Optimisation temporelle des réseaux programmables à base de LUT

van Viet Le

Thèse

Présentée par

Van Viet LE

Pour obtenir le grade de

Docteur de l'Institut National Polytechnique de Grenoble

(arrêté ministériel du 23 Novembre 1988)

Spécialité Microélectronique

Optimisation temporelle des réseaux programmables à base de LUT

Date de soutenance: 5 Janvier 1996

Composition du Jury:

Mesdames	Profeseur A-M TRULLEMANS	Rapporteur
	Professeur G. SAUCIER	
Messieurs	Professeur G. MAZARE	Président
	Professeur D. AUVERGNE	Rapporteur
	A. GUYOT	

Thèse préparée au sein du Laboratoire Conception de Systèmes Intégrés à l'INPG

TABLE DE MATIERES

Remerciements

Je tiens tout d'abord à remercier Madame Gabrièle SAUCIER, Professeur à l'ENSIMAG et directrice du Laboratoire Conception de Systèmes Intégrés, pour m'avoir accueilli dans son laboratoire, pour avoir dirigé mes recherches, pour le temps qu'elle m'a consacré, ainsi que le soin qu'elle a porté à la lecture et à la rédaction de ma thèse.

Je remercie également:

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

Monsieur Daniel AUVERGNE, Professeur à l'Université de Montpellier, d'avoir accepté d'être rapporteur de cette thèse.

Madame Anne-Marie TRULLEMANS-ANCKAERT, Professeur à l'Université de Louvain, Belgique, d'avoir accepté d'être rapporteur de cette thèse.

Monsieur Alain GUYOT, Habilitation à Diriger des Recherches au TIMA, ENSIMAG, d'avoir accepté de faire partie de mon jury.

Je remercie également le gouvernement français de m'avoir donné une bourse afin de préparer cette thèse et le CROUS de Grenoble pour les falicités qu'il m'a accordées lors de mon séjour en France.

Je remercie le Ministère de l'Education et de la Formation du Viet Nam et l'Institut National Polytechnique d'Ho Chi Minh Ville de m'avoir autorisé à aller en France pour préparer cette thèse.

Je remercie chaleureusement tous mes collègues du CSI pour l'agréable ambiance de travail, plus particulièrement Ion FLORICICA, Vincent SORNETTE, Philippe BRAHIC, Xavier WENDLING, Lionel REVERET, Raphael ROCHET, et tous ceux qui, de près ou de loin, ont contribué à l'élaboration de ce travail.

Je ne voudrais pas oublier Pascale PINEL et Martine DELMAS pour leur agréable accueil à chaque visite au secrétariat et pour leur aide.

Résumé

Cette thèse a comme objectif l'optimisation temporelle au cours du processus de synthèse des réseaux programmables à base de LUT, en particulier ceux de la famille Flex8000 d'Altera et de la famille XC4000 de Xilinx. L'optimisation temporelle consiste en deux étapes essentielles: la détection de la zone critique et la décomposition technologique orientée vitesse dans cette zone.

Les nouvelles notions de zone sensible et de zone critique sont utilisées pour rechercher la zone dont l'optimisation temporelle devrait satisfaire les demandes de l'utilisateur.

Un modèle prédictif réaliste du délai des chemins dans un réseau à base de LUT a été proposé. La fonction de coût proposée, exprimant bien la relation entre les entrées/sorties critiques, non critiques ou sensibles, permet à la décomposition technologique orientée vitesse d'optimiser efficacement la zone nécessaire, en augmentant très peu la surface.

De plus, les réinjections contrôlées des sous-fonctions booléennes appliquées dans la zone critique permettent d'obtenir des zones de travail, destinées à une resynthèse, avec une taille convenant à la technologie considérée.

L'approche proposée pour la décomposition technologique orientée vitesse en tenant compte des contraintes de l'utilisateur s'est avérée rapide et donne de bons résultats. Le gain en vitesse de l'optimisation temporelle par rapport à l'option surface est en moyenne de 32 % avec une augmentation de surface de 30% seulement.

Ce travail a fait l'objet d'un transfert technologique vers un système industriel de CAO, appelé ASYL+. La qualité des résultats a été démontrée en comparaison avec des outils commerciaux tels que Exemplar, Maxplus2 de la société Altera et XACT de Xilinx. Les tests sur de nombreux benchmarks donnent un gain moyen en performance temporelle d'ASYL+ sur Exemplar, Maxplus2 et XACT respectivement de l'ordre de 28%, 18% et 17%.

Mots clés:

Réseaux programmables, Factorisation algébrique, Réinjection, Synthèse logique, Modèle de délai, Détection de la zone critique, Décomposition technologique.

Abstract

This thesis proposes a performance optimization method for LUT based programmmable devices, especially the Flex8000 family from Altera and the XC4000 family from Xilinx. The method consists of two main steps: detection of the critical region and speed oriented mapping in the critical region.

The new notions of sensitive region and critical region are used to find the region whose performance optimization has to satisfy users' requirements.

A realistic predictable delay model for LUT-based programmable devices is proposed. The cost function that reflects correctly the relation between the critical, non-critical and sensitive inputs/outputs, allows the speed oriented mapping to optimize significantly the necessary region for only a limited area overhead.

In addition, the collapsing techniques for boolean sub-functions, applied only to the critical region, allow to obtain working regions, on which resynthesis will be performed, having a size suitable to a given technology.

The proposed method for speed oriented mapping taking into account users' constraints leads to quick and good results. The gain in terms of speed of the performance optimization compared to the area oriented optimization is about 32% on average in increasing only 30% its area.

This work is integrated in the industrial system, named ASYL+. The quality of the results is proved by comparisons with other commercial systems such as Exemplar, Maxplus2 and XACT. The tests on many benchmarks give an average performance gain with respect to Exemplar, Maxplus2 and XACT respectively of about 28%, 18% and 17%.

Key words:

Programmable Devices, Algebraic Factorization, Collapsing Techniques, Logic Synthesis, Delay Model, Detection of the Critical Region, Technology Mapping

INTRODUCTION GENERALE

Depuis une décade, la synthèse logique est un outil clef dans la conception des circuits et systèmes intégrés. Les premiers résultats spectaculaires ont concerné les capacités d'optimisation de circuits représentés par leur réseaux de portes. En particulier, un circuit conçu à la main peut être optimisé en un temps court avec une efficacité nettement supérieure aux possibilités humaines. Ces progrès ont d'abord été appliquées aux circuits intégrés à la demande (ASIC utilisant des cellules standards ou des gate arrays) puis plus récemment aux réseaux programmables.

Les réseaux programmables constituent aujourd'hui une part de marché très importante et en croissance constante. Les réseaux à forte granularité (PAL ou CPLD) ont vu leur part de marché rognée par les réseaux à faible granularité rappelant de près ou de loin les structures des gate arrays d'où leur nom de FPGA (Field Programmable Gate Array). Les réseaux programmables ont à leur tour bénéficié des progrès de la synthèse logique et les vendeurs de ces réseaux délivrent aux clients des solutions de synthèse clef en main adaptées à leur composant. Ces outils partant initiallement de schématique, puis de langages conçus par ces vendeurs (Palasm, ABEL) partent aujourd'hui de langages tels que VHDL, Verilog.

Cette thèse s'intéresse aux réseaux programmables utilisant comme cellule de base des cellules implémentant n'importe quelle fonction booléenne de k variables spécifiée par son tableau de valeur stocké dans une RAM. Une telle cellule est appelée LUT (LookUp Table). Le problème le plus critique dans la synthèse est l'optimisation temporelle. En effet les réseaux programmables sont des dispositifs relativement lents et les utilisateurs souhaitent obtenir les meilleures performances du composant. Le problème traité ici est l'optimisation temporelle de circuits réalisés sur une structure à base de LUT.

Les buts de l'optimisation temporelle sont la minimisation des chemins les plus longs dans une zone définie par lui. Les zones sur lesquelles les techniques d'optimisation sont appliquées sont des zone combinatoires.

Le première chapitre fait un tour d'horizon des réseaux programmables en insistant sur les réseaux à base de LUT et sur les charactéristiques ayant un impact sur la synthèse telles que les réseaux d'interconnexion.

Dans le deuxième chapitre, on rappellera les principales représentations booléennes connues en insistant sur celles utilisées dans ce travail à savoir les expressions factorisées. On considérera en particulier les représentations graphiques de ces formes factorisées en mentionnant des techniques jouant un rôle important dans la décomposition technologique ultérieure à savoir les techniques de "réinjection" de sous-fonctions visant à définir des zones de travail pour la décomposition technologique.

Le troisième chapitre qui est le cœur de la thèse est dédié aux techniques d'optimisation temporelle.

Dans une première partie on se focalisera sur le modèle de délai prédictif de cellules et de chemins de cellules destiné à guider la synthèse. Cette notion sera utilisée pour la détermination de la zone critique dont l'optimisation devrait permettre de satisfaire les vœux du concepteur.

La partie suivante aborde la décomposition proprement dite du réseau des cellules LUT en essayant de satisfaire ces contraintes d'optimisations. Cette décomposition est fondée sur une notion de coût hierarchique.

Cette décomposition technologique présentée de façon générale pour tout réseau LUT est ensuite personnalisée pour 2 familles à savoir les réseaux LUT d'Altera (famille Flex8000) et les réseaux LUT de Xilinx (famille XC4000).

De nombreuses expérimentation ont été faites validant cette approche par rapport à des outils du commerce (Exemplar de la société Exemplar Logic Inc, Maxplus2 de la société Altera et XACT de la société Xilinx)

CHAPITRE 1

CLASSIFICATIONS DES RESEAUX PROGRAMMABLES

INTRODUCTION

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

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

Outre ces qualités, ce sont récemment leur densité d'intégration et leur fréquence de fonctionnement sans cesse croissantes qui les placent en concurrents directs des réseaux prédiffusés. Dès leur apparition, les réseaux programmables comptaient environ 500 portes, ce qui leur a vite permis d'absorber puis de dépasser les composants réalisés en logique "MSI-SSI". Par la suite, la passage à des techniques nouvelles de programmation (EEPROM, antifusible PLICE/ViaLink, CMOS,...) les a conduits à des densités capables d'intégrer entre 1000 et 50.000 portes, et à temps de traversée inférieur à 5 ns. Dès lors, ils se trouvent en concurrence avec 80 % des réalisations en circuits intégrés classiques, en particulier celles qui requièrent moins de 20.000 portes et fonctionnent à des fréquences de moins de 30 MHz.

Ces progrès permettent de réaliser, à l'aide des réseaux programmables, des applications de complexité croissante, de plus en plus rapides, tout en conservant une grande souplesse en particulier la possibilité de reprogrammer un composant, donc de réaliser un nouveau circuit. sans remettre en question le routage d'une carte.

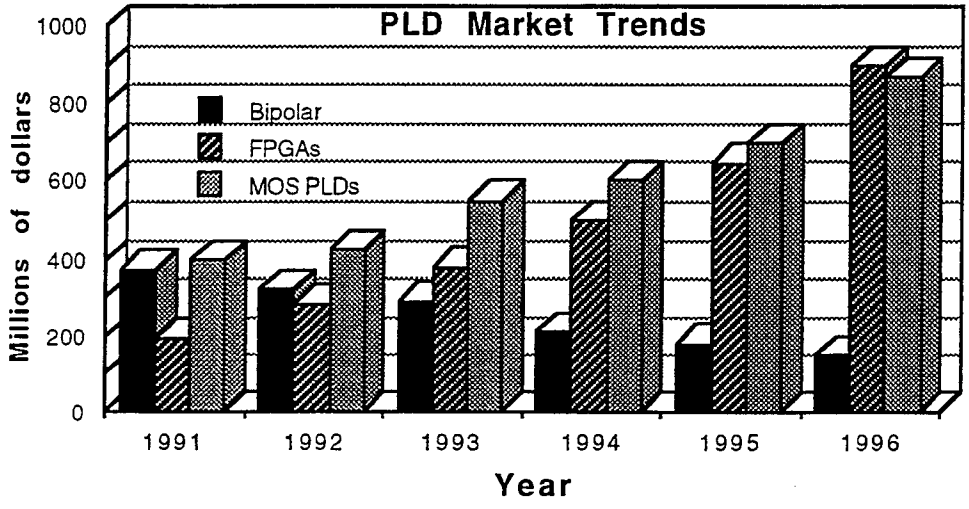


Figure 1.1 Croissance du marché des réseaux programmables

La figure 1.1 montre la croissance du marché des réseaux programmables [DAT92].
Pour des considérations de synthèse logique, on peut distinguer deux grandes familles de réseaux programmables suivant la primitive booléenne sousjacente à leur organisation topologique.

* Les réseaux à forte granularité organisés topologiquement en deux plans "ET" et "OU", implantant des fonctions Booléennes écrites sous forme de somme de monômes (les CPLDs: Complex Programmables Logic Devices)
* Les réseaux de cellules, offrant comme primitives des cellules proches des portes de base des "Gate Array" (les FPGAs: Field Programmable Gate Array).

Nous nous intéressons spécialement aux familles FPGA XC4000 de Xilinx et à la famille Flex8000 d'Altera pour lesquelles nous allons proposer des méthodes d'optimisation temporelle dans le chapitre 3.

I. CLASSIFICATION DES RESEAUX PROGRAMMABLES EN VUE DE LA SYNTHÈSE LOGIQUE

I.1. Réseaux à forte granularité (les CPLDs)

Cette famille de réseaux programmables est caractérisée par deux plans logiques. Le premier, appelé étage "ET" ou étage "Produit", réalise un "ET logique" entre les variables d'entrée pour former des monômes. Le second, appelé étage "OU" ou étage "Somme", réalise un "OU logique" entre les monômes pour former des fonctions booléennes. Soulignons que toute variable est présente à la fois sous sa forme normale ("a") et sous sa forme complémentée, (!a). A la sortie des deux plans, nous obtenons des fonctions booléennes du type suivant: $f = \sum m_i$ où m_i est un monôme.

Cette implantation des fonctions booléennes, sous forme de sommes de monômes, nécessite une minimisation pour chaque fonction, qui doit être globale si la structure du réseau autorise le partage des monômes.

La représentation la plus classique des réseaux à deux plans est illustrée par la figure 1.2

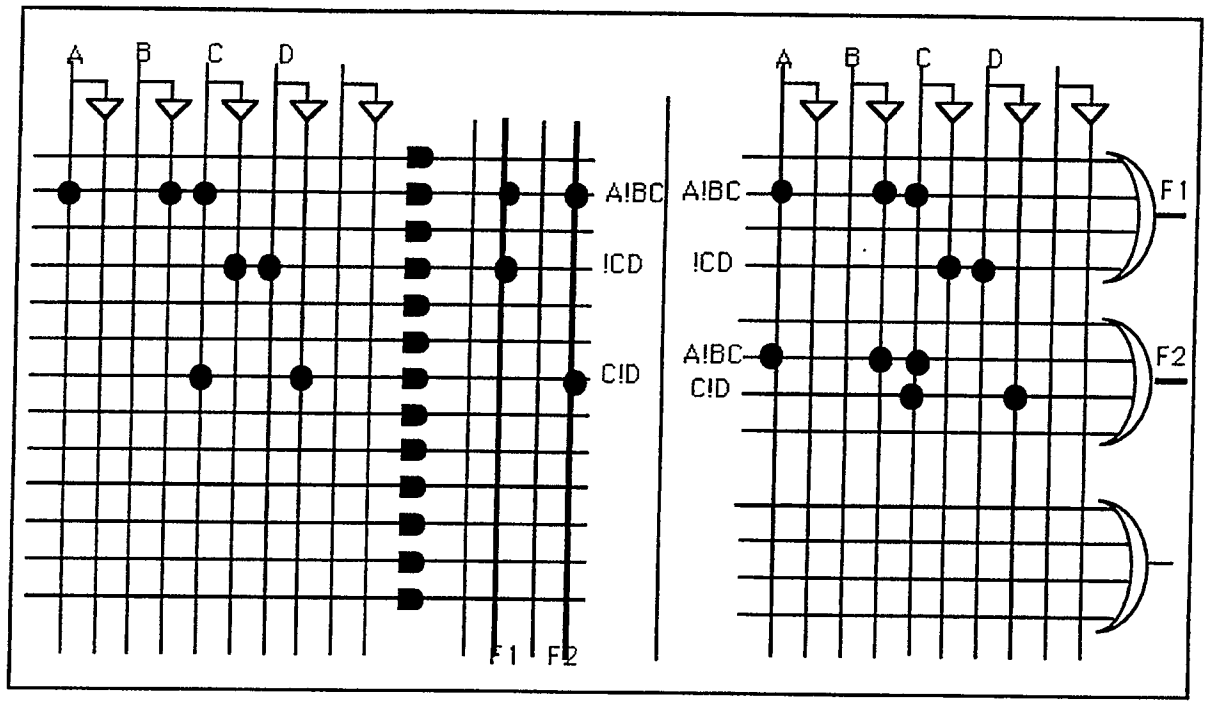


Figure 1.2 Réseau à deux plans

La programmation est réalisée à l'intersection des lignes verticales et horizontales de chaque plan, soit à l'aide de fusibles que l'on brûle dans le cas des réseaux non-reprogrammables, soit à base de cellules "EPROM", une par croisement, que l'on charge pour assurer le contact dans le cas des réseaux reprogrammables. Dans la figure 1.2, une connexion est caractérisée par un point "•", placé à l'intersection des lignes connectées. Dans le plan "ET", cela signifie que la variable associée à la ligne verticale est affectée au monôme correspondant à la ligne horizontale, et dans le plan "OU", que le monôme associé à la ligne horizontale est affecté à la sortie correspondant à la ligne verticale. On trouve parfois une croix "**X**" à la place du point. Elle signifie que l'intersection considérée peut être programmée par le concepteur, par opposition à celles qui ont déjà été programmées par le fabriquant.

La possibilité ou l'impossibilité, pour deux fonctions différentes, d'utiliser le même monôme, nous amène à distinguer:

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

I.1.1 Fonctions ne partageant pas les monômes

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

a) Réseaux monoblocs ne partageant pas les monômes

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

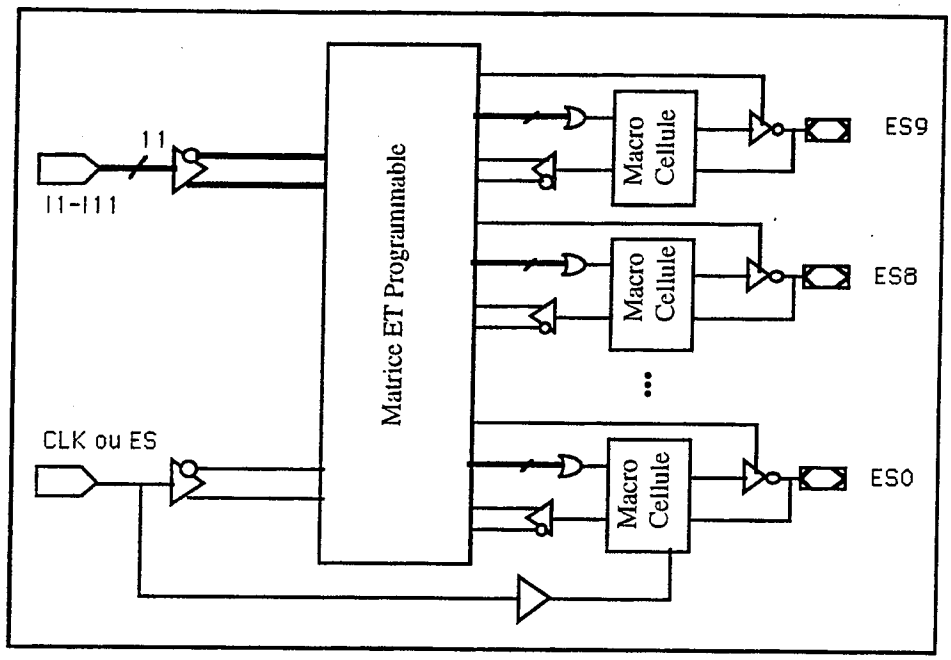


Figure 1.3 PAL 22V10

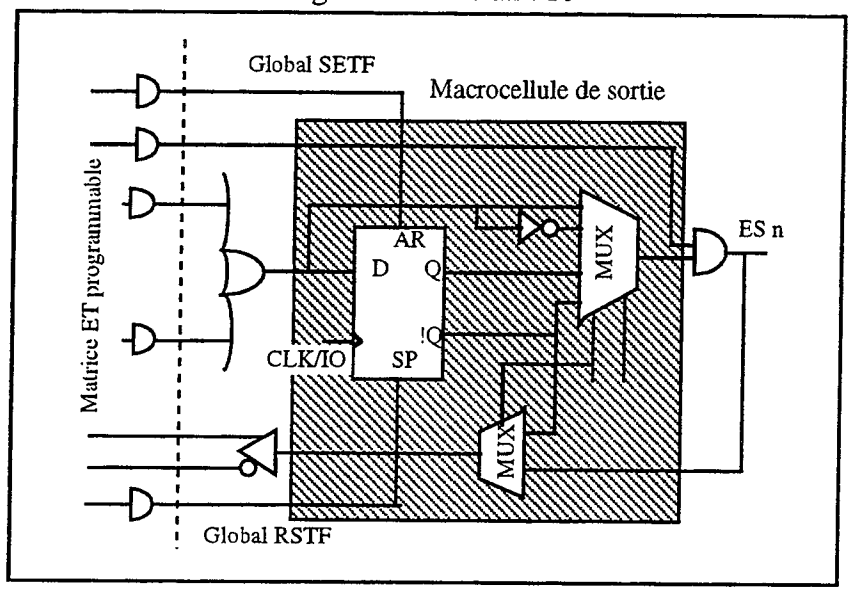


Figure 1.4 Macro cellule de PAL 22V10

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

b) Réseaux multiblocs ne partageant pas les monômes:

La structure de ces réseaux est constituée par un ensemble de blocs de même nature que celui des réseaux monoblocs, et par un canal de connexion assurant la distribution des entrées et la liaison entre les blocs. Cette structure est présentée dans la figure 1.5

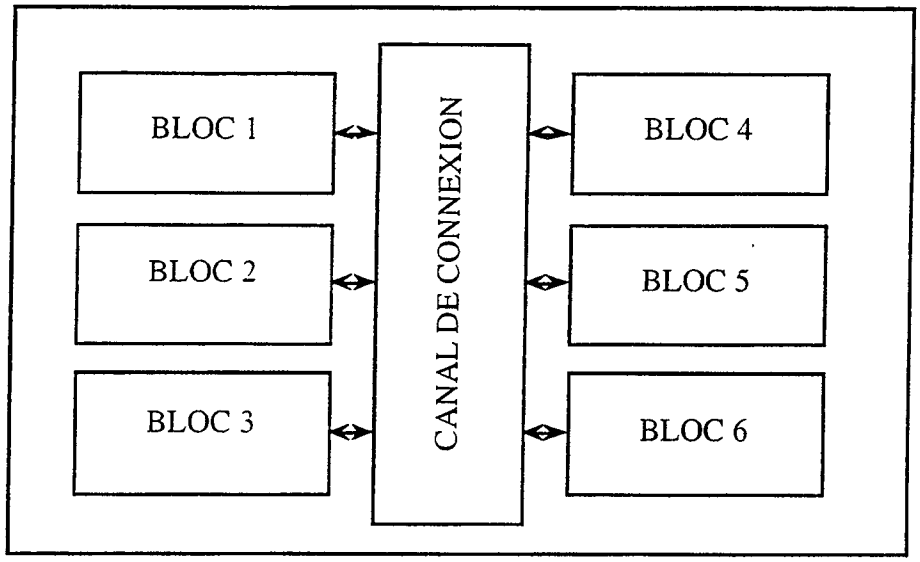


Figure 1.5 Réseau multibloc non-partageant des monômes

Chaque bloc a une structure très proche de celle des réseaux monoblocs, mais ils se singularisent par deux aspects. Le premier a trait à la taille du plan "ET". Elle est identique pour tous les blocs et chaque entrée est dupliquée pour réaliser sa forme normale et sa forme complémentée. Le second concerne les monômes. Ils ne sont pas toujours attribués qu'une seule fois dans la limite des monômes pour une sortie.

