-5: arbre de coupe
de l'additionneur 16 bits en ripple-carry

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

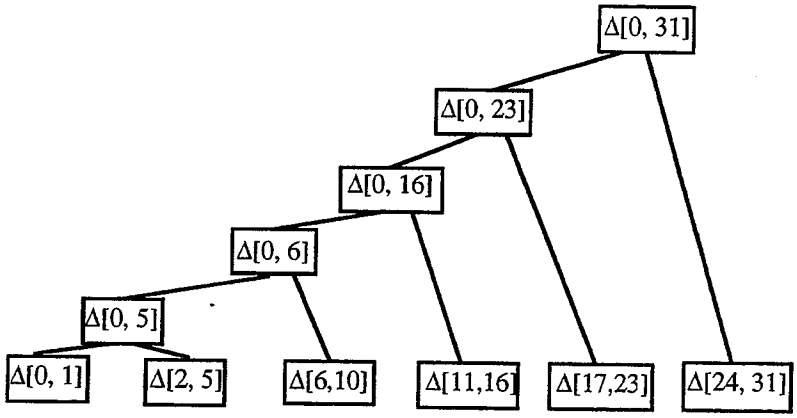

figure V-6: additionneur 32 bit en architecture carry-select

L'additionneur de Sklansky [56] est construit récursivement en parallélisant successivement les calculs par une division symétrique des blocs. C'est l'additionneur le moins profond en terme de cellules Δ, mais sa sortance croît exponentiellement le long du chemin critique. ce qui peut donner de mauvais résultats si cet additionneur est réalisé sur une technologie sensible à la sortance. Il demeure néanmoins l'additionneur le moins profond. Son code en LISP associé est: (16(24(20(18)(22))(28(26)(30)))(8(4(2)(6))(12(10)(14))))

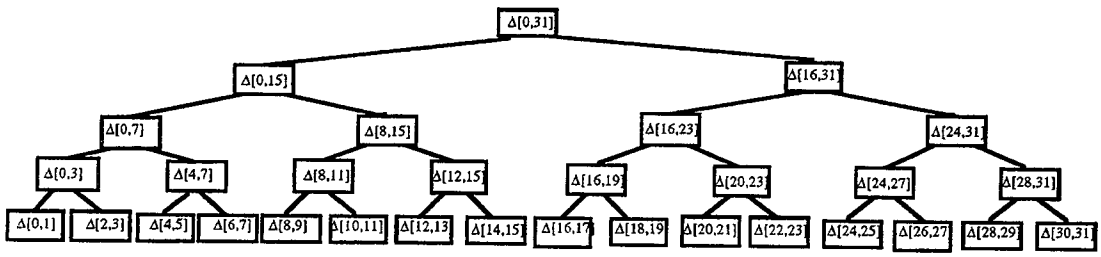

figure V-7: additionneur de Skansky

L'additionneur de Brent et Kung [59] est fondé sur la même structure que l'additionneur de Sklansky, mais la sortance est réduite grâce à une stérilisation locale des calculs. Cette stérilisation va provoquer une augmentation de la profondeur de l'additionneur.

V-4 Classification des additionneurs

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

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

On s'attachera ici a l'étude exclusive des additionneurs construits par inclusion.