Chaque sortie dispose d'une possibilité supplémentaire de rebouclage. Cela permet, si la macrocellule associée à cette sortie génère un signal uniquement interne, d'utiliser la sortie comme une entrée.

Le canal de connexion, pour sa part, assure la distribution des signaux d'entrée aux blocs. Ceux-ci ont plusieurs origines:

* Soit directement de l'extérieur, à partir des broches dédiées aux entrées ou des broches de sortie utilisées comme entrée.

* Soit de l'intérieur à partir du rebouclage d'une macrocellule.

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

I.1.2 Réseaux partageant les monômes:

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

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

a) Réseaux monobloc partageant les monômes

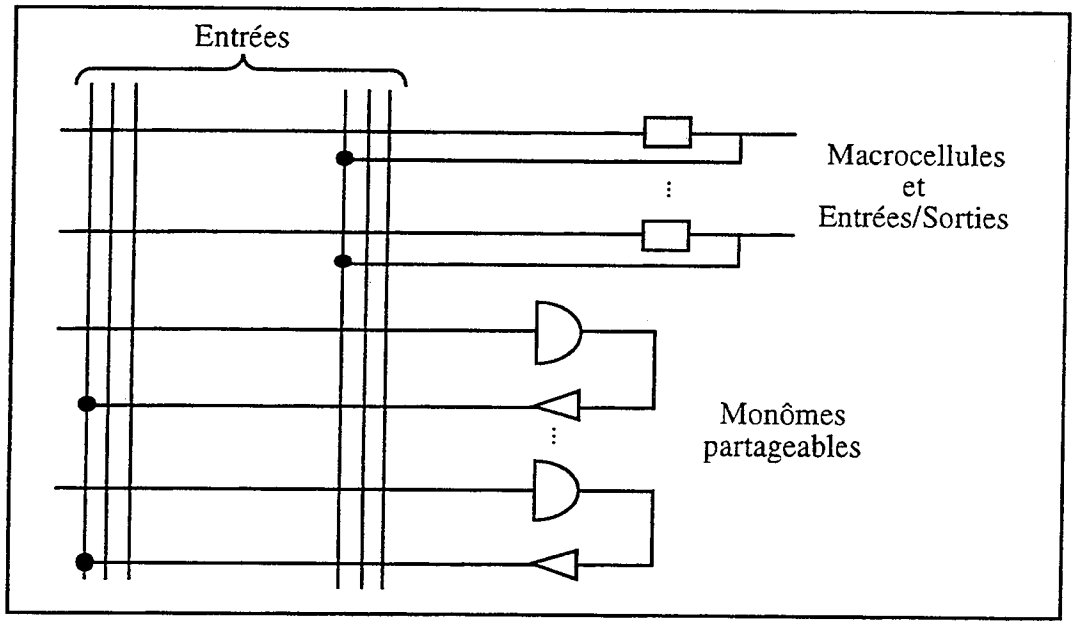


Figure 1.6 Réseau monobloc partageant des monômes

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

b) Réseaux multiblocs partageant les monômes:

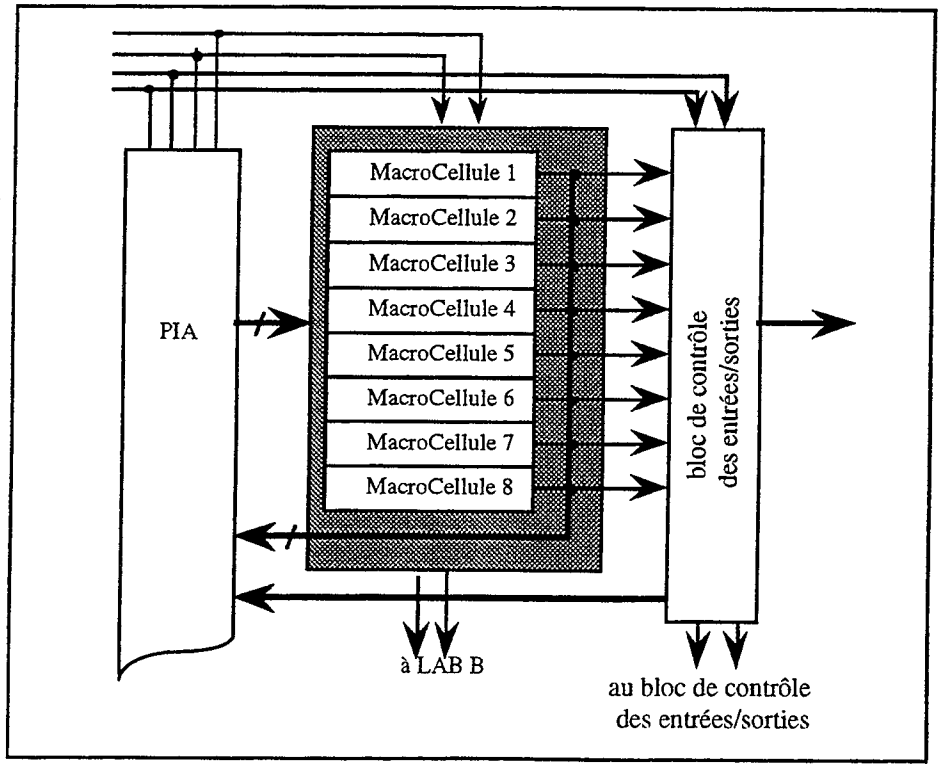


Figure 1.7 Réseaux multiblocs partageant les monômes

La structure de ces réseaux est constituée d'un ensemble de blocs et d'un canal de connexion assurant la distribution des entrées et la liaison entre les blocs. Cette structure multibloc est illustrée dans la figure 1.7 pour les réseaux multiblocs ne partageant pas les monômes mais dans ce cas, les monômes dans les blocs sont partagés.

I.2. Réseaux à faible granularité (les FPGAs)

Cette famille de réseaux programmables se rapproche des réseaux prédiffusés par sa structure régulière. Elle est caractérisée par un ensemble de petites cellules assurant la logique, réparties uniformément dans le boîtier et par des canaux de connexion permettant de relier les cellules entre elles, ainsi qu'à l'extérieur, via des cellules d'entrée et de sortie. Une fonction booléenne est implantée sur ces réseaux en la décomposant en sous-fonctions, chacune correspondant à la capacité d'une cellule.
Au cours de la synthèse, pour réduire au maximum le nombre de cellules, la fonction initiale est exprimée sous une forme factorisée. La méthode de factorisation sera présentée dans le chapitre 2. L'implantation finale, sous la forme de sous-fonctions elles-mêmes décomposées si nécessaire, est dite de "multicouche", par opposition à l'implantation deux couches (somme de monômes) des réseaux à deux plans.
Les réseaux à faible granularité ou les FPGAs (Field Programmable Gate Array) peuvent être divisés en deux sous-familles selon la configuration de la cellule de base: les FPGAs à base de "Look Up Tables" (LUT) et les FPGAs à base de multiplexeurs.

I.2.1 Les FPGAs à base de "Look Up Tables" (LUT)

La fonctionnalité de la cellule de base de ce type FPGA est déterminée par une petite mémoire appelée Look Up Table (LUT) stockant le tableau de vérité d'une fonction booléenne à m variables. On parlera alors de LUT-m, une telle cellule pourra implémenter n'importe quelle fonction de m variables. On peut citer à titre d'exemple les boîtiers de Xilinx leader du marché, XC3000, XC4000, XC5000, suivi du réseau ORCA de AT&T et plus récemment Flex8000 de Altera. Ces technologies diffèrent entre elles par la connexion et le partage de variables d'entrée et de sortie pour chaque cellule de base (LUT) et le réseau d'interconnexion entre les cellules.
Nous pouvons distinguer deux grands types de configurations de cellules LUTs:
* Les cellules dont les variables ne sont pas partagées.
* Les cellules dont les variables sont partagées

a) Les cellules dont les variables ne sont pas partagées

Ce type de FPGA est illustré par le boîtier Altera/Flex8000. Le réseau de cellules est composé de LUT-m. Les variables de chaque LUT-m sont propres à cette cellule et ne sont pas partagées. La synthèse pour cette cible de technologie a pour but de décomposer le réseau booléen en réseau de cellules d'au plus m variables.

La cellule de base du réseau programmable de la famille Flex8000 d'Altera est représentée dans la figure 1.8 Chaque cellule de base contient un LUT-4, une bascule et de la logique supplémentaire destinée à l'arithmétique et à la synchronisation.

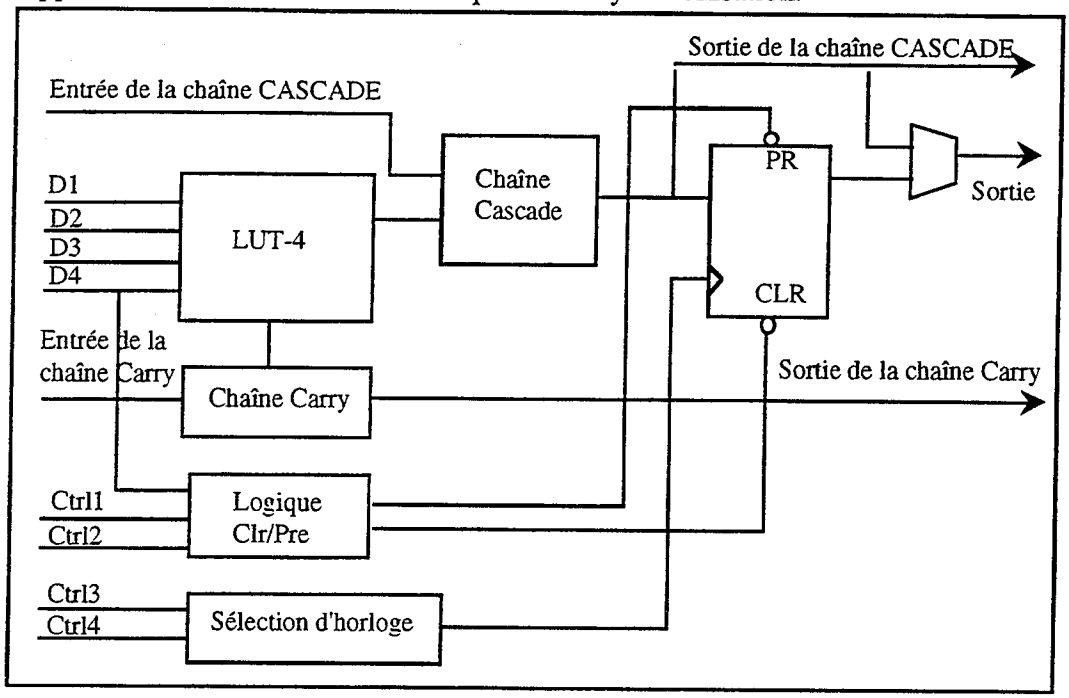


Figure 1.8 Cellule de base de Flex8000 (Logic Element)

b) Les cellules dont les variables sont partagées

La famille XC3000 de Xilinx partage les variables. Le partage de variables entre des cellules permet de réaliser simultanément des fonctions booléennes de plusieurs variables , en partageant ou ne partageant pas des variables entre des cellules. Mais le partage de variables pose un problème difficile pour la synthèse.

Les technologies XC3000 de Xilinx ou ORCA de AT&T font partie de ce type de FPGA. Leur cellules de base sont présenté dans la figure 1.9

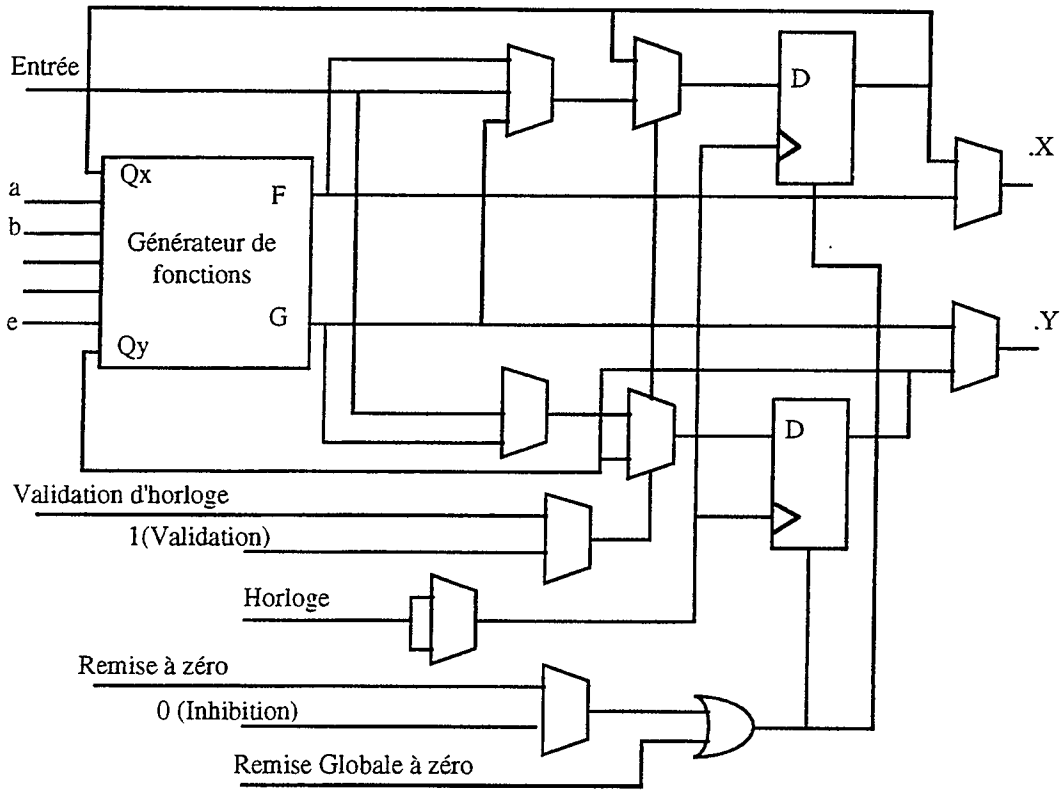


Figure 1.9 Le CLB de la famille XC3000 de Xilinx

Le générateur des fonctions de XC3000 peut réaliser deux LUT-4 dont le nombre total de variables ne dépasse pas 5. Le nombre maximal de variables partagées entre ces deux LUT-4 est donc 3.

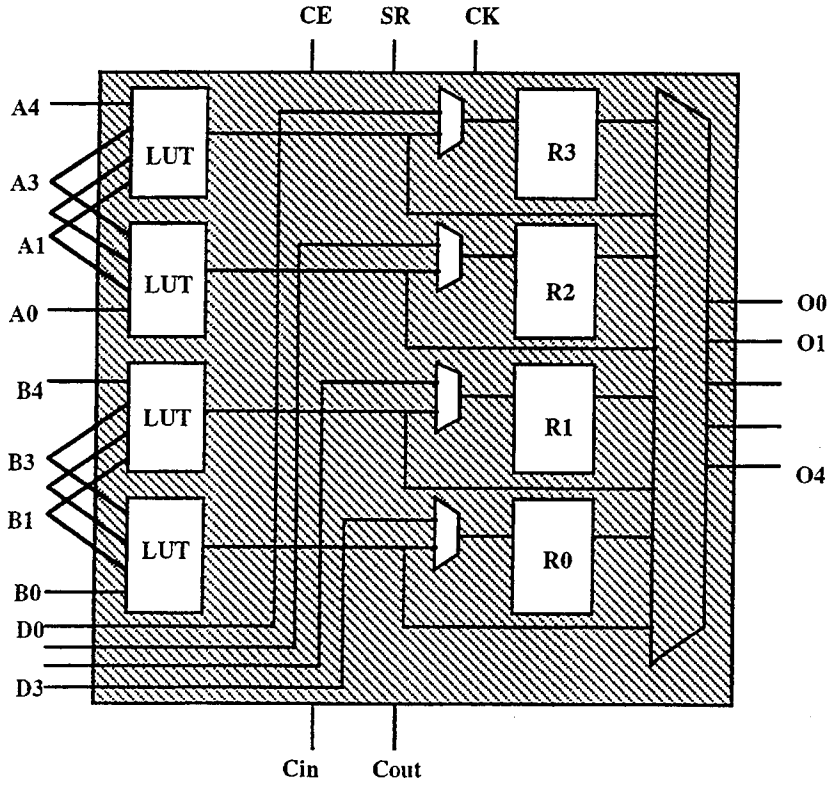
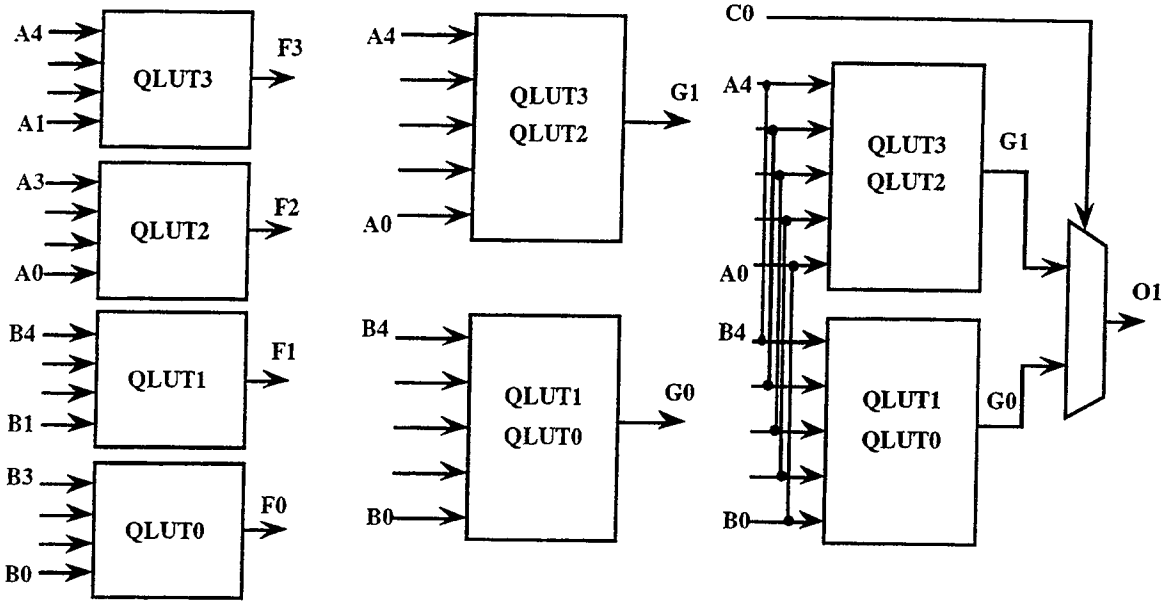


Figure 1.10 Cellule de base du boîtier ORCA d'AT&T

La cellule de base du boîtier ORCA (figure 1.10) est composée de quatre LUT-4 et d'une logique supplémentaire permettant de réaliser plusieurs configurations (Figure 1.11). Les QLUT3 et QLUT2 (resp. QLUT1 et QLUT0) peuvent être utilisés séparément pourvu que le nombre total de variable ne dépasse pas 5 (Figure 1.11a) ou ils peuvent être combinés pour créer un LUT-5 (Figure 1.11b). De plus, tous les quatre LUT-4 peuvent être regroupés pour former un LUT-6 (Figure 1.11c)

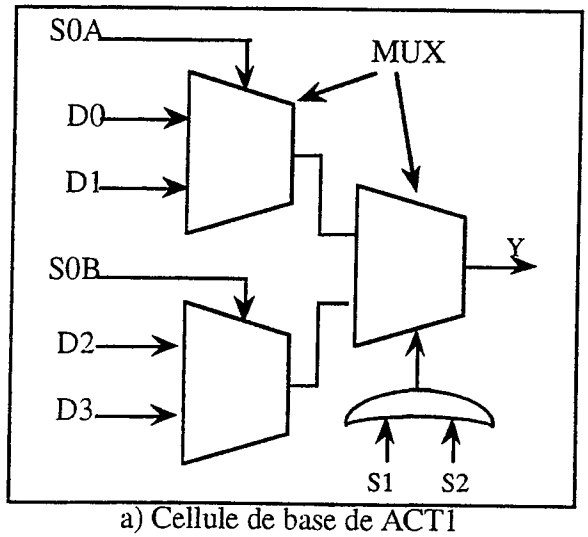


a) Model b) Mode2 c) Mode 3

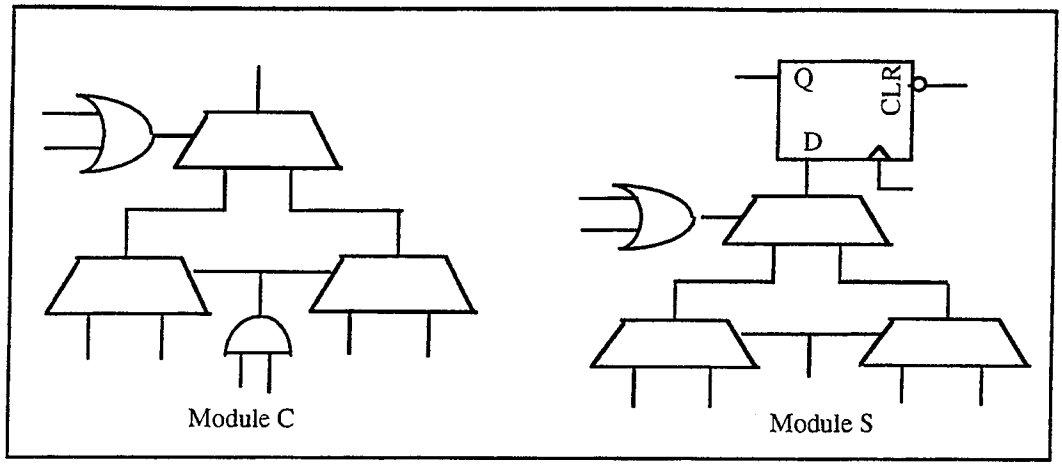
Figure 1.11 Les configurations de cellule de base du boîtier ORCA

I.2.2 Réseaux à base de multiplexeurs

La cellule de base est composée d'un arbre de multiplexeurs. Les familles ACT1, ACT2, ACT3 d'Actel et pASIC de QuickLogic font parties de ce type de réseau. Les configurations de cellule de base de ces technologies diffèrent par la logique autour des multiplexeurs. Ils sont présentées dans les figures 1.12, 1.13, 1.14.



a) Cellule de base de ACT1



c) Cellule de base de ACT2

Figue 1.12 Cellule de base à base de multiplexeurs

La cellule de base de ACT1 a huit entrées et une sortie qui permettent d'implanter plusieurs fonctions booléennes et une variété de bascules D.

Les cellules de base de ACT2 sont classées en deux types: combinatoire et séquentielle. Le bloc combinatoire est une version optimisée de ACT1 pour implanter des macro-fonctions de plusieurs entrées. Il peut réaliser des fonctions suivantes:

Z = !S1*(D00*!S0 + D01*S0) + S1*(D10*!S0 + D11*S0)

où S0 = A0*B0, S1 = A1 + B1

Le bloc séquentiel inclut aussi une partie combinatoire, ce qui permet d'implanter une couche logique de plus sans augmenter le délai. Ce bloc peut être programmé pour réaliser plusieurs configurations comme suit:

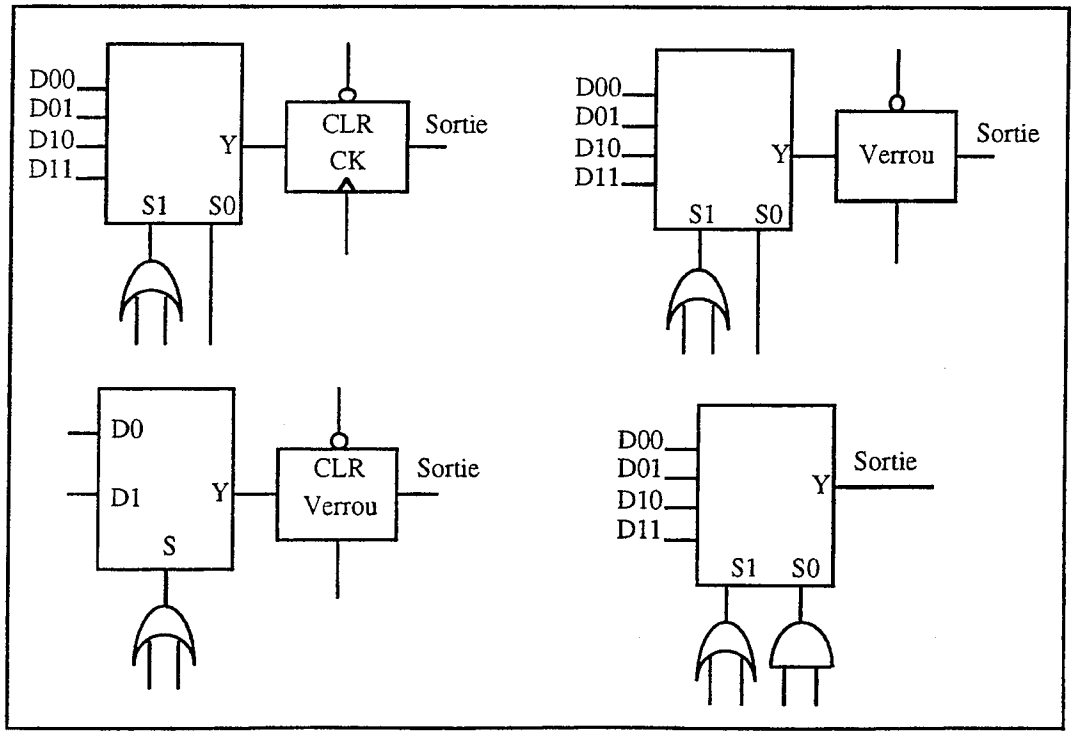


Figure 1.13 Configurations de module séquentiel de ACT2

La cellule de base de QuickLogic est composée de 3 multiplexeurs et d'une logique supplémentaire spécialement dédiée à l'implantation des décodeurs. La cellule a aussi une bascule qui permet de réaliser des circuits séquentiels (Figure 1.14)

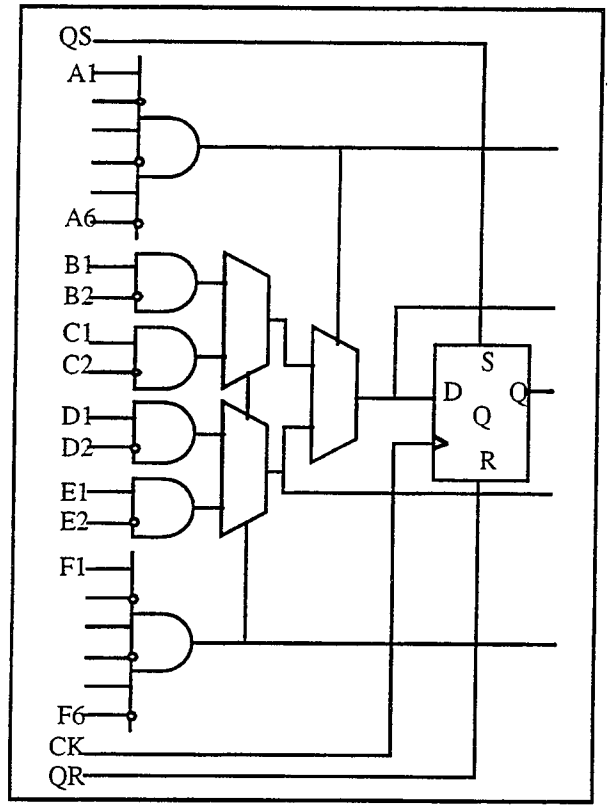


Figure 1.14 Cellule de base de QuickLogic

II. AUTRES CLASSIFICATIONS

II.1 En terme de programmabilité de l'élément de technologie

II.1.1 Programmable une fois seulement

L'élément de technologie ne peut être programmé qu'une seule fois. La programmation des boîtiers est faite hors de l'application en utilisant un dispositif spécial. Les technologies d'Actel utilisant les antifusibles PLICE (Programmable Low Impedance Circuit Element) et de QuickLogic, Crosspoint utilisant les antifusibles ViaLink font parties de ce type de FPGAs.

Le délai d'interconnexion est faible grâce aux faibles valeurs de RC du contact Antifusible pour les technologies Antifusibles (Actel, QuickLogic), ou du simple contact métallique pour les tecnologies PROM.

Les avantages de ce type de FPGA sont la non-volatilité et la haute vitesse grâce au faible délai de l'interconnexion. De plus, la performance de l'application est predictible.

II.1.2 Reprogrammable

L'élément de technologie peut être programmé plusieurs fois. Cette reprogrammabilité donne une grande souplesse à la réalisation des circuits de prototypage ou de multifonctions.

L'interconnexion dans les réseaux reprogrammables comme la famille XC4000 de Xilinx est assurée par les transistors de transfert, ayant une valeur RC importante, contrôlés par les éléments SRAM occupant une surface importante, la vitesse est limitée et la surface est plus grande que pour les boîtiers programmables une seule fois.

L'élément de technologie peut être programmé pendant le fonctionnement (in-circuit) ou hors de l'application (out of circuit).

* Hors de l'application: Le boîtier doit être enlevé de l'application pour le programmer en utilisant un dispositif spécial. La technologie EPROM et la plupart des technologies EEPROM appartiennent à ce type.

* Dans l'application: L'application peut être reprogrammée pendant le fonctionnement du boîtier sans avoir besoin d'un dispositif spécial.

II.2 En terme de technologies

II.2.1 Technologie SRAM

La programmation est assurée en utilisant des transistors de transfert, portes de transfert ou multiplexeurs qui sont contrôlés par des cellules SRAM. L'utilisation des cellules SRAM pour contrôler la programmation est illustrée dans la figure 1.15

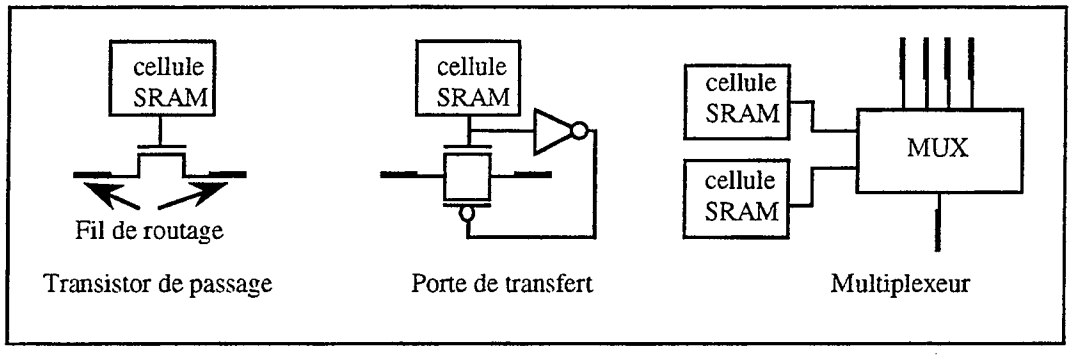


Figure 1.15 Les éléments de technologies

Comme la cellule SRAM est volatile, nous devons reconfigurer le boîtier à partir d'une mémoire permanente chaque fois que l'application est mise en service. La surface demandée pour un élément de technologie est plus large que pour les autres technologies à cause de la surface de la SRAM elle-même et des transistors ou multiplexeurs. L'avantage essentiel de ce type FPGA est sa reprogrammabilité. De plus, le processus de technologie CMOS peut être réutilisé.

II.2.2 Technologie EPROM&EEPROM

La technologie EPROM est la même que celle de mémoire EPROM. Son avantage est la non-volatilité mais il consomme de l'énergie en mode statique.
La technologie EEPROM ressemble à la technologie EPROM mais le transistor de la technologie EEPROM peut être reprogrammé dans l'application. Pourtant, sa surface est deux fois plus importante que celle de l'EPROM et le dispositif a besoin de plusieurs voltages différents pour son fonctionnement.

II.2.3 Technologies antifusibles

La surface d'un antifusible est très petite par rapport à celle d'autres technologies. La résistance et la capacité du contact de la connexion sont donc plus petites: ceci rend le délai introduit par l'interconnexion plus petit.

a) Antifusible PLICE: (d'Actel)

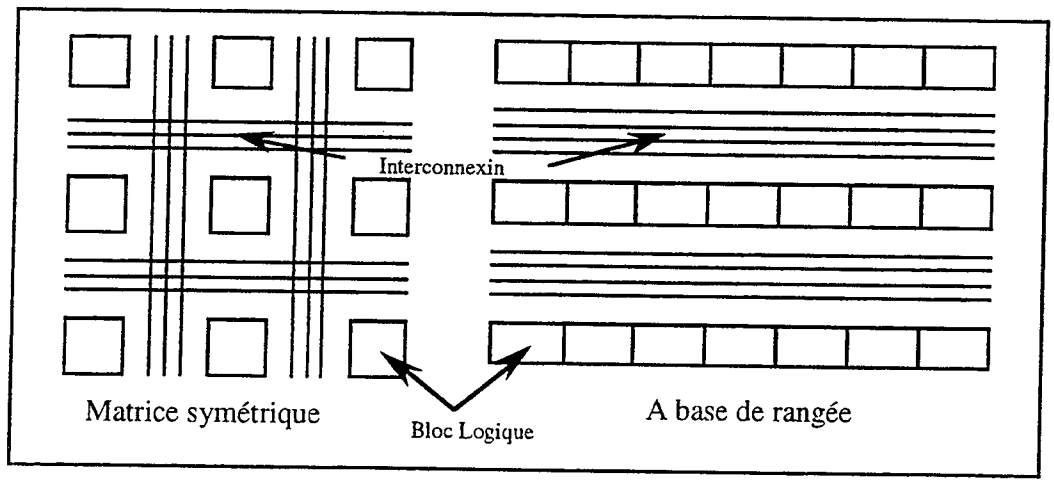
La connexion est assurée en faisant claquer le diélectrique isolant entre deux couches de diffusion et de polysilicium. Ce contact a une haute résistivité (500 Ohms) par rapport aux autres types d'antifusibles qui rend le délai causé par RC de la connexion plus élevé. En plus, la surface pour le contact est assez large.

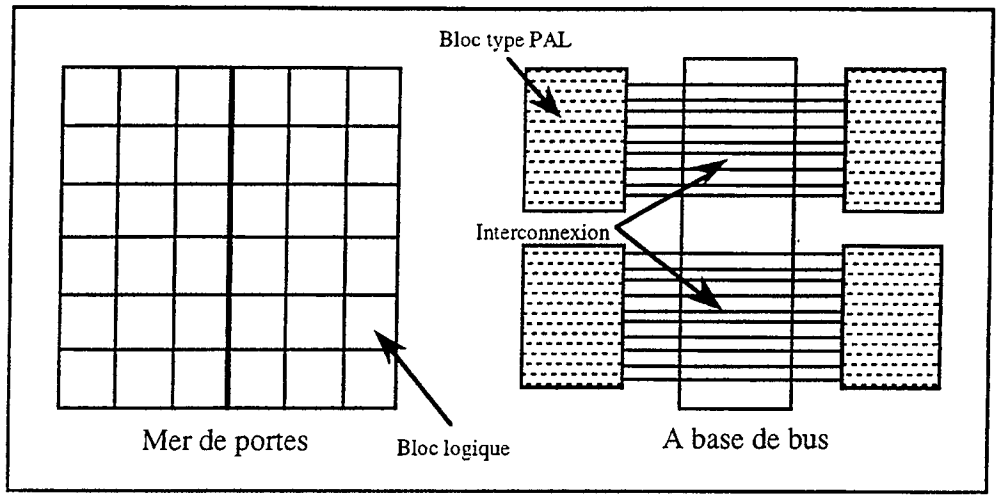
b) Antifusible ViaLink (de QuickLogic, Crosspoint)

La connexion est faite par un alliage de silicium amorphe qui a une faible résistivité (50-80 Ohms), ce qui diminue beaucoup le délai de l'interconnexion.

II.3 En terme d'architecture d'interconnexion

La figure 1.16a illustre de différentes architectures d'interconnexion. La connexion peut être assurée soit par des interrupteurs soit par des multiplexeurs.





Figures 1.16a Architectures d'interconnexion

II.3.1 L'architecture d'interconnexion à base d'interrupteurs

La connexion entre les cellules de base est faite par des interrupteurs et des lignes horizontales, verticales distribuées partout dans le boîtier.

Chaque interrupteur correspond électriquement à une paire de RC. Comme l'interconnexion traverse plusieurs interrupteurs, son délai est important car il augmente très vite en fonction du nombre d'interrupteurs sur le chemin connectant un bloc logique à l'autre. Ce type de délai contribue à une part importante dans le délai total. Il est souvent plus important que le délai du aux blocs logiques, surtout quand la ressource de routage est satutée.

De plus, comme le nombre d'interrupteurs sur le chemin d'interconnexion n'est connu qu'après le placement et le routage, le délai d'interconnexion n'est pas prédictible pendant l'étape de synthèse logique.

De toutes façons, son avantage est d'assurer une bonne connectivité à haute densité.

Ce type d'architecture peut être organisé sous forme des matrices:

* La matrice peut être symétrique (Xilinx, QuickLogic).
* La matrice peut être organisée en rangées (Actel, Crosspoint).
* La matrice peut être une mer de portes (AT&T, Atmel).

II.3.2 L'architecture à base de bus

Le réseau d'interconnexion entre les cellules est fait à partir de bus. Le bus est fait à base des multiplexeurs. Quand la ressource du routage n'est pas encore saturée, il assure un délai prédictible pour chaque connexion et une bonne connectivité mais avec une grande surface. On peut citer par exemple les boîtiers EFP5000, 7000 d'Altera.

La figure 1.16b illustre l'architecture d'interconnexion dans le boîtier MAX7000 d'Altera.

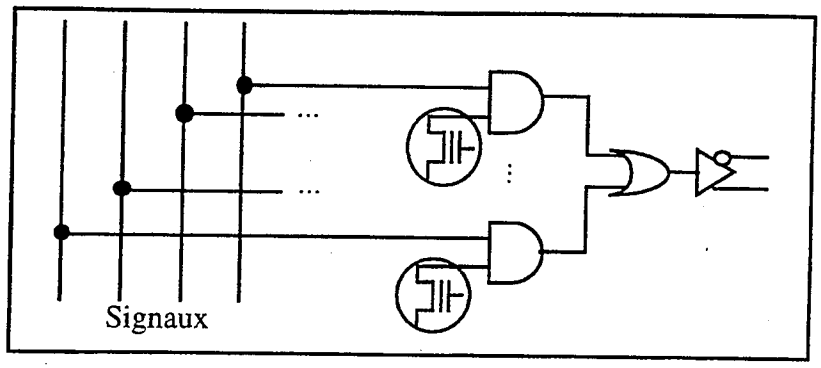


Figure 1.16b Architecture d'interconnexion dans le boîtier MAX7000

Les tableaux 1.1 et 1.2 présentent la liste des sociétés connues de CPLD/FPGA et des technologies des réseaux programmables.

Société	Interconnexion	Cellule de base	Technologie
AT&T	Interrupteurs	LUT	SRAM
Xilinx	Interrupteurs	LUT	SRAM
Actel	Interrupteurs	Multiplexeurs	Antifusible PLICE
Altera/MAX	Bus	Bloc PAL	EPROM
Altera/Flex8000	Bus	LUT	SRAM
AMD	Bus	Bloc PAL	EEPROM
QuickLogic	Interrupteurs	Multiplexeurs	Antifusible ViaLink

Tableau 1.1 Sommaire de quelques sociétés connues de CPLD/FPGA

Technologies	Volatile	Repro-gramma-ble	Surface	R(Ohm) du contact	C(pF) du contact
SRAM	Oui	Dans le circuit	Importante	1K-2K	10-20 pF
Antifusible PLICE	Non	Non	Très petite	300-500	3-5 pF
Antifusible ViaLink	Non	Non	Très petite	50 - 80	1-3 pF
EPROM	Non	Hors de circuit	Petite	2-4K	10-20 pF
EEPROM	Non	Dans le circuit	2 X EPROM	2-4K	10-20 pF

Tableau 1.2 Sommaire des technologies des réseaux programmables

II.4 Relations entre les fonctions implémentées et les CPLDs/FPGAs

II.4.1 Machines d'états (Finite State Machines: FSM)

Les architectures à base des PALs caractérisées par leur plans ET/OU conviennent très bien aux systèmes nécessitant des logiques rapides et simples avec plusieurs entrées. Pour

ce type d'application, nous pouvons citer les machines d'états, les décodeurs... Les familles MAX d'Altera, MACH d'AMD, Lattice sont très efficaces pour cette application.

II.4.2 Logique aléatoire

L'avantage principale des familles de FPGAs de faibles granularités (Actel, Xilinx, Flex8000, ORCA) est que la petite surface et la simple fonctionnalité de la cellule de base permettent de réaliser facilement et efficacement de la logique aléatoire.