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

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

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

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

V-5 Exploration de l'espace des solution

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

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

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

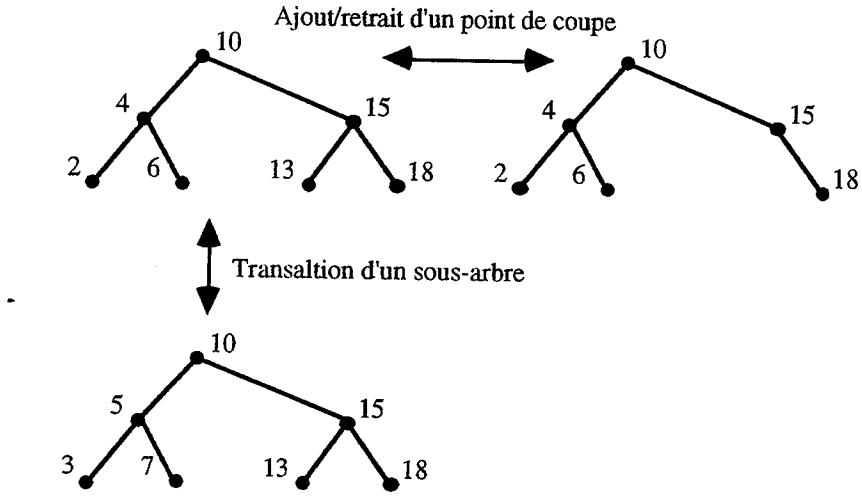

figure V-8: opérations élémentaires sur un arbre de coupe

V-5-1 Nuage d'additionneur

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

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

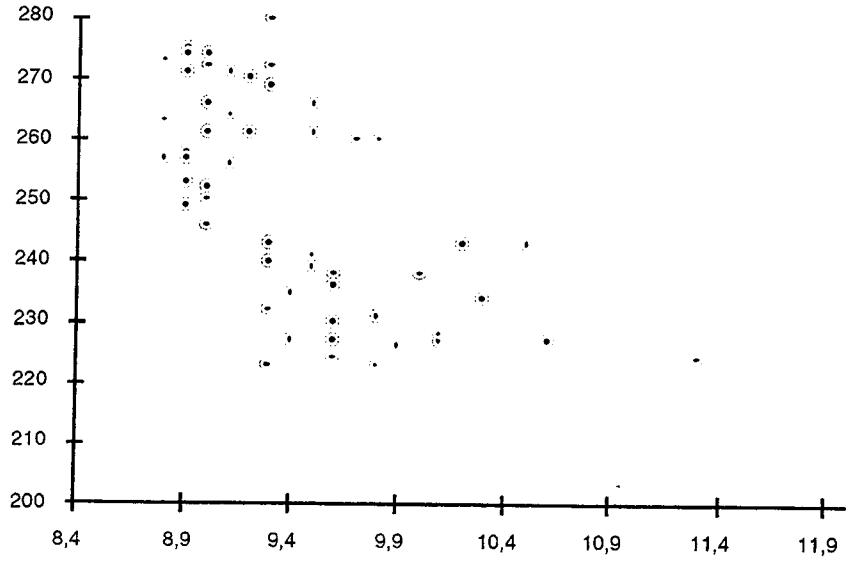

figure V-9: Nuage d'additionneur.

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

V-5-2 Filtrage de l'espace de solutions.

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

Définition d'une relation d'ordre:

La relation d'ordre partielle $\prec$ est définie sur les additionneurs par:
A1 $\prec$ A2 si et seulement si les trois conditions suivantes sont vérifiées:

- surf(A2) $\leq$ surf(A1)
- prof(A2) $\leq$ prof(A1)
- sort(A2) $\leq$ sort(A1)

Un additionneur A1 est dit moins performant qu'un additionneur A2 si et seulement si : A1 $\prec$ A2.

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

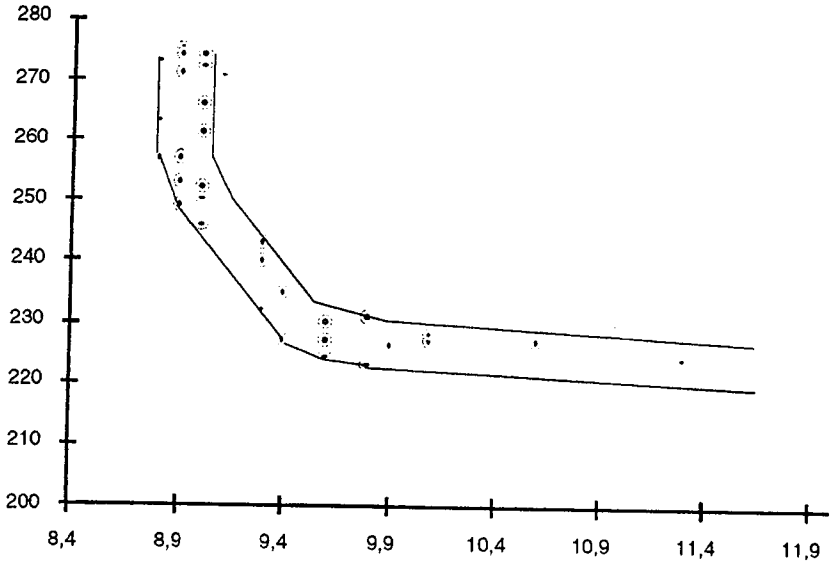

Figure V-10: Filtrage de l'espace de solution

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

V-5-3 Optimisation par dérivation d'une solution acceptable

A partir d'une solution acceptable A, on dérive l'arbre de coupe par l'application de la fonction M(A). La relation d'ordre partielle $\prec$ permet l'élimination des

additionneurs dont les caractéristique C(A) sont plus mauvaises que la caractéristique de l'additionneur de départ.