II.4.3 Décodeurs à haute vitesse

Les architectures dominantes pour réaliser les décodeurs à haute vitesse sont toujours celles à base de PAL (MAX, MACH, Lattice) mais quelques autres, par exemple, Flex8000, QuickLogic, Xilinx,.. possèdent des mécanismes spéciaux ou des cellules de base avec une grande sortance, sont améliorées pour bien s'adapter à cette application.

II.4.4 Arithmétique

La cellule de base des familles Flex8000, ORCA, Xilinx, Actel, Atmel, ... est conçue pour permettre de réaliser facilement des fonctionnalités arithmétiques performantes sous forme des blocs macros: des additionneurs rapides. Les blocs macros donnent des perspectives intéressantes dans les applications arithmétiques.

II.4.5 Mémoire

Les cellules de base de Xilinx, ORCA sont des LUTs ce qui permettent alors de réaliser facilement de la mémoire rapide à haute densité.

III PRESENTATION DES RESEAUX PROGRAMMABLES XC4000 ET ALTERA/FLEX8000

III.1 Réseau programmable de la famille XC4000

Le réseau reprogrammable de XC4000 est composé de trois éléments configurables:

* La cellule de base appelée bloc logique configurable (CLB: Configurable Logic Block) fournit la logique combinatoire et séquentielle.
* Le réseau d'interconnexion assure la connexion entre les CLBs, ou entre les CLBs et les blocs entrée/sortie.
* Les blocs entrée/sortie (Input/Output Block, IOB) réalisent l'interface entre les plots et les signaux internes.

III.1.1 Structure du CLB

La structure du CLB de la famille XC4000 est présentée dans la figure 1.17

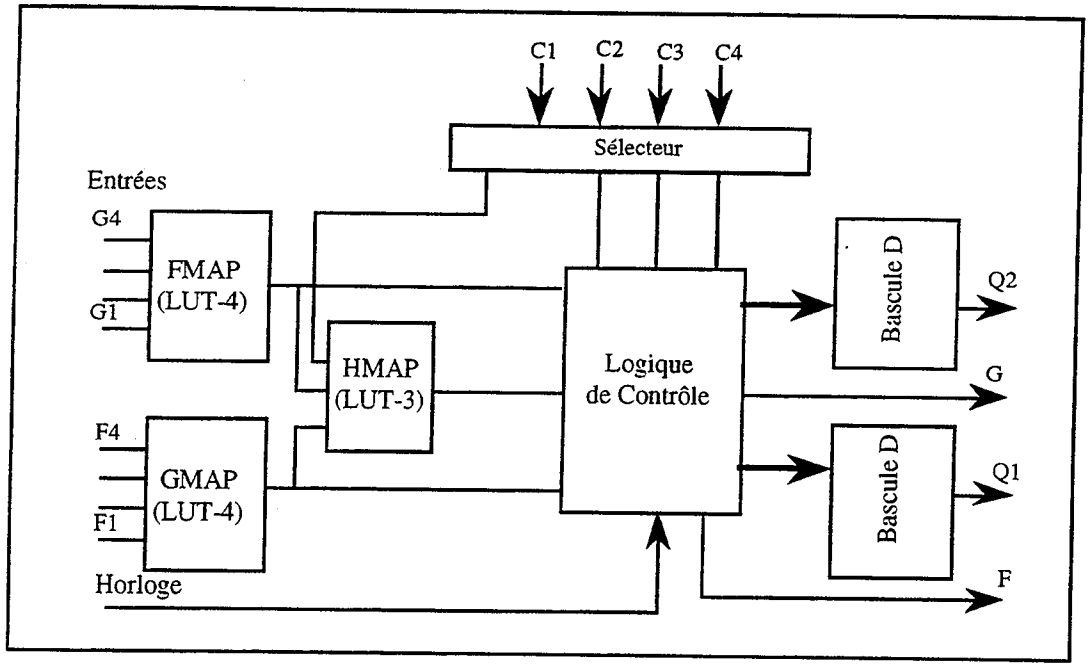


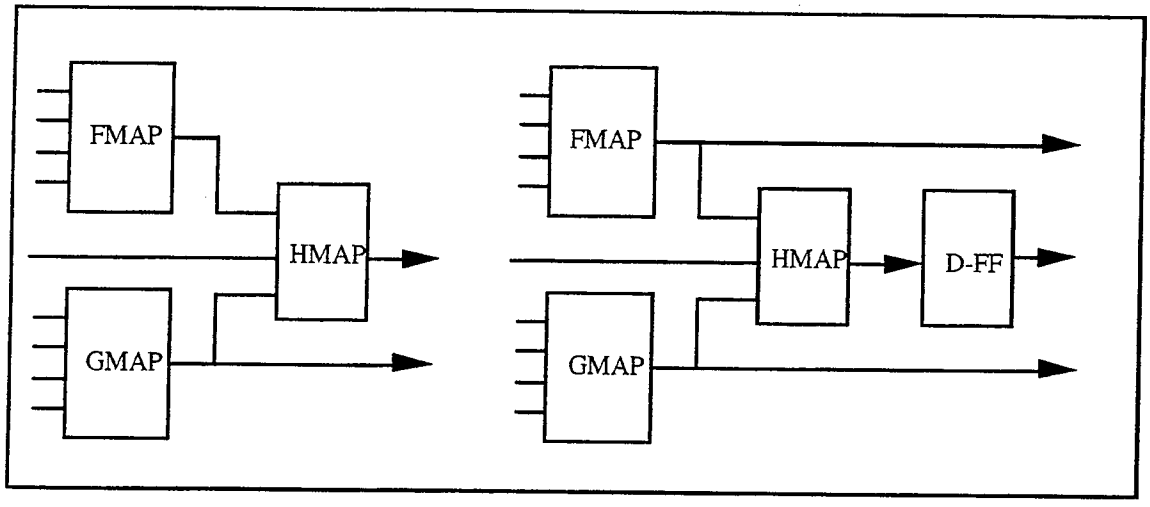
Figure 1.17. Structure de CLB4000

Du point de vue de la synthèse logique, le CLB de la famille XC4000 contient deux générateurs de fonctions d'au plus 4 variables (les FMAP et GMAP sont des LUT-4), un générateur d'au plus 3 variables (le HMAP est un LUT-3) et deux bascules D. Le CLB peut être programmé en plusieurs configurations comme dans les figure 1.18

Nous pouvons remarquer que:

* Deux sorties de ces trois LUTs sont utilisables à l'extérieur du CLB.

* Les entrées de deux bascules D peuvent être connectées aux sorties des parties combinatoires ou l'une de ces deux entrées peut être une entrée directe.



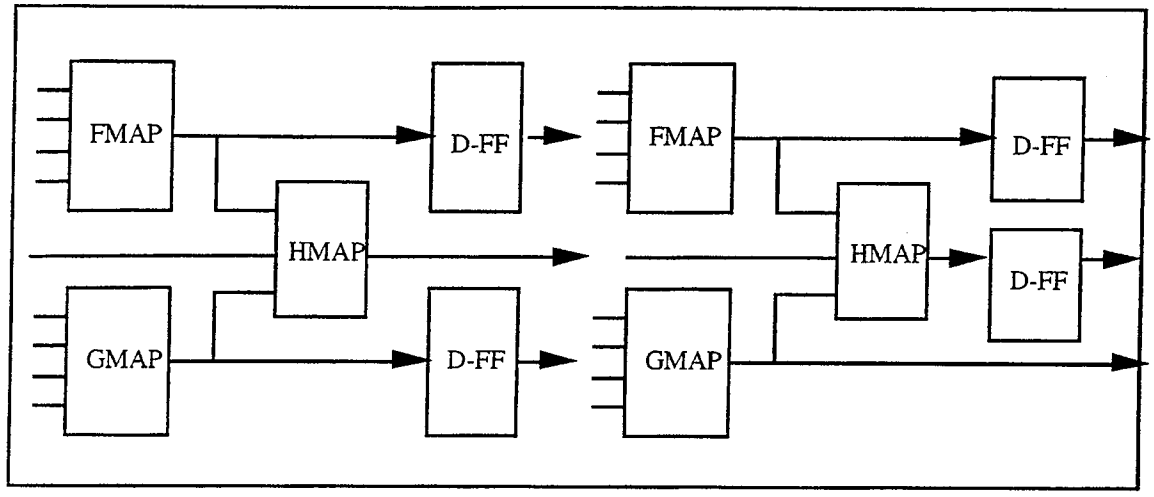


Figure 1.18 Les configurations du CLB de la famille XC4000

III.1.2 Réseau d'interconnexion

Le réseau d'interconnexion est composé de trois groupes principaux: interconnexion générale, interconnexion longue, interconnexion double longueur.

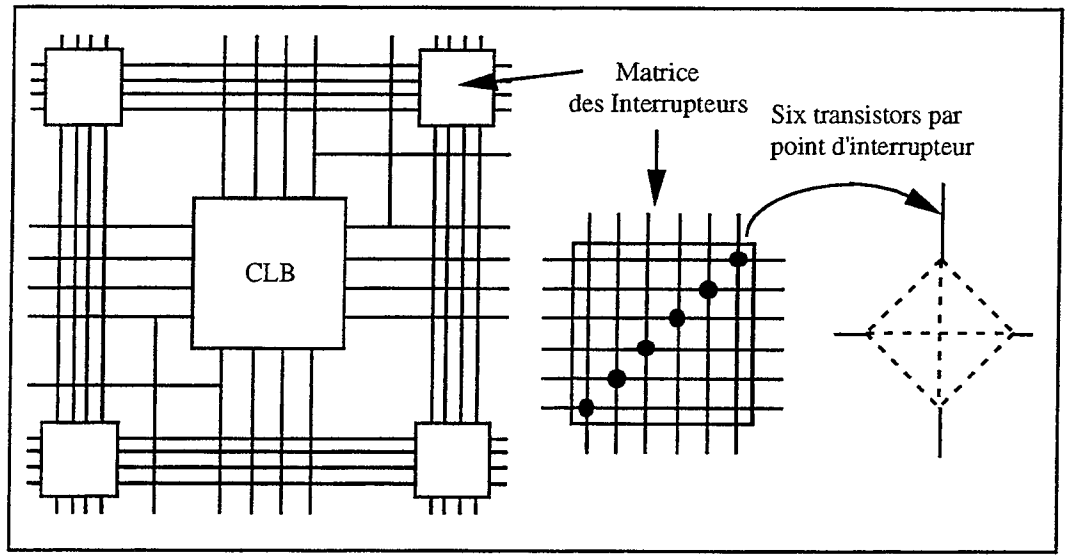


Figure 1.19. Interconnexion générale

* Les interconnexions générales sont en grand nombre et utilisent des matrices d'interrupteurs comme dans la figure 1.19. Ce groupe est le moins performant en temps de traversée mais donne une bonne connectivité.

* Les connexions longues lignes (Long Lines) parcourant le circuit entier en hauteur ou en largeur (Vertical Long Lines, Horizontal Long Lines) donnent un délai très court mais leur nombre est très limitées. Ces connexions ont pour but de distribuer des horloges, des signaux de haute sortance ou de minimiser le "skew" des horloges.

* Les interconnexions doubles longueurs sont semblables aux interconnexions générales mais alternées sur deux lignes de matrices de commutateurs.

III.1.3 Blocs d'entrée/sortie:

Le bloc d'entrée/sortie de la famille XC4000 est constitué de deux points de mémoire, d'un buffer d'entrée et d'un buffer de sortie. Le buffer de sortie peut être configuré en une porte trois états.

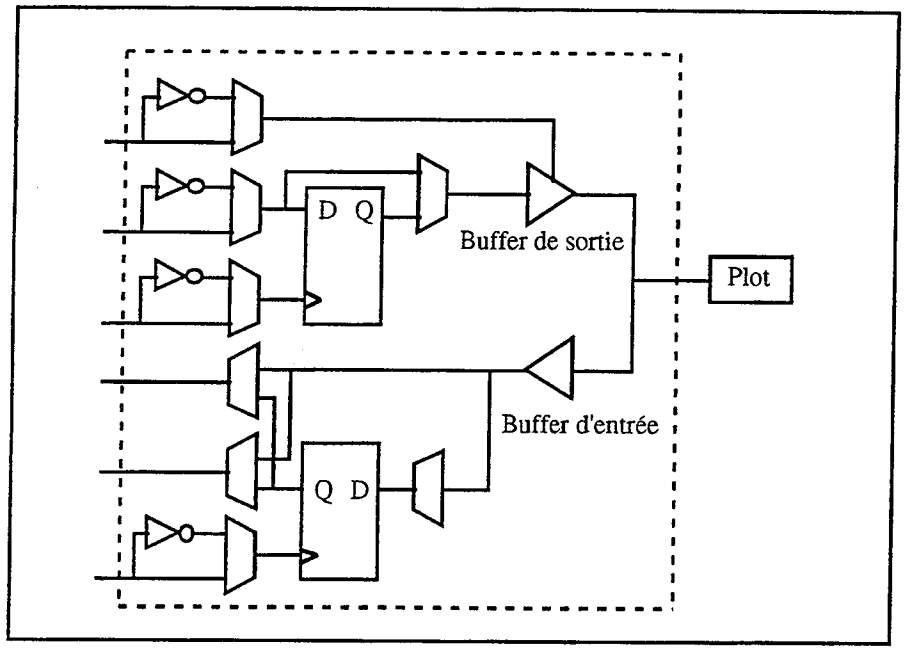


Figure 1.20 Bloc Entrée/Sortie de la famille XC4000

Cette configuration de bloc d'entrée/sortie permet d'y intégrer une bascule, une porte trois états pour réaliser un buffer de sortie de trois états, un buffer d'entrée latché ou non-latché.

III.2 Réseau programmable de la famille Flex8000

L'architecture du boîtier Flex8000 est constituée d'une matrice de cellules de base appelées éléments logiques (Logic Element: LE). Chaque LE contient un LUT-4 fournissant la logique combinatoire et un registre programmable assurant la logique séquentielle. Les LEs sont regroupés en ensembles de huit formant des Matrices de Blocs Logiques (en anglais: Logic Array Block, LAB). Chaque LAB est une structure indépendante composée des entrées, interconnexion et signaux de contrôle communs. L'interconnexion entre des LEs est assurée par un réseau appelé FastTrack Interconnect composé de canaux rapides et continus qui traversent entièrement le boîtier horizontalement et verticalement.

III.2.1 Structure de l'élément logique (LE)

Chaque LE contient un LUT-4, une bascule programmable, une chaîne CARRY et une chaîne CASCADE. La figure 1.21 montre la structure générale de LE.

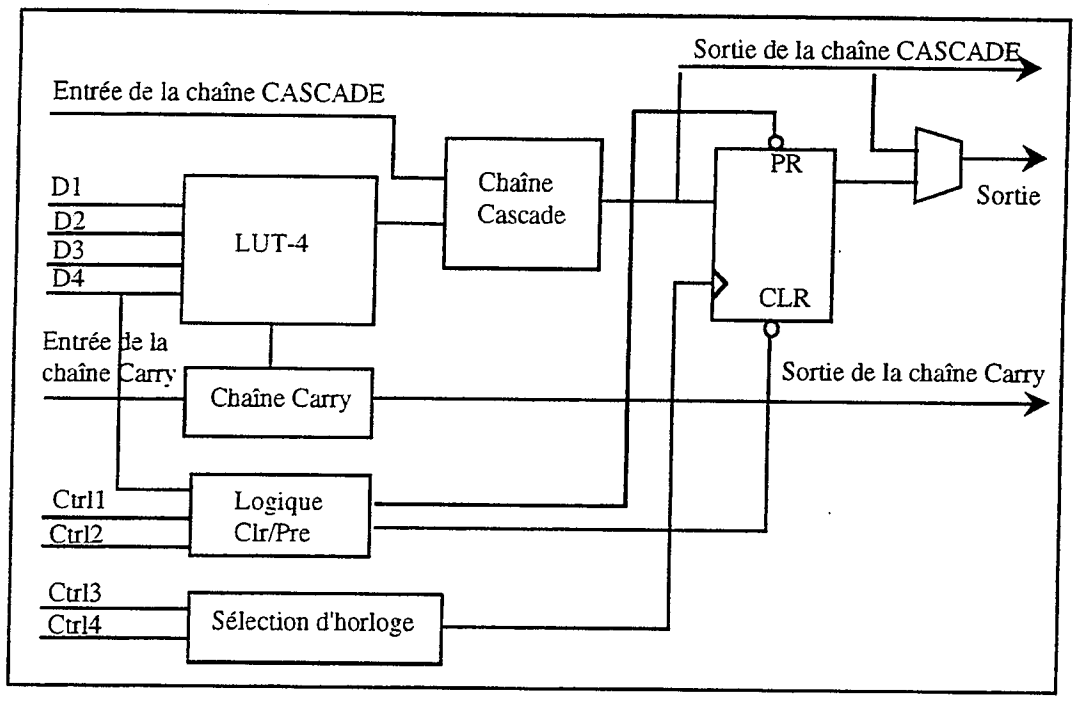


Figure 1.21 Structure de l'élément logique (LE) de la famille Flex8000

La bascule programmable peut être programmée pour obtenir un D, T, JK, RS-FlipFlop. Flex8000 fournit deux chemins dédiés à haute vitesse, la chaîne CARRY et la chaîne CASCADE. pour connecter les LEs voisins sans utiliser les connexions générales moins performantes. La chaîne CARRY est utilisée dans les opérations arithmétiques tandis que la chaîne CASCADE a pour but d'implanter des logiques avec de nombreuses entrées à délai minimal. De plus, la chaîne CASCADE peut être utilisée pour diminuer la surface et également le délai. Les chaînes CARRY et CASCADE connectent tous les LEs dans le même LAB et tous les LABs dans la même rangée.

a. Chaîne CARRY:

La chaîne CARRY transmet le signal CARRY entre les LEs avec un délai très court (moins d'une nanoseconde pour la traversée d'un LE). Ceci permet de réaliser des opérateurs arithmétiques de haute vitesse.

b. Chaîne CASCADE:

La chaîne CASCADE a pour but de réaliser des fonctions logiques, ayant de nombreuses entrées, avec un faible délai (une nanoseconde pour chaque CASCADE de LE). La chaîne CASCADE utilise des ET logique pour connecter les sorties des LEs voisins comme montré dans la figure 1.22

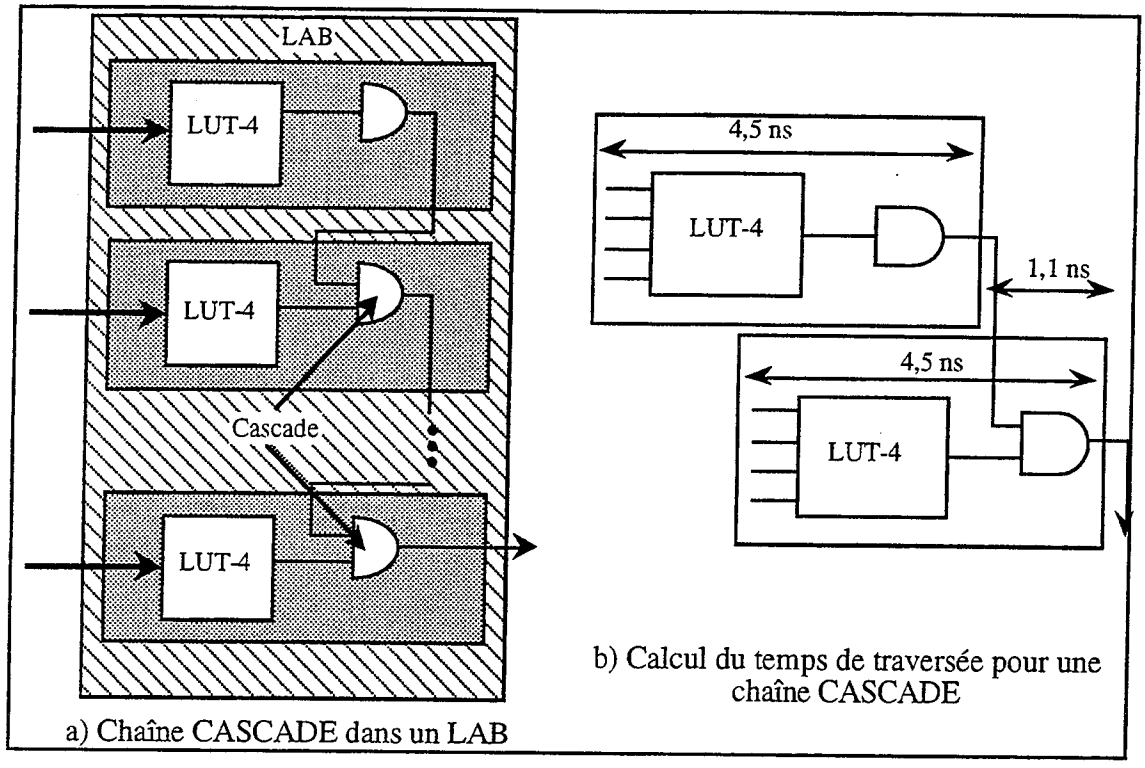


Figure 1.22. Chaîne Cascade ET

III.2.2 Réseaux d'interconnexion

L'interconnexion entre des éléments logiques est assurée par deux types de connexion:

* Un réseau de connexion locale (LAB Local Interconnect) assure la connexion entre les LEs dans le même LAB.

* Un réseau d'interconnexion rapide (FastTrack Interconnect) fournit la connexion entre des LEs de différents LABs et entre des LEs et les blocs d'entrée/sortie. Ce réseau est moins performant que celui de connexion locale.

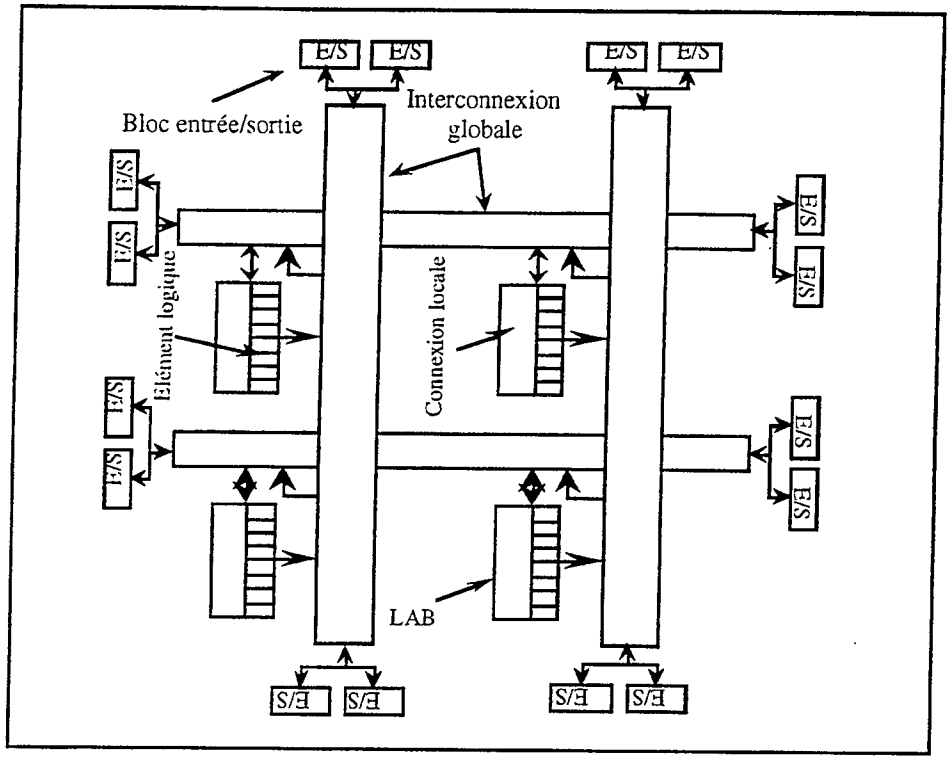


Figure 1.23 Architecture d'interconnexion de la famille Flex8000

Le délai d'interconnexion est presque invariable grâce au réseau de connexion à base de bus. Il est donc prédictible.

III.2.3 Bloc d'entrée/sortie

La figure 1.24 présente la structure d'un bloc d'entrée/sortie.

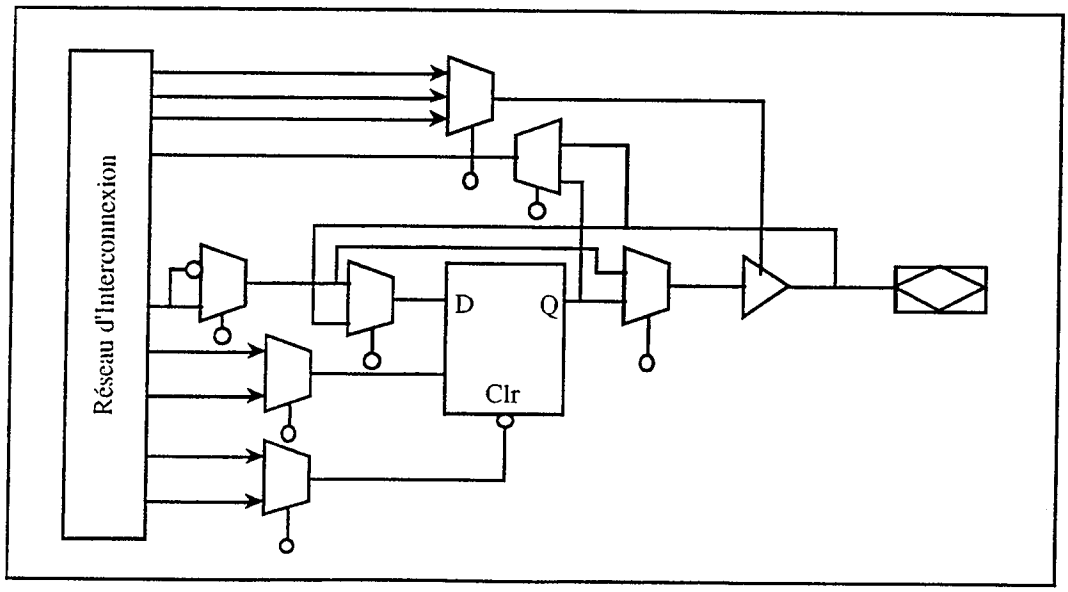


Figure 1.24. Bloc d'entrée/sortie de la famille Flex8000

Les blocs d'entrée/sortie peuvent être utilisés comme entrées, sorties ou éléments bidirectionnels. Chaque bloc a une bascule qui peut être utilisée comme un registre d'entrée ou de sortie. La porte trois états à la sortie permet d'implanter un plot bidirectionnel.

CHAPITRE 2

DEFINITIONS DE BASE ET REPRESENTATIONS DES FONCTIONS BOOLEENNES

I. REPRESENTATION CLASSIQUE DES FONCTIONS BOOLEENNES

I.1 Definitions préalables

Algèbre de Boole

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

Variables booléennes

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

Littéral

Un littéral est une variable booléenne simple dans sa forme directe ou complémentée.
Exemple: a et !a sont deux littéraux distincts.

Fonction booléenne

Une fonction booléenne est une fonction de B^n dans B^m.
- n = 1, m = 1 la fonction est dite fonction simple d'une variable générale.
- n > 1, m = 1 la fonction est dite fonction générale d'une variable générale.
- n = 1, m > 1 la fonction est dite fonction générale d'une variable simple.

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

Fonction booléenne incomplète (ou φ-booléenne)

Une fonction booléenne F est dite incomplète si elle est définie par
$F:B^n \mapsto Y^m$ ou $Y = \{0, 1, \phi\}$
ϕ représente la Valeur Indéfinie VI (appelée "Don't Care en anglais), et signifie que la fonction peut prendre indifféremment la valeur logique 0 ou 1.
E général, une fonction booléenne F peut être définie par trois ensembles disjoints:
- C1 (F) représente la couverture à 1 d'une fonction, c'est à dire l'ensemble des monômes pour lesquels la valeur de la fonction est 1.
- C0 (F) représente la couverture à 0 d'une fonction, c'est à dire l'ensemble des monômes pour lesquels la valeur de la fonction est 0.
- Cϕ (F) représente la couverture à ϕ d'une fonction, c'est à dire l'ensemble des monômes pour lesquels la valeur de la fonction est ϕ.

- La borne supérieure de F est obtenue en remplaçant tous les ϕ par des 1. Elle est notée SUP (F).
- La borne inférieure de F est obtenue en remplaçant tous les ϕ par des 0. Elle est notée INF (F).

Monôme (cube)

Un monôme m (dit également cube) est un produit de p variables simples distincts sous forme normale (x) ou complémentée ($\overline{X}$).
p est le degré (ou longueur) du monôme.
La taille d'un monôme est le nombre de points qu'il couvre.
On rappelle qu'un monôme couvre un point si la valeur de ce monôme vaut 1 en ce point.
Exemple: $m = a * b * \bar{c}$ est un monôme alors que $m = a * \bar{a}$ ne l'est pas.

Relation d'ordre entre monômes:

On dit qu'un monôme m_1 est inférieur à un monôme m_2, que l'on note $m_1 < m_2$ si tous les points couverts par m_1 sont couverts par m_2.
On dit aussi que m_1 est un multiple de m_2.
Exemple: $m_1 = a*b*c*d$, $m_2 = a*b \Rightarrow m_1 < m_2$

Expression booléenne

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

Expression booléenne polynomiale

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

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

Exemple: $F = a * b + c * \bar{d} + g$ est une expression booléenne polynomiale, et $M_F = \{a, b, c, \bar{d}, g\}$

Relation d'ordre sur les fonctions booléennes

Etant donné deux fonctions booléennes F et G, une relation d'ordre sur ces deux fonctions est définie par
$F \leq G \Rightarrow F*G = F$ et $F + G = G$
Autrement dit, tout point couvert par F l'est par G. Cette relation d'ordre est partielle.
Exemple:

$F = a*b*c$

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

$H = a + e$

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

I.2 Représentation classique des fonctions booléennes

Les systèmes de représentation les plus primaires sont basés sur les tables de vérité et les tableaux Karnaugh qui représentent l'énorme avantage d'être canonique. Cependant, étant donnée la taille nécessaire à de talles représentations(2^n unités mémoires pour une fonction de n variables), ces méthodes se sont avérées inefficaces quant à la manipulation des expressions de grande taille.

Les approches les plus utilisées actuellement, nécessitent une taille non exponentielle pour la plupart des fonctions. C'est le cas de la somme des monômes que nous décrivons ci-dessous.

I.2.1 Somme des monômes

Cette représentation consiste à exprimer une fonction en terme de somme de monômes.

Monôme premier

Un monôme m est premier dans une fonction f si et seulement si:

* $m \leq f$
* il n'existe pas de monôme m' différent de m tel que:

$m' \leq f$ et $m \leq m'$

Exemple:

Soit $F = \bar{a}*\bar{c}*\bar{d} + \bar{a}*b*\bar{c} + b*\bar{c}*d + a*b*d + a*c + a*c*\bar{d}$

$\bar{a}*\bar{c}*\bar{d}$, $\bar{a}*b*\bar{c}$, ... et a*c sont les monômes premiers de F tandis que $a*c*\bar{d}$ n'est pas un monôme premier car $a*c*\bar{d} \leq a*c$

Base d'une fonction booléenne

Définition

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 enlève un de ses monômes.

Exemple:

Dans l'exemple précédent, $\bar{a}*\bar{c}*\bar{d} + \bar{a}*b*\bar{c} + b*\bar{c}*d + a*c$ est une base de F. Elle n'est pas irrédondante puisque $\bar{a}*\bar{c}*\bar{d} + b*\bar{c}*d + a*c$ est encore une base de F.

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

II. REPRESENTATION DES FONCTIONS BOOLEENNES PAR FORME FACTORISEE

II.1. Définitions

Support d'une expression

Le support d'une expression d'une fonction booléenne F noté SUPP (F) est défini par:
SUPP (F) = {a / ∃ un monôme m ∈ M_F tel que a ∈ m ou $\bar{a}$ ∈ m}

Exemple:

$F = a * b + \bar{b} * c \Rightarrow SUPP(F) = \{a, b, c\}$

Produit algébrique et booléen

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

$F * G = \sum m_i m'_j$ pour i= 1...n et pour j = 1...m

Exemple:

F = a+c

$G = b + \bar{d}$

$F * G = a * b + a * \bar{d} + c * b + c * \bar{d}$

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 ($a * \bar{a} = 0$; $a * a = a$)

Exemple:

$F' = a * c + b * \bar{e}$

$G' = a * e$

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

$F' * G' = a * c * a * e + a * b * \bar{e} * e = a * c * e + 0 = a * c * e$

Diviseur algébrique et booléen

D est un diviseur algébrique de F si:

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

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

Le quotient H et le reste R sont uniques.

Exemple:

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

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

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

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

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

Exemple:

F = a*b+c+d

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

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

Facteur algébrique et booléen

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

G est un facteur booléen de F si F = G*H, avec H une expression booléenne et G*H le produit booléen de G par H.

Substitution algébrique

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

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

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

Exemple:

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

$G = a + b + c$

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

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

Expression libre

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

Exemples:

a*b + a*c n'est pas libre car a est un facteur algébrique

a*b + c est libre

a*b n'est pas libre car a et b sont des facteurs algébriques triviaux

Remarque: un monôme n'est pas une expression libre

Noyaux algébriques

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

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

- Un noyau K est dit un noyau de degré 0 s'il n'accepte comme noyau que lui-même.
- Un noyau K est de degré n s'il a au moins un noyau de degré (n-1) mais aucun noyau de degré supérieur ou égal à n excepté lui-même.

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

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

Remarques:

- On appelle K noyau global s'il est noyau de plusieurs fonctions booléennes; un noyau local est noyau d'une seule fonction booléenne.
- Un noyau local peut avoir plusieurs conoyaux associés.
- On peut remarquer qu'un noyau est formé de plus d'un monôme car un monôme n'est pas une expression libre.
- Si F est une expression polynomiale libre, elle est elle-même un noyau algébrique et son conoyau associé est égale à 1.

Exemple:

Soit F_1 une fonction dont l'expression polynomiale est:

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

Les couples conoyaux/noyaux de F_1 sont

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

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

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

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

Portée d'un noyau:

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

K: est un noyau de l'expression F.

C: est le conoyau associé au noyau K.

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

Exemple:

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

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

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

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

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

Compatibilité algébrique

Soient deux couples de conoyaux/noyaux (C, K) et (C', K') d'une expression booléenne F. Soit $\mathfrak{E}$ (resp. $\mathfrak{E}'$) l'ensemble des monômes de C*K (resp. C'*K') écrit sous forme polynomiale.

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

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

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'

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

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

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

II.2 Factorisation

II.2.1 Objectif de la factorisation

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

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

La première implantation s'appelle une implantation 2 couches (figure 2.1). La première couche réalise les monômes des variables d'entrées au moyen de portes ET, et la deuxième couche réalise la fonction somme de monômes au moyen de portes OU.

Une implantation plus générale utilise plusieurs couches de logique. Elle est fondée sur des expressions booléennes factorisées. Cette forme factorisée est ensuite décomposée en sous-fonctions correspondant à des éléments de bibliothèque appelés "cellule standard".

Exemple:

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

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

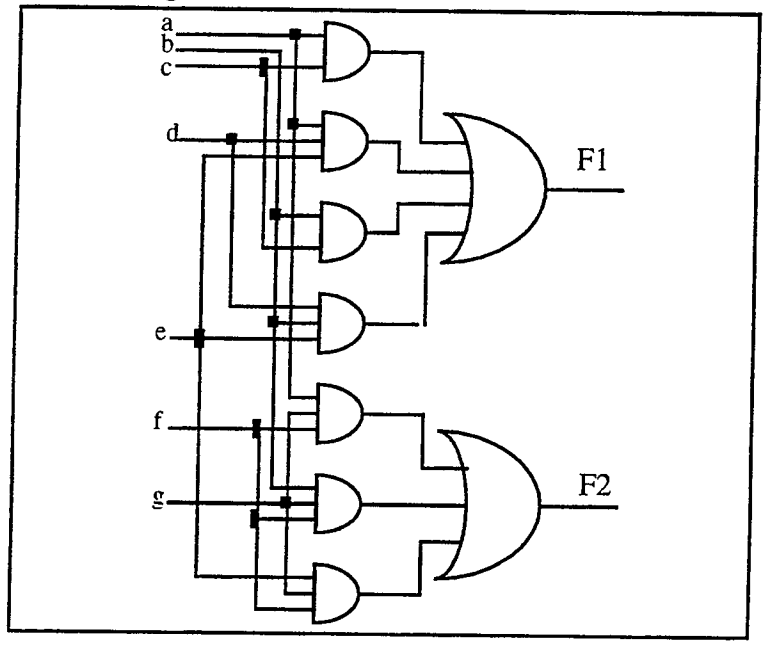


Figure 2.1 Implantation deux couches

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

$SF = a + b$

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

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

Une implantation multicouche utilisant uniquement des cellules réalisant des ET et OU booléens est donnée dans la figure 2.2.

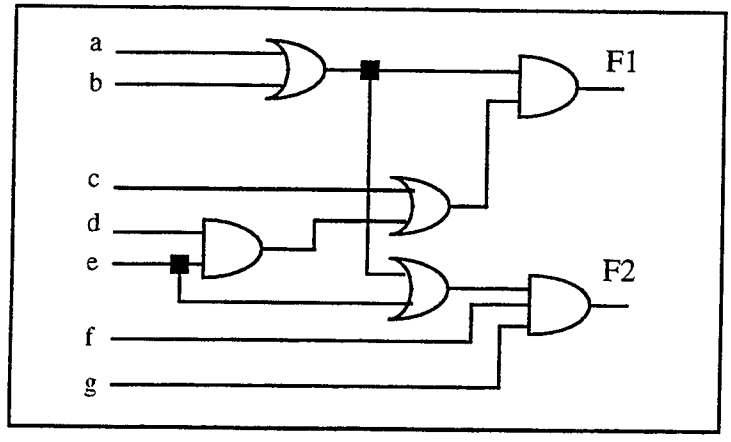


Figure 2.2 Implantation multicouche

II.2.2 Principe de la factorisation

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

On peut énumérer essentiellement deux types de factorisations:

- Factorisation algébrique utilisant le produit algébrique (noyau algébrique)
- Factorisation booléenne utilisant le produit booléen (facteur ou noyau booléen)

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

Exemple:

Soient les deux fonctions booléennes suivantes:

$$F1 = a*c*d*g + b*c*d + \bar{e}*c*d$$

$$F2 = a*b*f*g + b*c*f + \bar{e}*b*f$$

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

Les noyaux algébriques sont les suivants:

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

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

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

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

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

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

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

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

L'expression $(a*g + b*c + \bar{e})$ est commune à F1 et à F2, et devient une sous-fonction commune notée SF

F1 = c*d*SF

F2 = b*f*SF

$SF = (a*g + b*c + \bar{e})$

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

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

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

I. Génération des candidats diviseurs.

II. Tanqu'il y a des candidats diviseurs

- Sélection des candidats de gain maximal en terme de nombre de littéraux.
- Division par les candidats sélectionnés.

Fin Tan que

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

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

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

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

II.3 Génération des candidats diviseurs

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

- Noyaux algébriques et éventuellement leurs intersections.
- Monômes ou parties de monômes communs pour un ensemble de fonctions.

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

II.3.1 Génération des noyaux algébriques

L'algorithme proposé pour la génération des noyaux est celui présenté dans [Bra82, Bra87]. Cet algorithme est fondé sur l'ordonnancement des littéraux d'une fonction booléenne, puis la mise en facteur des littéraux un par un, afin de trouver les différents couples conoyaux/noyaux.

L'algorithme de génération des noyaux est le suivant:

```
Noyaux (F)
    début
        m ← le monôme, de plus grand degré, facteur algébrique de F;
        noyauxList rnoyaux (0. F/m);
        si m = 1 alors noyauxList ← noyauxList + F;
    fin
rnoyaux (j, F)
    début
        noyauxList {F};
        pour i := j+1 jusqu'à n faire
        début
            si l_i apparaît dans plus d'un monôme de F alors
            début
                m le monôme, de plus grand degré,
                facteur algébrique de F/l_i;
                si l_k n'appartient pas à m pour tous k  i alors
                    noyauxList ← noyauxList + rnoyaux (i, F/(l_i*m));
            fin;
        fin;
        retourner (noyauxList);
    fin;
```

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

Explication de l'algorithme

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

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

Le nombre de noyaux d'une expression peut croître d'une façon exponentielle par rapport au nombre de littéraux du support de celle-ci, ceci est particulièrement évidant dans le cas où la fonction a des variables symétriques, car chaque permutation des variables

symétriques donne un nouveau noyau. Cependant, dans la pratique l'ensemble des noyaux est relativement petit.

Exemple:

Soit la fonction F:

F = a*b*c*d + a*b*c*e + a*d*f*g + a*e*f*g + a*d*b*e + a*c*d*e*f + b*e*g

L'arbre généré par l'algorithme est le suivant:

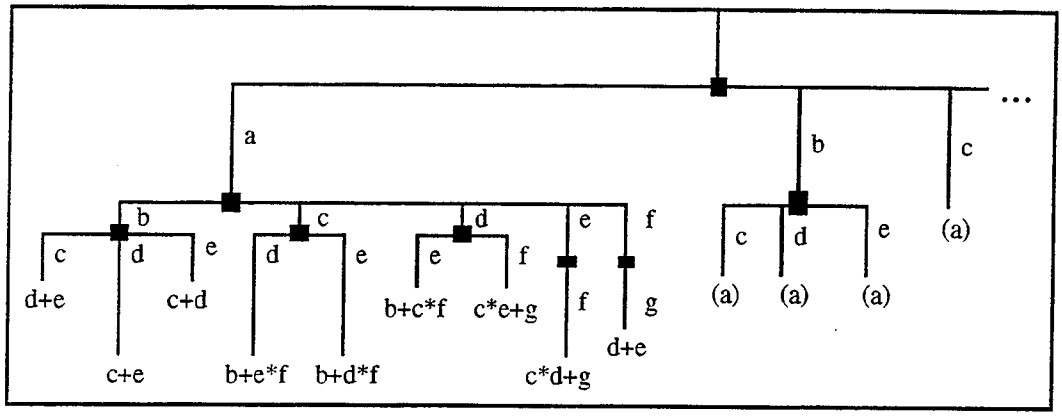


Figure 2.3 Représentation des noyaux générés par l'algorithme ci-dessus

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

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

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

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

L'algorithme utilisé est le suivant:

Début

- On ajoute à chaque monôme de chaque fonction F_i le littéral v_i;
- On construit la fonction **F**, la somme de tous les monômes (après l'ajout de v_i) de l'ensemble des fonctions booléennes;
- On calcule tous les noyaux et conoyaux associés de **F**.
- Les conoyaux où aucun littéral vi n'apparaît, correspondent à des parties communes de monômes. Si plusieurs conoyaux associés au môme noyau existe, ils forment une somme de monômes communs;

fin

Exemple:

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

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

on construit **F** de la façon suivante:

F = a*b*c*d*v_1 + d*e* v_1 + h* v_1 + a*b*c*e*v_2 + d*e* v_2 + h* v_2

Les noyaux et les conoyaux associés où v1 et v2 n'apparaissent pas sont:

{a*b*c*(d* v_1 + e* v_2), d*e*(v_1+ v_2), h*(v_1+ v_2)}

a*b*c est une partie de monôme commun à F1 et F2

d*e et h sont deux conoyaux associés au même noyau (v_1 + v_2) cela veut dire que (d*e + h) forme une somme de monômes communs.