L'algorithme suivant est appliqué recursivement jusqu'à l'obtention d'une solution jugée meilleure. La qualité de la solution n'est pas seulement estimée par la caractéristique de l'additionneur, mais aussi après une phase de décomposition technologique. L'algorithme est basé sur une méthode tabou [65] qui permet de ne pas revenir sur des solution déjà testées. Sa complexité est exponentielle, mais l'usage du filtre dont le coût est linéaire et très faible rend utilisable l'algorithme.

Algorithme d'optimisation:

Soit A1 une solution acceptable
modification aléatoire de A1 : A2 = M(A1)
Si A2 $\prec$ A1
    éliminer A2
Sinon
    décomposition technologique de A2 $\rightarrow A2_{dec}$
    Si $A2_{dec}$ est meilleur que $A1_{dec}$ alors
        $A1 \leftarrow A2$
        $A1_{dec} \leftarrow A2_{dec}$
    Sinon
        éliminer A2

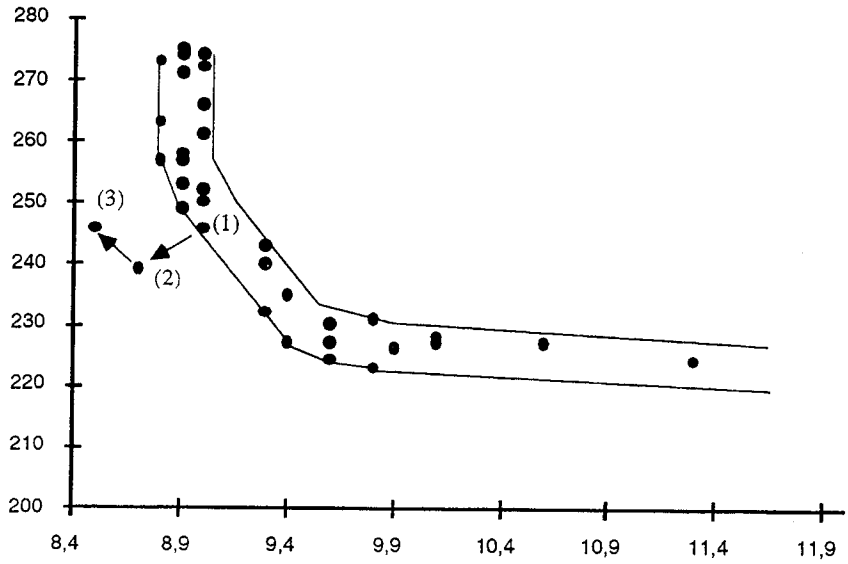

figure V-11: Exemple de procédure de dérivation

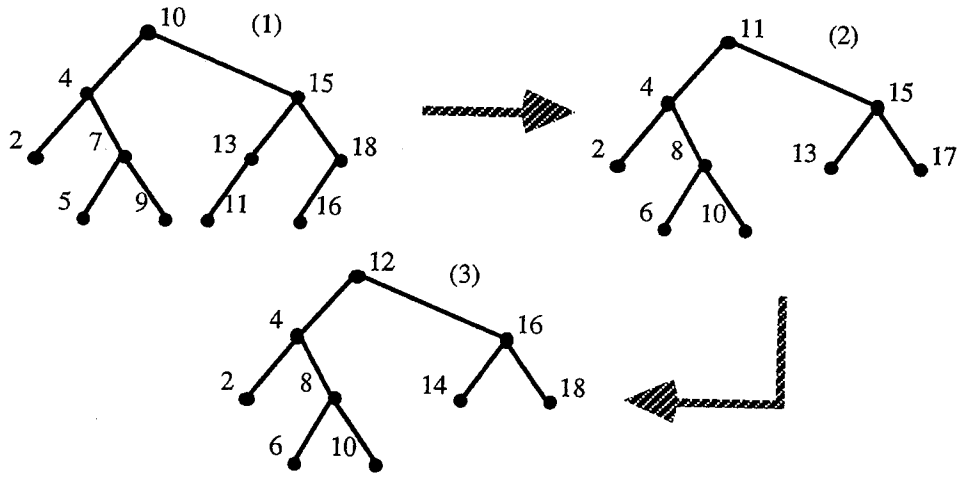

figure V-12: Succession d'arbre de coupe

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

C(A1)=(41,5,9), C(A2)=(40,5,8), et C(A3)=(40,5,8).

Les additionneurs correspondant possèdent les caractéristiques suivantes:

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

V-5-4 Un outil pour l'exploration de l'espace des solutions

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

L'outil comprends deux phases majeures:

- Génération d'un additionneur ou d'un ensemble d'additionneurs à partir d'un ou d'un ensemble d'arbres de coupe. Les additionneurs résultant sont synthétisés et estimés.

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

V-6 Résultats expérimentaux