II.4 Calcul du gain d'un candidat diviseur

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

II.4.1 Calcul de gain pour les noyaux locaux:

Le noyau apparaît dans une seule fonction booléenne, le gain défini dans le paragraphe II.2.2 du chapitre 2 est donc:

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

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

Exemple:

Soit F = a*b*c + a*b*d + a*b*e*f + g + h*c + h*d + h*e*f.

Le nombre total de littéraux est 18

Le noyau K = (c + d + e*f) a des conoyaux associés: a*b et h (m=2). F peut être réécrite:

F' = a*b * (c+d+e*f) + g +h*(c+d+e*f)

Le gain associé au noyau K est calculé comme suit:

Gain (K) = (3-1)*2 + (3-1)*1 + (2-1)*4 = 4+2+4 = 10

Dans F', le noyau (c+d+e*f) est exprimé une seule fois. Le nombre total de littéraux est 8.

II.4.2 Calcul de gain pour les noyaux globaux

Le noyau K est un noyau de plusieurs fonctions booléennes. Si ce noyau apparaît p fois dans l'ensemble des fonctions, il sera remplacé par un littéral dans chaque fonction. Le gain en nombre de littéraux associé est le suivant:

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

Ce gain sera ajouté dans le gain associé à un noyau global.

$F_{k=1...n}$ sont les fonctions où apparaît K.

Soit $(C^i)_{i=1...m_k}$ les conoyaux associés au noyau K dans la $k_{ième}$ fonction.

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

$$\text{Gain (K)} = \sum_{k=1}^{n} \{ \sum_{i=1}^{m_k} ((\text{NbMon(K)}-1)*\text{NbLit}(C^i)) + (m_k-1)*\text{NbLit(K)}\} + (p-1)*(\text{NbLit(K)}) - p$$

Exemple:

$F_1 = a*b*c + a*b*d + a*e*f + g + h*c + h*d$

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

Le nombre total de littéraux est 20

$K_0 = c+d$ noyau global conoyaux: a*b et h pour F_1, $m_1=2$ et conoyau a pour F_2, $m_2=1$.

$K_1 = b*c + b*d + e*f$ noyau local conoyau a (F1)

$F_1 = a*b*(c+d) + a*e*f + g + h*(c+d)$

$F_2 = a*(c+d) + g*h$

Le nombre total de littéraux est 14 pour K_0.

Gain $(K_0) = ((2-1).2) + (2-1).1)) + (2-1).2 + (2-1).1 + (2-1).(2) - 2 = 2+1+2+1+2-2 = 6$

Remarquons que (c+d) est remplacé par un littéral pour chaque fonction et n'apparaît qu'une fois.

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

II.4.3 Gain des monômes ou parties de monômes communs

On appelle:

NbLit (m): le nombre de littéraux du monôme;

NbOcc (m): le nombre d'occurrences du monôme m.

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

$$\text{Gain(m)} = (\text{NbOcc(m)} - 1).\text{NbLit(m)} - \text{NbOcc(m)}$$

Exemple:

F1 = a*b*c*d + d*e + h

F2 = a*b*c*e + d*e + h

m1 = a*b*c est une partie commune de monôme dont le gain associé est 1

m2 = d*e est un monôme commun, son gain est égal à 0.

Gain d'une sous-fonction

Au cours de la phase de factorisation, les candidats diviseurs ont donné lieu à des sous-fonctions dites sous-fonctions partagées.

On appelle gain associé à une sous-fonction SF partagée le nombre de littéraux gagné en remplaçant l'expression booléenne correspondant à cette sous-fonction par une variable étiquetée SF dans la fonction booléenne globale.

On appelle:

NbLit (SF): le nombre de littéraux de la sous-fonction SF.

NbOcc (SF): le nombre d'occurrences de la sous-fonction SF dans la fonction booléenne globale.

La formule du gain associé à une sous-fonction SF est la suivante:

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

Exemple:

F1 = a*b*c*d + d*e + h

F2 = a*b*c*e + d*e + h

=> SF1 = a*b*c a le gain de 1

SF2 = d*e n'a pas de gain.

II.5 Sélection des candidats diviseurs

Cette étape permet de choisir dans l'ensemble des candidats diviseurs déterminés lors de la phase de génération celui qui impliquera un gain maximal de logique, exprimé en nombre de littéraux. Mais ce choix doit tenir compte de l'incompatibilité algébrique éventuelle entre les différents candidats, ce qui rend complexe le problème de sélection de diviseurs.
Une première méthode considère globalement l'ensemble des candidats pour tenir compte de l'impact créé par le choix de l'un d'entre eux sur l'ensemble des candidats restants. Cette méthode aboutit à l'extraction d'un ensemble de candidats mutuellement compatibles et de gain maximal. Ce problème peut être modélisé de la façon suivante:
On construit le graphe non orienté G qui a pour sommets l'ensemble V. Chaque sommet v de V correspond à un diviseur d. On donne un poids à chaque sommet v qui correspond au gain associé au diviseur d et deux sommets v1 et v2 sont liés par une arête si les candidats diviseurs correspondants sont incompatibles.
Trouver l'ensemble de tous des diviseurs de gain maximum revient exactement à résoudre le problème de la recherche du stable de poids maximum dans G. Ce problème est NP complet et donc la recherche d'un ensemble de candidats diviseurs compatibles deux à deux et de gain maximal ne peut pas être résolu par un algorithme polynomiale.
Une heuristique, appelée principe de la couverture rectangulaire, fondée sur la couverture disjointe des cellules d'une matrice est proposée dans [BRA87]. Cette méthode consiste à construire une matrice M où chaque ligne correspond à un unique noyau et où chaque colonne est associée à un unique monôme d'un noyau de la fonction. Ainsi un monôme de la fonction est une case de la matrice.
Bien que la formulation de cette méthode semble intéressante, il s'avère que la complexité actuelle des circuits ne permet pas toujours de l'appliquer de manière optimale. Par exemple, une expression qui comporte 100 monômes pour 10 variables peut facilement générer un million de rectangles. Dans ce cas, le choix de l'algorithme de synthèse peut dépendre de la complexité des expressions booléennes et permet à l'utilisateur d'utiliser une méthode ou une autre, ce qui lui procure un bon compromis temps de calcul / qualité des résultats.
La deuxième méthode choisit le candidat diviseur permettant d'obtenir un gain local maximal (algorithme " greedy ") sans vérifier la comptabilité algébrique avec les autres.

II.6 Division et substitution algébriques

II.6.1 Algorithme de la division algébrique

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

Exemple:

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

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

noyau	conoyau	gain
$K1 = a.*b*c+a*\overline{b}*\overline{c}+a*d+b*c*e*f*g+\overline{b}*\overline{c}*e*f*g$	1	0
$K2 = b*c+\overline{b}*\overline{c}+d$	a	2
$K3 = a+e*f*g$	b*c	2
$K4 = a+e*f*g$	$\overline{b}*\overline{c}$	2
$K5 = b*c+\overline{b}*\overline{c}$	e*f*g	3

L'application de la méthode de division algébrique sur le noyau de plus grand gain (K5) donne le résultat suivant:

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

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

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

Principe de la division algébrique:

Soient deux fonctions polynomiales F et D. Le principe de la division algébrique est la recherche du quotient H de la fonction D dans l'expression de F. F s'écrit alors: F = D*H + R, R est le reste tel que M_R est le sous-ensemble de monômes de M_F non impliqués dans le produit algébrique D*H.

La division s'effectue comme suit:

* Pour tout monôme mi du diviseur D

- Faire la somme des monômes de F multiples de mi.
- Diviser cette somme par mi.
- soit $V_{m_i} = \sum m_j$ le quotient de cette division.

* L'intersection des $M_{V_{mi}}$ constitue l'ensemble des monômes du quotient H de la division de F par D.

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

L'implantation algorithmique de cette division algébrique a été introduite par Braton [BRA87B]

Division_Algébrique (F, D)

début

U = restriction de F aux littéraux de D

V = restriction de F aux littéraux qui ne sont pas dans D

/* $u_j v_j$ est le $j^{ième}$ monôme de F */

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

/* Recherche du quotient de la division de F par chaque monôme d_i */

/* d_i est le $i^{ème}$ monôme de D et V_i est l'ensemble obtenu en divisant F par u_i */

$M_H = M_{Vi}$

/* H est le quotient de F par D */

$M_R = M_F - M_{D*H}$

/* R est le reste de la division de F par D */

retourner (H, R)

fin

Exemple:

Soit une fonctions de 5 monômes:

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

Doit D une fonction de 2 monômes D = a*b+c*d.

La restriction de F aux littéraux de D est:

U = a*b+c*d+a*b+c*d+a*b

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

V = e*f+e*f+g*h+g*h+k

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

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

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

$M_R = a*b*k$

retourner (e*f+gh, a*b*k)

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

F = (a*d+c*d)*(e*f+g*h)

Remarques:

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

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

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

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

II.6.2 Algorithme de la substitution algébrique

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

début

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

fin

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

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

avec x = D

Remarques

Cette substitution fait appel à 2 divisions: k*O(n*m) (k est une constante) et un calcul de complément. Le coût du complément est difficile à évaluer. Le pire des cas est celui des OUs exclusifs où la couverture à 1 de la fonction contient $2^{(v-1)}$ points où v est le nombre de variables, donc exponentielle.

Cependant, en général dans la pratique le coût est acceptable.

Exemple:

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

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

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

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

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

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

$x = a+b+c$

II.7 Représentation graphique de fonctions booléennes factorisées

II.7.1 Arbre de factorisation

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

Exemple:

Soit une équation booléenne factorisée F:

F = abc +(!a+!b)(!e+!f)+!c!d(i+jk)+!g(!i+!k)

L'arborescence correspondante est donnée dans la figure 2.4

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

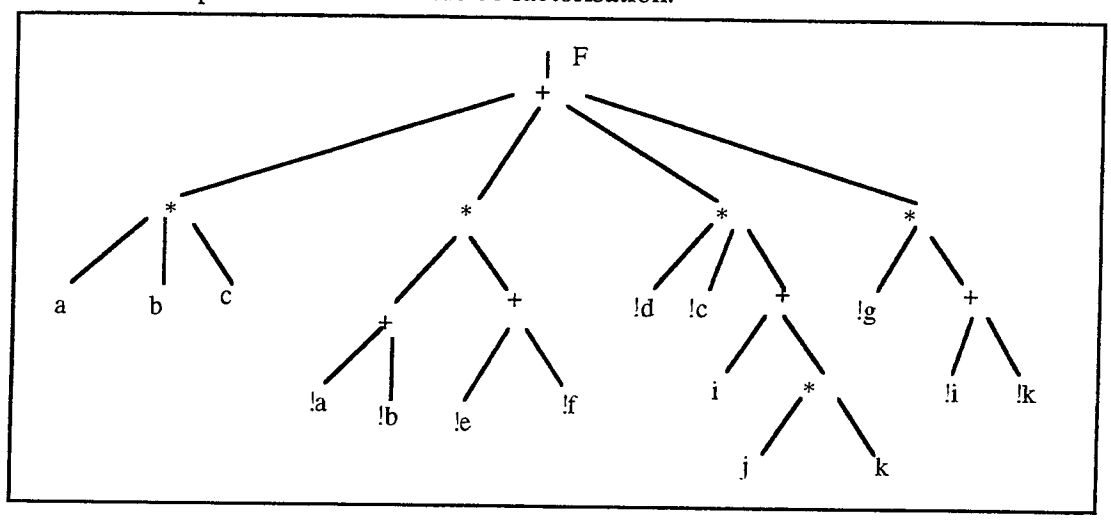


Figure 2.4 Arbre de factorisation de fonction booléenne

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

• Les nœuds prédécesseurs d'un nœud de l'arbre constituent le cône prédécesseur de ce nœud. Le cône prédécesseur d'un nœud est montré en figure 2.5

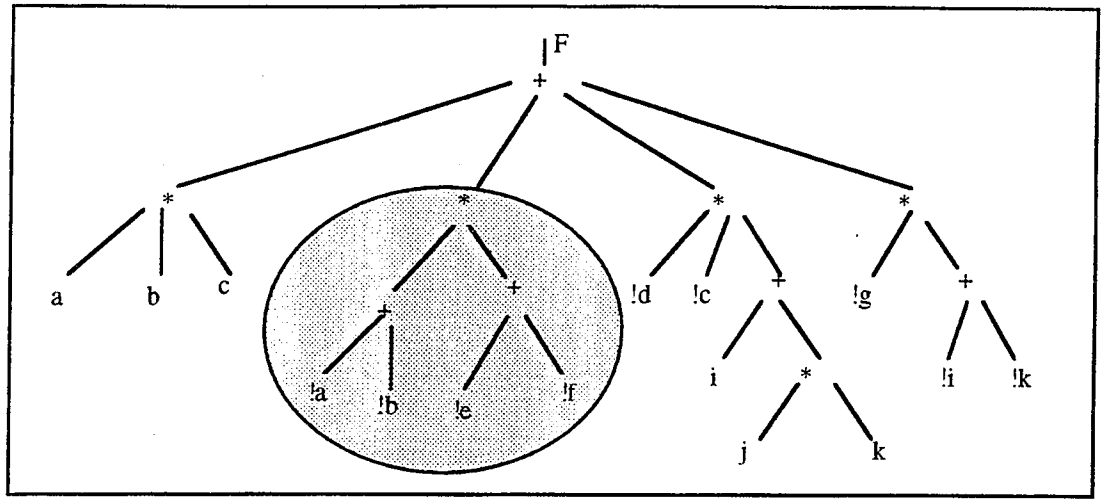


Figure 2.5 Cône prédécesseur d'un nœud

II.7.2 Arborescence hierarchisée

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

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

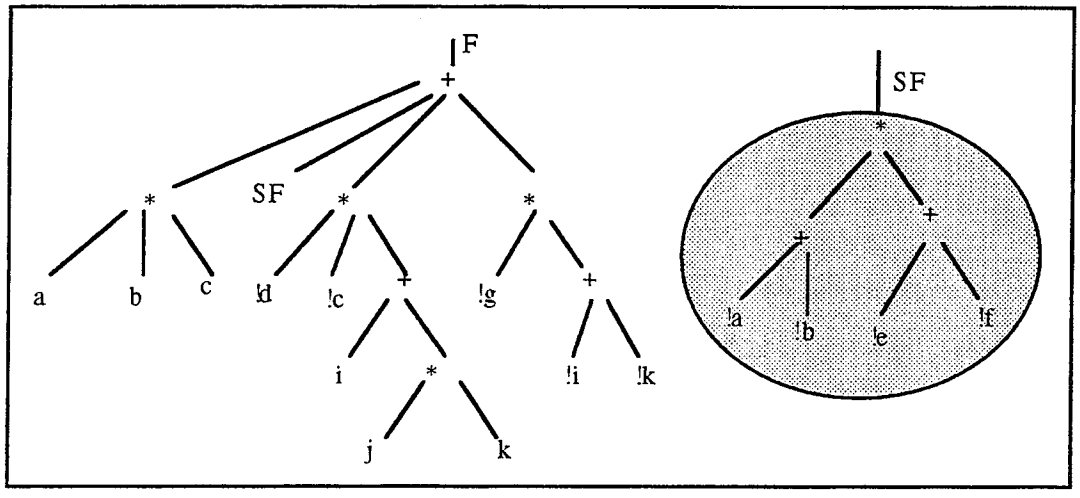


Figure 2.6 Arborescence hierarchisée

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

II.7.3 Sous-fonction partagée et graphe orienté hierarchique

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

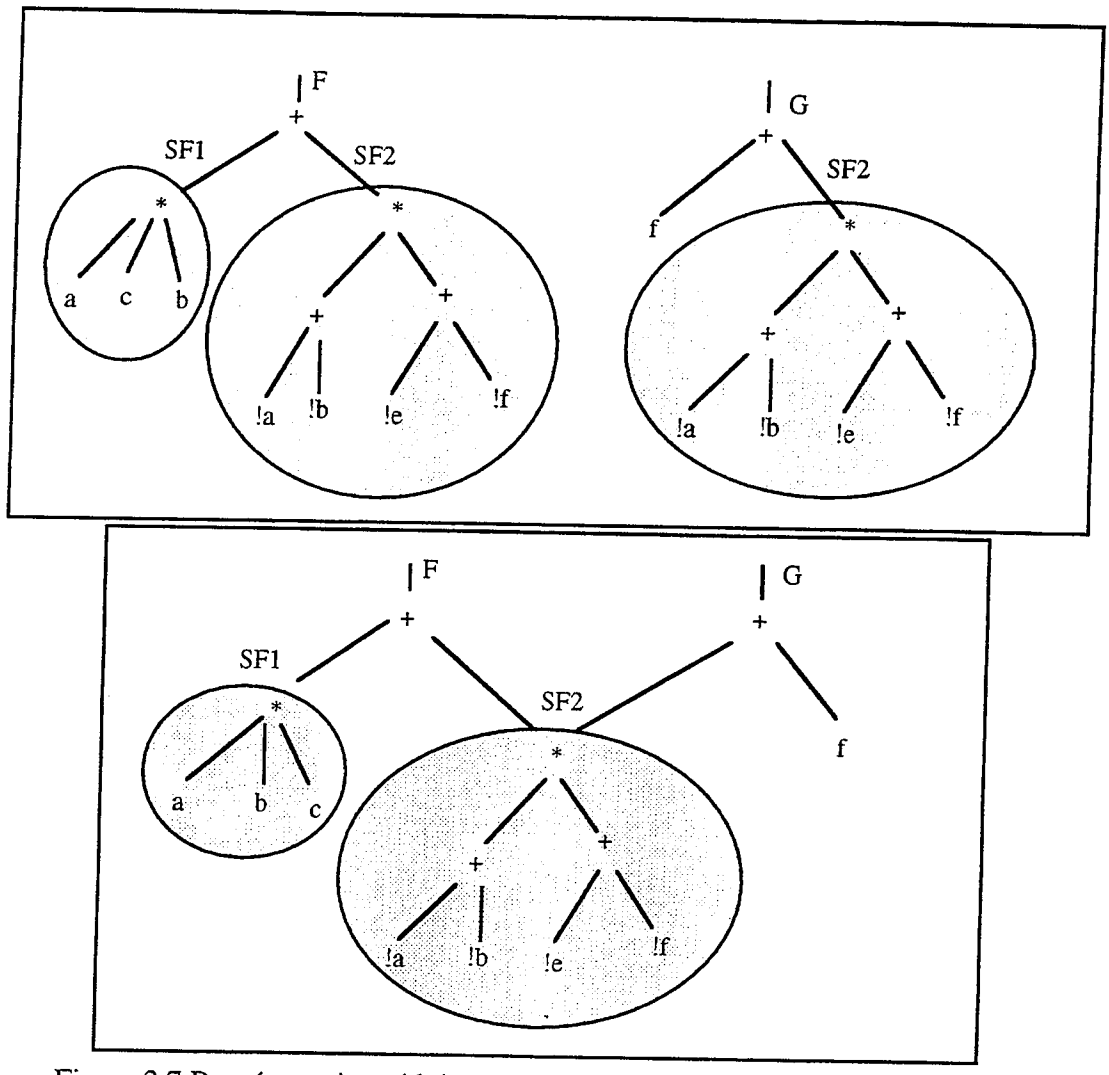


Figure 2.7 Représentation réduite de sous-fonctions partagées/non-partagées

Comme précédemment, un tel graphe peut être également réduit en remplaçant la sous-arborescence par une feuille étiquetée SF.

Dans la figure 2.7, SF1 est une sous-fonction non-partagée. SF2 est une sous-fonction partagée.

II.7.4 Graphe hierarchique réduit

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

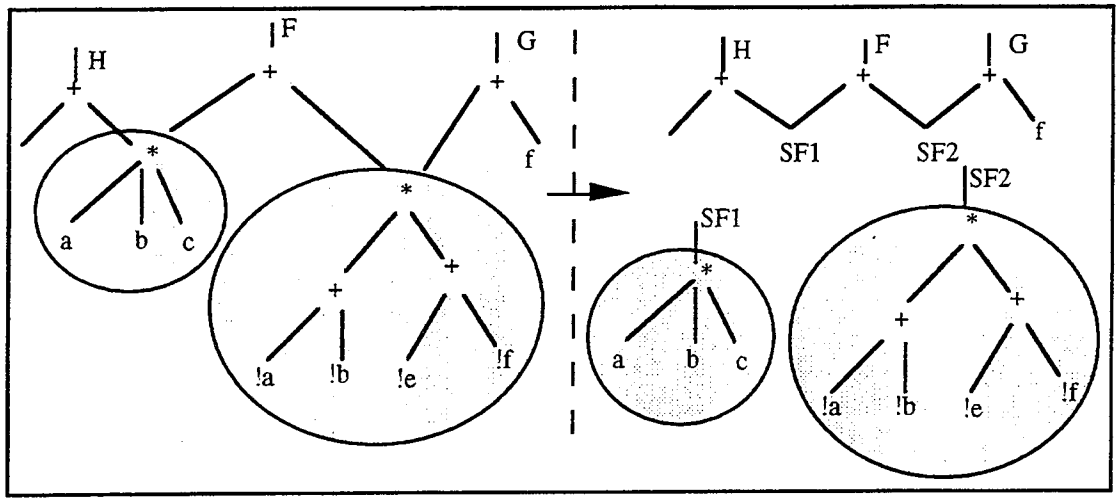


Figure 2.8 Graphe hierarchique réduit

• **Degré de hierarchie**

La hierarchie est un procédé récursif et la profondeur de la hierarchie est un paramètre de la description.

Exemple:

Dans la figure 2.9 la profondeur de la hierarchie dans la décomposition de F1 est de 4. La profondeur de F3 est de 5.

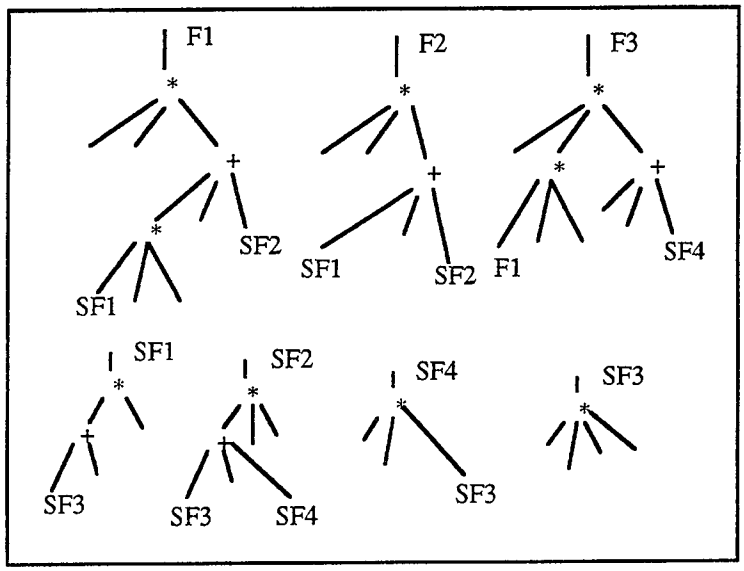


Figure 2.9 Exemple du calcul de la profondeur du graphe hierarchique

III. REINJECTION CONTROLLEE DES SOUS-FONCTIONS

Introduction

La synthèse de fonctions booléennes se fait par décomposition technologique en portes de base respectant la décomposition en sous-fonctions. Ceci veut dire que cette décomposition est faite séparément pour chaque sous-fonction.

Cette granularité a un impact important sur l'efficacité ultérieure de la décomposition technologique orientée vitesse. En effet, on ne pourra plus "traverser" des barrières créées par les sous-fonctions. Le principe de la réinjection a comme but de déplacer ces barrières pour rendre la décomposition technologique plus efficace.
La manière la plus simple pour améliorer la performance temporelle est de réinjecter le réseau total sous forme de réseau à deux couches laissant aussi toute la liberté à la décomposition technologique orientée vitesse. Cette approche peut donner de bons résultats mais elle peut augmenter considérablement la surface finale du circuit.
La réinjection sera faite d'une manière contrôlée pour profiter de l'effet de réduction de la profondeur du graphe d'imbrication tout en gardant un coût en surface acceptable.

III.1 Défintion

Soit une fonction F(a, b, c, ..., SF) exprimée en terme de ses variables et d'une sous-fonction dépendant de ces mêmes variables.
On appelle réinjection de SF dans F l'expression de F dans laquelle SF a été remplacée par son expression exprimée en terme des entrées.
Le processus de réinjection générale consiste à diminuer la profondeur du graphe d'imbrication correspondant à l'expression booléenne. Elle peut être faite à n'importe quel niveau.

Exemple :

$$\left.\begin{aligned} F &= a.b.SF1 \\ G &= \overline{c}.\overline{d}.SF1 \\ H &= SF2 \\ SF1 &= e + f.SF2 \\ SF2 &= g.h \end{aligned}\right\} \xrightarrow{\text{après la reinjection de SF1}} \left\{\begin{aligned} F &= a.b.(e + f.SF2) \\ G &= \overline{c}.\overline{d}.(e + f.SF2) \\ H &= SF2 \\ SF2 &= g.h \end{aligned}\right.$$

La figure 2.11 montre l'effet de la réinjection au niveau des arbres factorisées.

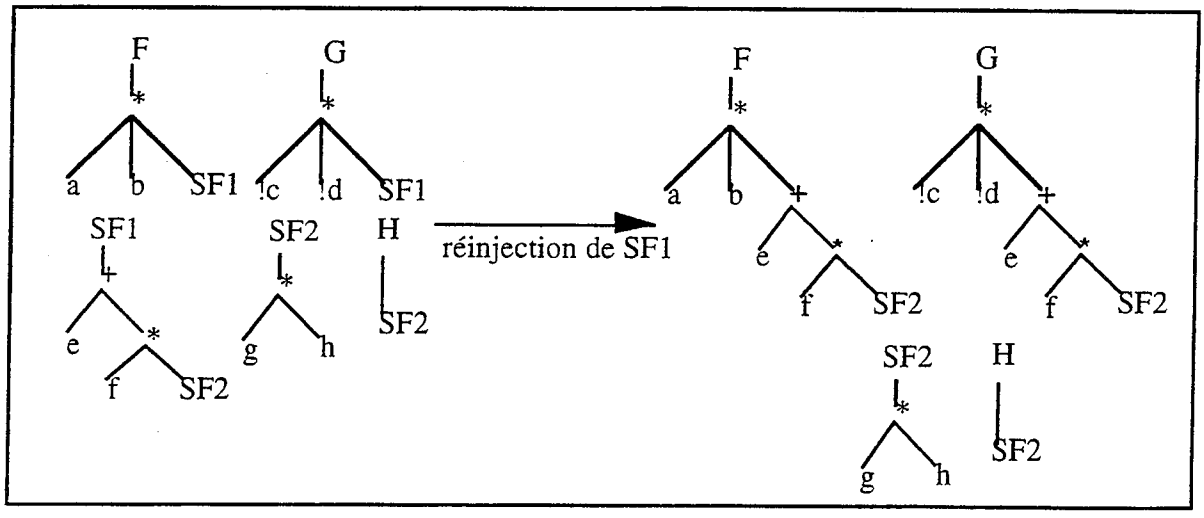


Figure 2.11 Reinjection des sous-fonctions

III.2 Approche proposée

Les critères pendant la réinjection de sous-fonctions dépendent des cibles technologiques, de l'étape où la réinjection est appliquée, de la structure et de la complexité de la fonction booléenne. Cette réinjection peut intervenir à des étapes différentes: avant, pendant et après la décomposition technologique pour améliorer les résultats.

La réinjection partielle de sous-fonction peut être faite suivant différentes heuristiques. Nous ne considérons ici que les réinjections avant décomposition technologique.

III.2.1 Réinjection totale

La réinjection totale d'un réseau booléen a pour but de réinjecter récursivement toutes les sous-fonctions. L'algorithme part des entrées.

Réinjection_Totale (Fct)

Ordonner les sous-fonctions en ordre non-décroissant de profondeur dans le graphe d'imbrication

Pour (toutes les sous-fonctions SF)

Si (SF n'est pas une sortie généralisée)

Réinjecter SF dans ses successeurs

La réinjection totale est illustrée dans la figure 2.12. La fonction booléenne F est présentée dans la figure 2.12a. La sous-fonction SF2 est réinjectée dans son successeur SF1 (figure 2.12b) et puis SF1 dans F (figure 2.12c).

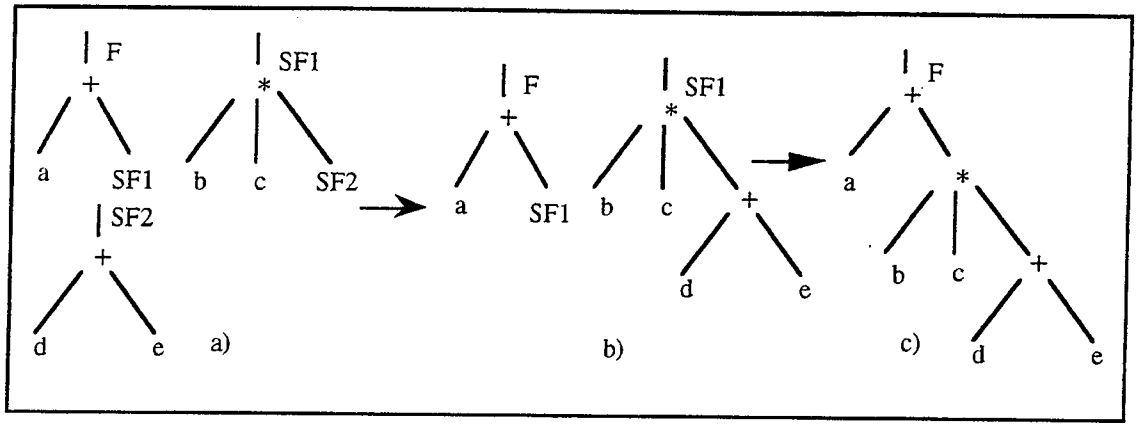


Figure 2.12 Réinjection totale

III.2.2 Réinjection des sous-fonctions non-partagées

Pour les sous-fonctions non-partagées, on peut les réinjecter pour diminuer tout d'abord la profondeur du graphe d'imbrication.

Réinjection_SF_non-partagées

Ordonner les sous-fonctions en ordre non-décroissant de profondeur dans le graphe d'imbrication

Pour (Toutes les SFs)

Si (SF n'est pas partagée)

Réinjecter SF

Exemple:

Soit

F1 = a + SF1

F2 = c + SF2

SF1 = b*c*SF2

SF2 = d*e

La sous-fonction SF1 est réinjectée car elle n'est pas partagée (Figure 2.13)

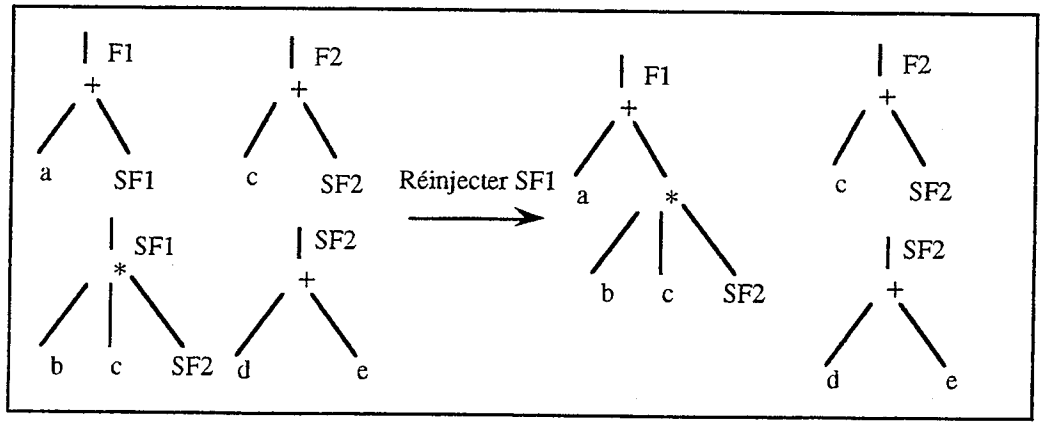


Figure 2.13 Réinjection des sous-fonctions non-partagées

III.2.3 Réinjection des sous-fonctions partagées en tenant compte du gain

La réinjection peut être faite en contrôlant le gain en terme de nombre de littéraux pour adapter à une technologie donnée.

Si le gain d'une sous-fonction est petit (dépendant de technologie), il vaut mieux la réinjecter pour aider l'étape de décomposition technologique.

Réinjection_SF_en_tenant_compte_du_gain

Ordonner les sous-fonctions en ordre non-décroissant de profondeur dans le graphe d'imbrication

Pour (Toutes les SFs)

Si (Gain (SF) < k)

Réinjecter SF

Exemple:

Dans la figure 2.14

F1 = a + SF

F2 = SF + c + !d

Supposons que k = 2.

Gain (SF) = (2-1)*2 - 2 = 0 < 2 = k ⇒ SF sera réinjectée dans F1 et F2.

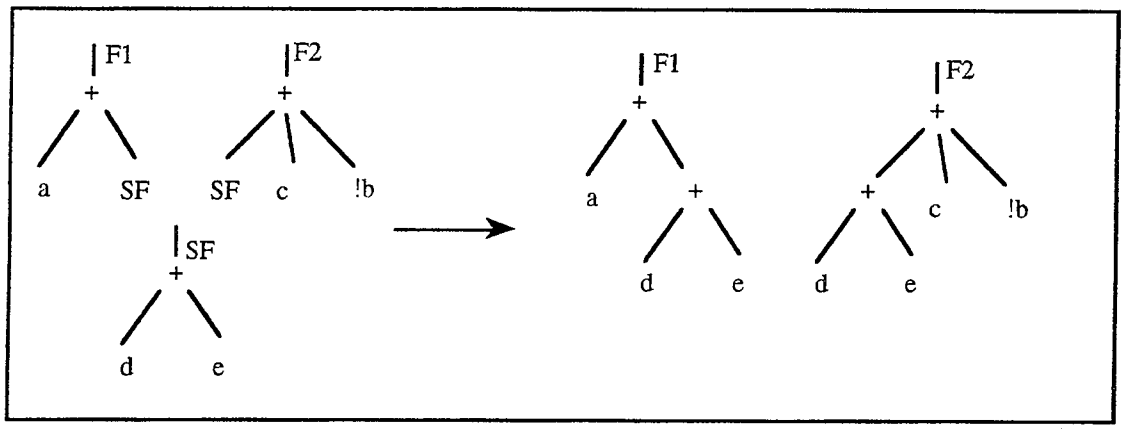


Figure 2.14 Réinjection en tenant compte du gain

III.2.4 Réinjection spécifique aux cibles technologiques

En vue de la synthèse logique, la surface des FPGAs dépend (plus ou moins directement) soit du nombre de littéraux (comme dans les technologies de cellules standard, les boîtiers Actel, QuickLogic ...), soit du nombre de variables (comme dans les technologies à base de LUT telles que Xilinx, AT&T, Flex8000...).

La réinjection doit être faite différemment pour des cibles technologiques différentes. Nous allons présenter deux types de réinjection pour les deux grandes familles de FPGAs citées ci-dessus.

a) Réinjection contrôlée pour les FPGAs dont la surface dépend essentiellement du nombre de littéraux

Le critère utilisé pour décider la réinjection d'une sous-fonction dans une autre est le nombre de littéraux de la sous-fonction résultat après la réinjection.

L'algorithme de cette réinjection est le suivant:

Réinjection_Pour_Litéraux

Ordonner les sous-fonctions en ordre non-décroissant de profondeur dans le graphe d'imbrication

Pour (Toutes les sous-fonctions SF)

Pour ($SF_i \in$ successeurs (SF))

SF_i ' = Réinjectant SF dans SF_i;

Si (Nombre de littéraux de SF_i ' < k)

$SF_i = SF_i$ ';

Sinon

Libérer SF_i ';

Exemple:

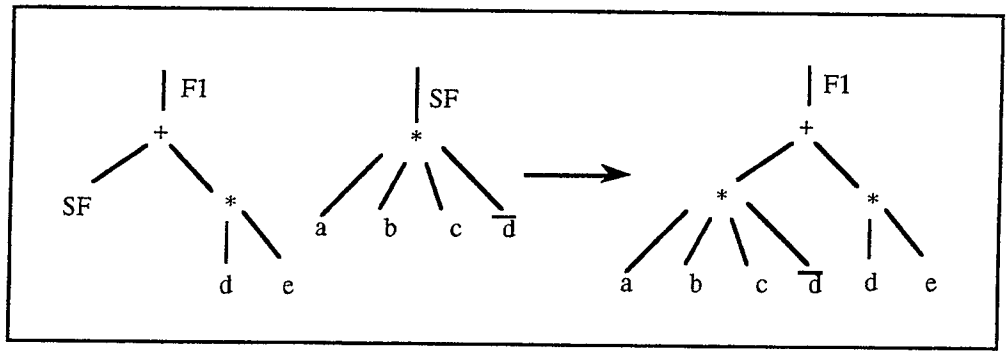


Figure 2.15 Réinjection_Pour_Litéraux

Dans la figure 2.15 nous avons:

F1 = d*e + SF

SF = a*b*c* $\overline{d}$

Nous prenons l'hypothèse que: k = 7

La réinjection de SF dans SF1 donne F1'. Nous avons:

Nombre de litéraux de F1' = 3+4-1 = 6 < 7 $\Rightarrow$ La réinjection est acceptée.

b) Réinjection contrôlée pour les FPGAs à base de LUT

L'algorithme est similaire à l'algorithme pour les FPGAs dont la surface dépend du nombre de littéraux en remplaçant le nombre de littéraux par celui des variables comme suit:

Réinjection_Pour_Variables

Ordonner les sous-fonctions en ordre non-décroissant de profondeur dans le graphe d'imbrication

Pour (Toutes les sous-fonctions SF)
Pour ($SF_i \in$ successeurs (SF))
SF_i ' = Réinjectant SF dans SF_i;
Si (Nombre de variables de SF_i ' < k)
$SF_i = SF_i$ ';
Sinon
Libérer SF_i ';

Exemple: Dans la figure 2.16 prendre l'hypothèse de k = 9. Nous avons:
/sup(SF3) ∪ sup(SF2)/ -1 = 5 < 9 = k ⇒ SF3 sera réinjectée dans SF2, donnant SF2'.

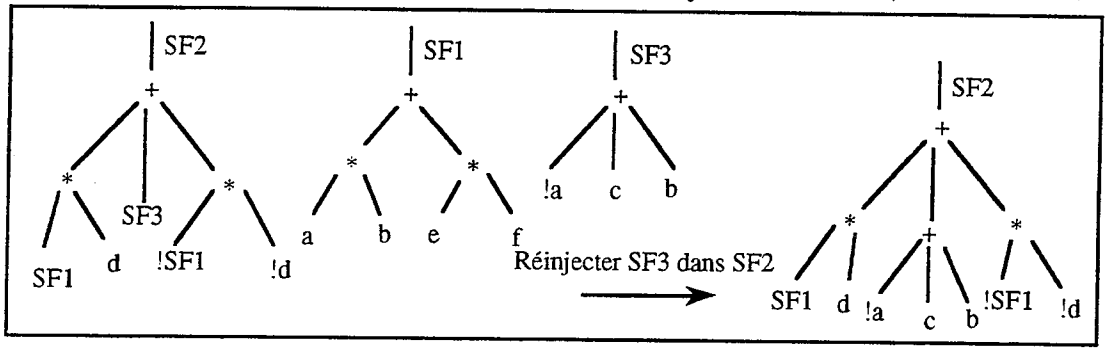


Figure 2.16 Réinjection contrôlée pour les FPGAs à base de LUT

On peut combiner ce type de réinjection avec d'autres types de réinjection présentés précédement pour profiter des effets de réinjection. Par exemple, on peut réinjecter les sous-fonctions non-partagées. la sous-fonction résultat devant avoir le nombre de variables inférieur à un certain nombre k.

Exemple:

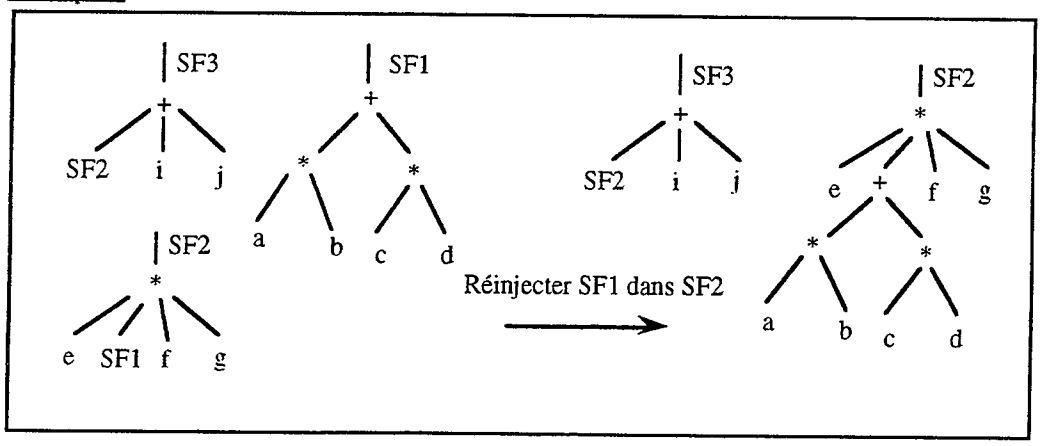


Figure 2.17 Exemple de combinaison de types de réinjection

La figure 2.17 illustre la combinaison des types de réinjection des sous-fonctions non-partagées et pour les FPGAs à base de LUT. Supposons que k est égale à 9 pour cet exemple.

Dans la figure 2.17 SF1 sera réinjectée dans SF2 pour donner SF2' car elle vérifie les conditions:

- SF1 n'est pas partagée.
- Le nombre de variables de la sous-fonction résultat SF2' est de $7 < 9 = k$.

Ensuite. SF2' sera examinée. SF2' n'est pas partagée mais elle ne vérifie pas la condition:

- Le nombre de variables de la sous-fonction résultat est de $10 > 9 = k$.

Donc. elle ne peut plus être réinjectée dans SF3.

CHAPITRE 3

OPTIMISATION TEMPORELLE DES RESEAUX PROGRAMMABLES A BASE DE LUT

INTRODUCTION

Un réseau de portes est composé de blocs combinatoires et de points mémoires (bascules). Comme les bascules sont déjà bien définies dans chaque technologie, les parties, sur lesquelles les techniques d'optimisation sont appliquées, sont limitées aux parties combinatoires. Le but de l'optimisation temporelle est la minimisation des chemins les plus longs dans une zone (combinatoire) définie par l'utilisateur.

Le délai d'un chemin traversant des portes logiques dépend essentiellement de trois éléments suivants:
- Le délai interne des blocs logiques.
- Le nombre de blocs logiques sur le chemin considéré.
- Le délai causé par l'interconnexion des blocs logiques sur le chemin.

Le délai interne du bloc logique dépend directement de la performance du boîtier. L'optimisation temporelle ne peut pas intervenir sur ce délai. Pour obtenir une meilleure performance, on peut utiliser tout simplement des boîtiers à haute vitesse, dont le délai interne du bloc logique est petit, mais à un coût beaucoup plus élevé.