Le tableau suivant présente un ensemble d'additionneurs obtenus à l'aide de l'algorithme décrit précedemment. Les Additionneurs sont technologiquement décomposés sur la bibliothéque de Cellules Standards vsc370. L'algorithme utilisé est basé sur la programmation dynamique et décrit dans [66]. La colonne "additionneur initial" donne le resultat en terme de surface (exprimée en μ^2) et le delai (exprimé en nano-secondes) de l'additionneur utilisé comme point de départ de la procedure de derivation; la seconde colonne donne les caracteristiques de l'additionneur resultant de la dérivation . Le critère principal de l'optimisation est la performance des additionneurs. Ainsi, après la phase de décomposition technologique, on ne retient que les additionneurs dont les performances ont été améliorées.

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

On remarque des dégradations de la surface. Le filtre utilisé ne permet pas d'évaluer très précisément le résultat final mais permet, comme nous le voyons sur ce tableau de contenir la viarition de surface qui n'exéde pas 7,7%.

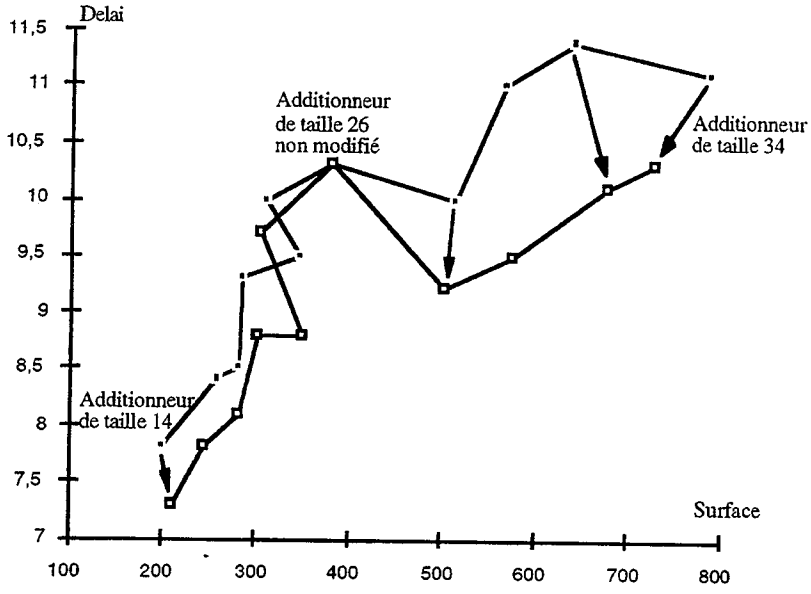

figure V-13: Résultat du processus de dérivation

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

VI- Conclusion

Cette thèse avait dans un premier temps pour objectif de développer et améliorer des méthodes de décomposition technologiques par des ROBDDs. Cette représentation qui a suscité un regain d'intérêt durant la dernière décennie, notamment pour les problèmes de vérification, est également un excellent support pour les problèmes de décomposition technologique. Le problème de cette représentation est une bonne maîtrise de sa complexité et ceci est lié au bon ordonnancement des variables. Il nous a donc semblé nécessaire dans cette thèse de faire le point sur cette représentation et sur les méthodes d'obtention d'un bon ordre.

La partie principale de cette thèse a consisté à proposer des méthodes de décomposition de la logique représentée par ces ROBDDs sur des technologies à base de LUT-k.

Les algorithmes de décomposition classiques sur ROBDDs (ou sur une représentation autre telle que les DAGs) pour les technologies-cibles FPGAs à base de LUT-k ne réalisent en fait qu'une décomposition sur un LUT-k, ou une configuration donnée du FPGAs. Les méthodes proposées prennent en compte un ensemble de motifs et de configurations possibles du FPGA et choisissent le motif le plus approprié à chaque pas d'optimisation.

A la lueur de cette expérience, on peut se poser plusieurs questions:

Les FPGAs des générations futures seront-ils des structures de plus en plus diversifiées?

On peut répondre que l'on assiste à une certaine stabilisation dans ce domaine quant aux marchés, aux fournisseurs de tels composants, et à la diversité de leurs structures. Il semble bien que trois architectures restent essentiellement en jeu:

Les cellules à base de LUT-k,

Les cellules à base de multiplexeurs,

Les cellules de type Cellules Standards.

Ceci semble lié aux principes même de fabrication.

Faut-il réellement optimiser de façon spécifique une décomposition technologique au cours d'un processus de synthèse?

On peut imaginer une décomposition initiale sur des librairies virtuelles puis un principe de réoptimisation spécifique à une cible donnée. Ce principe est utilisé dans de nombreux outils et il semble résoudre le problème de la diversité des flots d'optimisation.

A-t-on lieu d'espérer que la décomposition en LUT puisse faire l'objet de progrès importants?

Cette thèse montre que si l'on peut diversifier les approches, il faut avouer que les progrès sont modestes.

La dernière partie de cette thèse est justifiée par la constatation qu'un circuit est constitué de 80% de ,macros et seulement 20% de glue. Il est donc important de rééquilibrer les efforts d'optimisation. Afin de répondre aux nouvelles exigences en matière de macro-blocs, l'accent a été mis sur la génération sous contraintes. Pour offrir à un utilisateur potentiel le macro-bloc le plus proche possible des caracteristiques désirées, une première étape permet de parcourir l'ensemble des solutions, et une seconde étape permet d'optimiser une solution afin de se rapprocher des caractéristiques requises. Il faut remarquer que l'avenir sera consacré à des mega-fonctions ou "reuse blocks". Il est en effet dérisoire de penser que la granularité des blocs d'aujourd'hui sont les constituants de base de demain. Il est courant de stocker aujourd'hui des coeurs de microprocesseurs ou des blocs de télécommunication.

Nous pensons que cette thèse aura apporté une contribution au problème de la décomposition technologique pour les réseaux programmables à base de FPGAs ainsi qu'au problème de la macro-génération sous contraintes.

Bibliographie

[1] Actel Data Book ACTEL 1995.

[2] The Programmables Logic Data Book XILINX 1995.

[3] Advanced Data Sheet AT&T Microelectronics 1993.

[4] R. Brayton, R. Rudell, A. Wang, A. Sangiovanni-IVincentelli, "MIS : A multiple-level logic optimization system", IEEE CAD 1987.

[5] R. Brayton, G. Hachtel, C. Mac Mullen , A. Sangiovanni-IVincentelli, "Logic Minimisation algorithms for IVLSI synthesis", Boston, MA: Kluwer Academic, 1984

[6] R. Brayton, "New directions in logic synthesis", SASIMI 1990.

[7] C. Morrison, R. Jacobi, G. Hachtel "TECHMAP: Techonolgy mapping with delay and area optimization", Logic and Architecture Synthesis for Silicon Compilers, Elsevier Science Publishers B.IV. Noth Holland 1989.

[8] D. Muller, "Application of boolean algebra to switching circuit design and to error correction", IRE Trans. Electr. Comp. 1954.

[9] E. Lawler, "An approach to multi-level boolean minimization". Journal of ACM, 1964.

[10] D. Bryan, F. Brglez, R. Lisanke, "Redundancy identification and removal", IWLS, Research Triangle Park 1989.

[11] S.Akers, "Binary Decision Diagrams", IEEE Trans on Computers 1978.

[12] M.R.C.M. Berkelaar, J.A.G. Jess, "Technolgy mapping for standard cells generators", IEEE ICCAD 1988.

[13] R. Lisanke, "Technology mapping", IFIP Working Conference on Logic and Architecture Synthesis, Paris, 1990.

[14] S. Minato, "Fast generation of irredundant Sum-Of-Products forms from binary decisio diagrams", SASIMI 1992.

[15] B. Bollig, P. Savicky, I. Wegener, "On the improvement of variables ordering for OBDDs", IFIP Workshop 1994.

[16] J. Burch, D. Long, "Efficient boolean function matching", IEEE ICCAD 1992.

[17] M. Fujita, H. Fujisawa, N. Kawato, "Evaluation and improvement of boolean comparison method based on binary decision diagrams", IEEE ICCAD 1988.

[18] A. Malik, R. Brayton, A. Newton, A. Sangiovanni-IVincentelli, "A modified approach to two-level logic minimization", IEEE ICCAD 1988.

[19] R. Jacobi, A. Trullemans, "Generating prime and irredundant covers from binary decision diagrams", EDAC 1992.

[20] S. Jeong, B. Plessier, G. Hachtel, F. Somenzi, "IVariable ordering for binary decision diagrams", EDAC 1992.

[21] N. Calazans, Q. Zhang, R. Jacobi, B. Yernaux, A. Trullemans, "Improving binary decision diagram throught incremental reduction and improved heuristics", IEEE ICCAD 1992.

[22] D. Filo, J. Yang, F. Maillot, G De Micheli, "Technology mapping for a two output RAM-based FPGAs", EDAC 1991.

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

[24] R. Jacobi, P. Moceyunas, H. Cho, H. Hachtel, "New ATPG technics for logic optimization", IEEE ICCAD 1989.

[25] K. Karplus, "Representing boolean functions with If-Then-Esle DAGs",Internal report UCSC-CRL-88-28, 1988.

[26] T. Besson, H. Bouzouzou, I. Floricica, R. Roane, G. Saucier, "Synthesis on multiplexor-based FPGAs using Binary Decision Diagrams", ICCD USA 1992.

[27] T. Besson, H. Bouzouzou, G. Saucier, "Ordering for ROBDDs based on sharing analysis", SASIMI Japan 1993.

[28] O. Coudert, J. Madre, "Implicit and incremental computation of prime and essential implicants of boolean functions", DAC 1992.

[29] A. Malik, R. Brayton, A. Newton, A. Sangiovanni-IVincentelli, "A modified approach to two-level logic minimization", IEEE ICCAD 1988.

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

[31] R. Jacobi, P. Moceyunas, H. Cho, H. Hachtel, "New ATPG technics for logic optimization", IEEE ICCAD 1989.

[32] P. Abouzied,"Méthode de factorisation algébrique dédiée aux circuits intégrés complexes", Thèse de Doctorat, INPG, 1992.

[33] A H Ferrari, M. Sarrafzadeh, "Complexiy of the LUT minimization problem for FPGA technology mapping", Northwestern University Evaston 1994.

[34] M. Garey, D. Johnson, "Computers and Interractybility: a guide to the theory of the NP-Completeness", Freeman 1979.

[35] B. Babba, "Synthèse optimisée sur les réseaux programmables de type Xilinx", Thèse de Doctorat, INPG, 1995.

[36] R. Murgai, N. Shenov, R. Brayton, A. Sangiovanni-IVincentelli, "Improved logic synthesis algorithm for LUT Architectures", ICCAD 1991.

[37] IV. Le, "Optimisation temporelle des réseaux programmables à base de LUTs", Thèse de Doctorat, INPG 1996.

[38] P. Sawkar, D. Thomas, "Area and delay mapping for LUT based FPGAs", IEEE DAC 1992.

[39] IV. LE, "G. Saucier, M. Crastes, "Performance optimization for LUT-based FPGAs", EDTC 1996.

[40] G. Saucier, H. Bouzouzou, B. Babba, R. Roane, F. Poirot, "Use of Binary Decision Diagrams for Logic Synthesis", IEEE ICCAD 1993.

[41] G. Bosco, A. Amoura, M. Belrhiti, "An Efficient ILP Mapping algorithm for LUT-based FPGAs", IFIP Workshop 1993.

[42] G. Bosco, A. Amoura, "Recusrsive and optimized Roth&Karp decomposition", SASIMI 1996.

[43] J. Roth, J. Karp, "Minimization over boolean graphs", IBM Journal 1962.

[44] S. Chang, M. Marek-Sadowska, "Tehcnology mapping for LUT FPGAs based on Decomposition of binary decision diagrams", University of California, 1995.

[45] Y. Lai, M. Pedram, S. IVrudhula,"BDD based decomposition of logic functions with application to FPGA synthesis" IEEE DAC 1993.

[46] C. Scholl, P. Mollitor, "Efficient ROBDD based computation of common decomposition of multi-output boolean functions", Dagstuhl Seminar, 1995.

[47] C. Berge, "Graphes", Gauthier-IVillard, Paris, 1983.

[48] M. Sakarovitch "Optimisation combinatoire", Herman, Ensignement des sciences, Paris, 1984.

[49] A. Shrijver "Theory of linear and integer programming", John Wiley&Sons UK, 1987.

[50] H. Brézis, "Opérateurs maximaux monotones", Mathematics Studies, Nrth Holland, 1973.

[51] J. Moreau, "Proximité et dualité dans un espace Hilbertien", Bull Soc. Math. de France 1993.

[52] P. Mahey, "Proximal decomposition on the graph of a maximal monotone operator", Research report ARTEMIS Grenoble 1992.

[53] D.Bertsekas, "Relaxation Methods for network flow problems with convex arc costs", Siam J Control of Optimization, 1987.

[54] O Khalil, "Utilisation des bibliothèques de macro-blocks dans la synthèse de haut niveau et dans la migration technologique", Thèse 1995.

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

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

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

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

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

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

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

[62] A. Guyot "Computer arithmetic", Cours ENSIMAG 1995.

[63] M. Belrhiti, G. Bosco, A. Guyot, "Efficient generator of adders", IWLS Tahoe City 1995.

[64] G. Kinke, "Optimisation combinatoire", Cours ENSIMAG 1992.

[65] K. Sakouti, "Synthese et optimisation de réseaux booléens", Thèse de Doctorat, INPG 1993.

Table des matières

Résumé

Cette thèse s'intéresse d'une part au problème de décomposition technologique orienté surface sur des réseaux programmables de type FPGAs (Field Programmable Gate Arrays) et d'autre part à la synthèse des macro-générateurs sur ASICs et plus précisément de la synthèse des additionneurs.
La décomposition s'articule autour de deux axes essentiels: tout d'abord, il s'agit d'optimiser la taille de la représentation des fonctions booléennes. Les représentations de base choisies ici sont les ROBDDs (Reduced Ordered Binary Decision Diagrams) ainsi qu'une structure dérivée, les ITE (If Then Else). La deuxième étape concerne la décomposition proprement dite. Les technologies cibles sont ici des FPGAs à base de LUT-k (Look Up Table), en particulier les FPGAs XC5200 de Xilinx et Orca de AT&T.
Les deux méthodes de décomposition technologique orienté surface proposées permettent une décomposition hétérogène en prenant en compte non pas une seule configuration mais un ensemble de configurations possibles de la cellule cible. la première méthode est fondée sur un parcours descendant et optimisé du ROBDD. La seconde méthode s'appuie sur une modélisation en recouvrement d'hypergraphe du problème de décomposition technologique. Dans les deux méthodes, le coût exact en terme de surface finale du circuit est pris en compte à chaque étape de la décomposition.
L'étude menée dan la deuxieme partie de la thèse sur la macro-génération conduit dans un premier temps à l'exploration de l'espace des solutions possibles puis à l'optimisation d'une solution sélectionnée par un algorithme de dérivation discrète. L'utilisation d'un filtre permet la restriction de l'espace des solutions à explorer et d'autre part guide le processus de dérivation en éliminant les solutions trivialement médiocres. La combinaison des processus d'exploration et de dérivations permet la génération de macros dont les caractéristiques physiques sont les plus proches possibles de celles fixées par un utilisateur potentiel.

Ces méthodes ont été intégrées au sein d'un outil universitaire ASYL+ développé au laboratoire CSI.