Les buts de l'optimisation temporelle restent donc la minimisation du nombre de blocs logiques sur les chemins critiques et des délais d'interconnexion entre ces blocs.
La minimisation du nombre de blocs logiques sur le chemin critique implique souvent l'abaissement du délai de ce chemin mais ce n'est en général pas suffisant. Le délai d'interconnexion contribue à une part très importante dans le délai total, surtout dans les boîtiers des technologies à base de SRAM comme Xilinx, AT&T. Il doit être modélisé et traité correctement pendant l'optimisation temporelle.

La minimisation du délai des chemins critiques du circuit est souvent accompagnée par une augmentation de surface. Quand la surface est trop augmentée, la ressource de routage peut être saturée. Ceci provoque une augmentation considérable du délai d'interconnexion. Le gain en terme du nombre des blocs logiques sur les chemins critiques peut être neutralisé par l'augmentation du délai d'interconnexion.

C'est pourquoi une optimisation du délai de la zone considérée en surveillant l'augmentation de la surface est l'un des objectifs que nous allons traîter dans cette thèse.

Avant de passer à notre approche qui sera présentée dans la deuxième partie de ce chapitre, nous faisons un résumé des approches proposées récemment dans la littérature.

ETAT DE L'ART

Les algorithmes d'optimisation de performance temporelle des FPGAs à base de LUT ont été étudiés et améliorés ces dernières années. La plupart d'entre eux visent à réduire la profondeur en cellules du réseau booléen en maintenant une surface raisonnable [CONG92], [CONG93], [MUR91A]. Certains algorithmes essayent de prendre en compte une estimation prédictive des délais d'interconnexion dû à la sortance des signaux lors de la phase d'optimisation. D'autres utilisent une méthodologie d'amélioration itérative entre la synthèse logique et l'outil de placement- routage de façon à obtenir des résultats plus réalistes.

Nous allons présenter les approches d'optimisation de performance temporelle pour des FPGAs à base de LUT publiées pendant les dernières années. Puis, nous en tirons des remarques ayant motivé notre approche qui sera proposée dans une section ultérieure.

Chortle-d [FRA91B]

Cette approche prend la profondeur en nombre de LUT pour critère essentiel d'optimisation. Elle est composée de 4 étapes:

* Construction de couches ("strata") contenant des nœuds du réseau booléen de même profondeur.
* Utilisation d'une méthode de regroupement du type "bin-packing" pour chaque couche afin de minimiser le nombre de blocs ou "bins" dans chaque couche.
* Décomposition finale en LUT
* Amélioration des résultats en étudiant des chemins reconvergeant et en répliquant une partie de la logique. L'augmentation de surface d'une décomposition en LUT ayant comme objectif l'optimisation temporelle sera réduite par un post-traitement.

Cette approche minimise la profondeur du réseau booléen et le temps CPU reste réduit. On peut noter deux inconvénients:

* L'approche décompose le réseau booléen en arbres distincts, bien qu'elle garantisse l'optimalité pour chaque arbre (optimalité locale) elle n'est pas optimale pour le réseau complet.
* Malgré le post-traitement après la décomposition orientée vitesse, la surface reste encore très importante. Par conséquent, le délai d'interconnexion est augmenté si bien que le gain réel en chemin critique obtenu en ne considérant que la profondeur en portes ou cellules est réduit voire annulé.

MIS-PGA New [MUR91A]

En constatant que le délai introduit par les interconnexions est important, l'algorithme de MIS-FPGA essaie de minimiser non seulement la profondeur du réseau booléen comme

précédemment mais cherche à contrôler à la fois le nombre de nœuds (correspondant aux cellules logiques) et d'arêtes (correspondant aux interconnexions).
L'optimisation se fait en 2 étapes principales:
* A partir d'un réseau booléen en portes à 2 entrées, la technique consiste à "réinjecter" des portes dans leur porte successeur en veillant à ne créer que des fonctions "acceptables" (LUT-m), la fonction objective étant à la fois le nombre de nœuds, d'arêtes et la profondeur globale.
* Boucle de resynthèse combine la décomposition Roth-Karp [ROT62], la réinjection partielle, le placement et le routage pour trouver le meilleur résultat.

Remarques:

* Le résultat de l'optimisation est bon mais le temps de calcul est trop long car l'algorithme calcule toutes les combinaisons possibles pour trouver la meilleure solution.
* En remarquant que le délai d'interconnexion contribue pour une part importante du délai final, le modèle utilisé est le modèle d'Elmore [LIN84], [CHA89]. Pour que ce type de délai soit significatif, le placement et le routage doivent être faits avant la resynthèse. Or dans cette approche, le placement et le routage sont simulés ou prédits. En conséquence, le résultat n'est donc pas forcément fiable. Remarquons que le temps de calcul est très long essentiellement à cause de l'utilisation de ce modèle de délai.

MIS-PGA perd 2,9 % en terme de profondeur par rapport à Chortle-d mais la surface est réduite de 60%. Cela donne un gain d'environ 16% en termes de délai après le placement et le routage par rapport à Chortle-d.

DAG-Map [CONG92A]

DAG-Map vise à minimiser la profondeur du réseau tout en diminuant le nombre de cellules logiques. Il est composé de 2 étapes:

* Etape 1: Transformation du réseau booléen en un réseau de portes à 2 entrées par l'algorithme DMIG (Decompose Multi Input Gate) en tenant compte de la profondeur du réseau complet. Cela permet d'avoir un réseau de profondeur minimale avant la décomposition technologique.
* Etape 2: Décomposition technologique se déroulant en 2 étapes:
  - Affecter à chaque nœud une fonction de coût correspondant au délai à ce nœud.
  - Créer des LUTs en descendant des sorties primaires vers les entrées à l'aide des fonctions de coût calculées précédemment.

La surface après la minimisation de la profondeur est réduite par un post-traitement.

FlowMap [CONG92B], [CONG94A]

FlowMap hérite des principes de DAG-Map et ajoute des phases d'optimisation supplémentaires. L'étape consiste à calculer la coupure k-faisable à profondeur minimale.

Cette approche a prouvé que le problème de l'optimisation de la profondeur est résolu en temps polynomial.

Les étapes de l'optimisation sont décrites comme suit:

+ Transformer le réseau en réseau de portes à 2 entrées par DMIG comme dans DAG-Map.

+ Affecter une fonction de coût à chaque nœud en introduisant la notion de la coupure k-faisable à profondeur minimale.

+ Décomposer technologiquement à base de coupure k-faisable et des fonctions de coût calculées ci-dessus. Pendant la décomposition, maximiser la taille de coupure pour optimiser la surface sans augmenter la profondeur.

+ Diminuer la surface par l'algorithme FlowPack sans augmenter la profondeur.

Remarques:

FlowMap est un peu meilleur que DAG-Map. Il conduit à un gain d'environ 2,4% en termes de délai et 8,6% en termes de surface ce qui fait gagner encore pour le délai de la conception après le placement et le routage.

FlowSyn [CON93A]

FlowSyn à son tour hérite des principes de FlowMap mais comprend aussi une optimisation du réseau booléen par l'utilisation de graphes de décision binaire pour représenter des fonctions booléennes. La génération des cellules LUT est identique à celle de FlowMap mais la fonction de coût est modifiée.

Dans FlowMap, chaque fois qu'une coupure a le nombre de variables supérieure à k, on incrémente une variable l (t) représentant la profondeur courante: l(t)=p+1. FlowSyn essaie de resynthétiser le nœud t pour que l (t) n'augmente pas.

La minimisation de surface est assurée pendant la resynthèse en réinjectant les nœuds non-essentiels sans augmenter la profondeur du réseau.

En introduisant la resynthèse fondée sur les graphes des décisions binaires, FlowSyn gagne beaucoup sur FlowMap (15 % en terme de délai et de 25 % en terme de surface).

FlowMap-r [CONG93B]

Cette solution est une variante de FlowMap donnant aux utilisateurs la possibilité de choisir une solution compromise entre une solution orinentée chemin critique et une solution minimisant la surface.

CutMap [CONG95B]

L'approche "CutMap" améliore les résultats de FlowMap en terme de surface en maintenant la profondeur. CutMap combine la minimisation de la profondeur et de la surface en calculant la coupure k-acceptable de hauteur minimale et de coût minimum pour les nœuds critiques et la coupure k-acceptable de coût minimum pour les nœuds non-

critiques. Cela permet à CutMap de gagner 15 % en terme de surface par rapport à FlowMap.
Pourtant, FlowSyn est un peu meilleur que CutMap mais CutMap calcule en un temps CPU plus court que FlowSyn.

FlowMap modifié [CONG95A]

En constatant avec Mathur et Liu que le délai d'interconnexion contribue pour une part importante au délai total et que le délai d'interconnexion et proportionnel à la sortance des cellules logiques, FlowMap est modifié pour tenir compte du délai nominal. Il a été prouvé que le problème d'optimisation en utilisant le modèle de délai nominal n'est plus résolu en temps polynomial comme dans le cas de modèle de délai unité mais NP-difficile.
Le délai nominal est défini comme le délai total contenant également un délai du aux interconnexions proportionnel à la sortance des cellules logiques.
Soit un nœud v, le délai nominal à ce nœud v est calculé comme suit:
$D(v) = d_L$ + /Sortance de v/* d_N
Où:
* d_L est le délai d'un LUT; celui-ci est constant pour un type de cellule logique d'une technologie donnée.
* $d_N > 0$: délai ajouté dû à la sortance.

TechMap-L [SAW92]

Cette approche utilise un modèle de délai unitaire. Leur approche est constituée des étapes suivantes:
- Optimisation orientée surface en utilisant un partitionnement fondé sur la notion de "clique"
- Calcul des "slacks" (différence entre le temps requis et le temps d'arrivé) qui permettent de détecter la zone critique. Cette zone critique comprend les nœuds des "slacks" négatifs.
- Redécomposition de la zone critique en utilisant le partitionement fondé sur la notion de clique
- Mise à jour des "slack" pour détecter de nouveau la zone critique.
- Minimisation de la profondeur en utilisant la décomposition de Shannon dans la nouvelle zone critique.

Remarques:
TechMap-L a fait des progrès:
- L'algorithme de réduction de la profondeur n'est appliqué que dans la zone critique. Cette zone est l'ensemble des nœuds critiques qui forment une bande très étroite. La limitation de la zone sur laquelle appliquer l'algorithme permet de minimiser la profondeur en minimisant également la surface.

- Réduire la profondeur par un partitionnement fondé sur la notion de "clique" donnant un meilleur résultat que le bin packing (Chortle-d).

TechMap-D [SAW93]

TechMap-D hérite des principes de TechMap-L et utilise un modèle de délai plus sophistiqué et modifie un peu les traitements. La fonction de coût à chaque nœud est calculée comme suit:

Coût = α Profondeur + β Sortance - γ Entrées

L'approche est décomposée en 2 grandes étapes:

a) Minimisation de la profondeur:

- Optimisation orientée surface en utilisant le partitionnement fondé sur la notion de "clique" et les théorèmes de Shannon
- Calcul des "slack" pour détecter la zone critique.
- Les nœuds de la zone critique sont redécomposés pour diminuer la profondeur.
- Les nœuds de la zone non-critique sont redécomposés en utilisant l'algorithme d'optimisation de surface.
- Pour les nœuds qui ne peuvent pas être optimisés, appliquer l'expansion Shannon.

+ Placement orienté vitesse:

Le placement est fait dans le but de minimiser le délai d'interconnexion et détecter un ensemble de contraintes de placement et de routage pour les passer à l'outil de placement et de routage de FPGAs.

L'approche de Steve Trimberger et Mon-Ren Chen [TRI92]

L'approche est issue de Chortle-D et dédiée aux boîtiers de Xilinx. Elle a favorisé la génération alternée de FMAPs et de HMAPS en remarquant que le délai interne du CLB (entre FMAPs et HMAPs) est beaucoup plus faible que celui d'interconnexion. Dans ce sens, pour éviter le délai d'interconnexion, il est intéressant dans certains cas de dupliquer les logiques pour éliminer les nœuds de grande sortance.

L'approche de Anmol Mathur et C.L.Liu [MAT94]

En constatant que le délai d'interconnexion venant d'un noeud est proportionnel à la sortance de ce noeud, cet algorithme a utilisé le modèle de délai général de Penfield-Rubinstein [RUB83] pour estimer la performance temporelle. L'approche comprend la décomposition technologique, le placement et le routage dans un boucle.

Remarques:

Le délai général utilisé par cette approche permet d'introduire le délai d'interconnexion dans le délai total mais il se révèle trop compliqué à calculer. De plus, il dépend beaucoup des positions des cellules qui seront déterminées par l'outil de placement et de routage du fabriquant dans l'étape suivante. D'ailleurs, l'itération entre l'outil de synthèse et de

placement et de routage dans la boucle d'optimisation requiert une interactivité et exigera beaucoup de temps.

Edge-Map [YANG94]

En confirmant les remarques de Mathur et Liu à propos de l'importance du délai d'interconnexion dans le délai total, cette approche calcule le délai non pas à partir des délais à chaque noeud mais à partir des arêtes. Cela permet de donner aux arêtes des délais différents.

L'algorithme est une généralisation de FlowMap utilisant le modèle de délai général en combinant à nouveau la synthèse et le placement pour estimer avec une précision suffisante le délai d'interconnexion.

Il applique tout d'abord Edge-Map en affectant le délai unité à toutes les arêtes pour avoir la solution initiale. Puis le programme de placement est appliqué pour trouver le délai d'interconnexion. Ces informations de délai seront reaffectées aux arêtes de la solution initiale. La synthèse est appelée de nouveau avec les nouvelles informations de délai.

Edge-Map donne un bon résultat après le placement et routage. Le gain en termes de délai par rapport à FlowMap est environ 27 %. Cela confirme que la profondeur n'est pas le seul critère à estimer la performance temporelle de FPGAs à base de LUT mais qu'il faut prendre en compte également le délai d'interconnexion.

APPROCHE PROPOSEE

I. INTRODUCTION ET ALGORITHME GENERAL

A partir des approches présentées ci-dessus, nous pouvons tirer les remarques suivantes ayant mené à notre approche:

* Le modèle de délai joue un rôle très important dans l'estimation prédictive de performance temporelle. Les modèles de délai utilisés dans ces approches sont soit trop simples pour donner une bonne estimation (le modèle de délai unitaire), soit trop compliqués pour pouvoir les appliquer dans l'estimation en temps réel (le modèle de délai général [RUB85]). Nous allons utiliser le modèle de délai nominal qui tient compte du délai d'interconnexion dans le délai total. Ce modèle réaliste du délai permet d'estimer avec suffisamment de précision la performance temporelle sans trop augmenter le temps de calcul.

* Toutes ces approches visent à minimiser le délai des chemins les plus longs de la partie combinatoire. L'estimation de leur performance temporelle est fondée sur la fonction de coût en considérant que toutes les entrées ont la même priorité dans le calcul de la fonction de coût. On peut remarquer que l'optimisation temporelle dans les outils de synthèse élaborés optimise non seulement les chemins les plus longs mais aussi des régions ou des chemins spécifiques définis par l'utilisateur. Cela veut dire que l'on peut définir une région dite sensible qui doit faire l'objet de l'optimisation. La fonction de coût affectée à chaque nœud devrait refléter cette relation. Nous allons présenter une nouvelle méthode du calcul de la fonction de coût en tenant compte de cette relation pour satisfaire la demande d'optimisation de l'utilisateur.

* De plus, l'estimation traditionnelle de performance temporelle, basée sur le calcul du délai présentée dans les approches récentes, implique que la zone critique (à optimiser) est celle qui amène un délai maximal à partir de toutes ses entrées. Cette définition de la zone critique ne convient plus dans le cas d'optimisation demandée par l'utilisateur. De plus, la zone critique est définie traditionnellement comme étant l'ensemble des chemins critiques à base du calcul de "slack". Cette définition ne permet pas d'améliorer suffisamment la performance. Nous allons proposer une nouvelle définition de la zone critique, des chemins critiques ainsi que quelques nouvelles notions pour mieux l'adapter à la demande d'optimisation.

* L'optimisation temporelle implique souvent une augmentation de surface qui complique le problème de placement et de routage, ce qui dégrade effectivement la performance de la conception. Notre approche sera appliquée seulement dans la zone critique (selon la nouvelle définition) pour améliorer la performance. La zone non-critique sera optimisée

utilisant un critère orienté surface pour non seulement maintenir une surface raisonnable mais encore ne pas dégrader la performance temporelle du réseau booléen.
Pendant la resynthèse, différents types de réinjections seront appliqués dans la zone critique pour encore améliorer le résultat.

Notre approche [LE95A], [LE96A], [LE96B] est présentée de manière générale dans la figure 3.1:

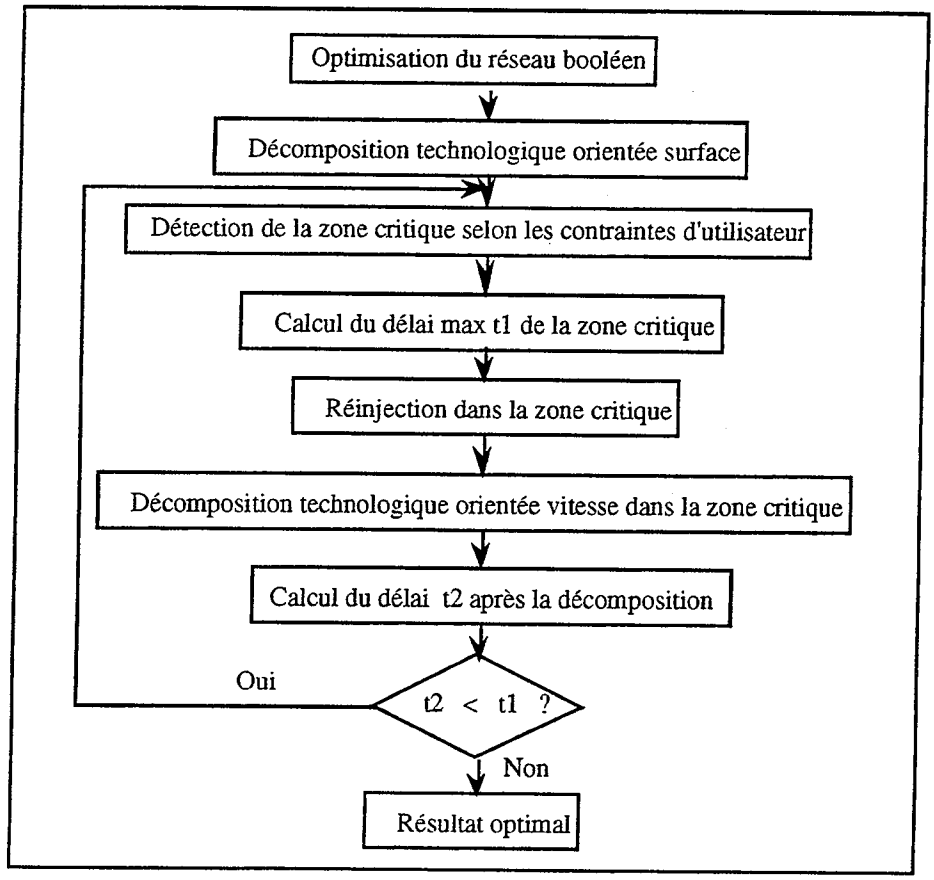


Figure 3.1 Algorithme général de l'optimisation de performance temporelle

L'étape de décomposition technologique orientée surface a pour but d'obtenir un premier réseau booléen de cellules LUT-m pour pouvoir détecter le plus précisément possible la zone critique en minimisant aussi la surface. Il a été constaté qu'il s'agit d'un bon point de départ pour l'optimisation temporelle [MUR91A].

Cette étape peut être réalisée à base de deux représentations des fonctions booléennes: forme factorisée algébrique ou les graphes de décision binaire [BES94], [LE95].

Le réseau booléen donné par l'étape d'optimisation de surface est déjà m-faisable. La zone critique sera détectée selon les cas d'optimisation. Selon la complexité de la fonction booléenne, nous allons déterminer la zone critique ou ε-critique (cela sera défini plus tard). La zone critique est donc aussi m-faisable. Pourtant, travailler directement sur cette structure n'améliore pas grand-chose car comme le réseau est déjà décomposé en cellules

LUT-m, on ne peut appliquer que la simplification de la fonction booléenne qui n'est pas très efficace.

Donc, pour aider l'étape de la décomposition technologique orientée vitesse, les cellules de la zone critique seront réinjectées en combinant les types de réinjection présentés dans la section III du chapitre 2. La réinjection transforme le réseau en un réseau k-faisable, k>m, qui sera redécomposée de nouveau en minimisant le délai des chemins critiques. Ceci donne un meilleur résultat que dans le cas où l'on travaille directement sur un réseau m-faisable.

La décomposition technologique orientée vitesse sera appliquée finalement pour diminuer le délai de la zone critique ou ε-critique. Ces étapes seront répétées jusqu'à l'obtention du meilleur résultat possible.

II RESEAUX BOOLEENS

II.1 Définitions

Un réseau booléen est un graphe biparti comprenant deux sortes de nœuds (N,E):

- Les nœuds N sont associés à des éléments logiques. Ces éléments logiques sont de 3 types: des portes logiques notées G, des points de mémorisation notés M et des plots d'Entrée/Sortie notés Z. Un tel nœud a une sortance de 1 car nous nous restreignons à des portes à une seule sortie. Son entrance correspond aux différentes entrées d'une porte.
- Les nœuds équipotentiels E. Un tel nœud a une entrance de 1 et une sortance qui correspond aux entrées des portes atteintes par cette équipotentielle.

Un arc du graphe relie un nœud N (resp. E) à un nœud E (resp. N).

Un exemple de réseau est montré dans la figure 3.2

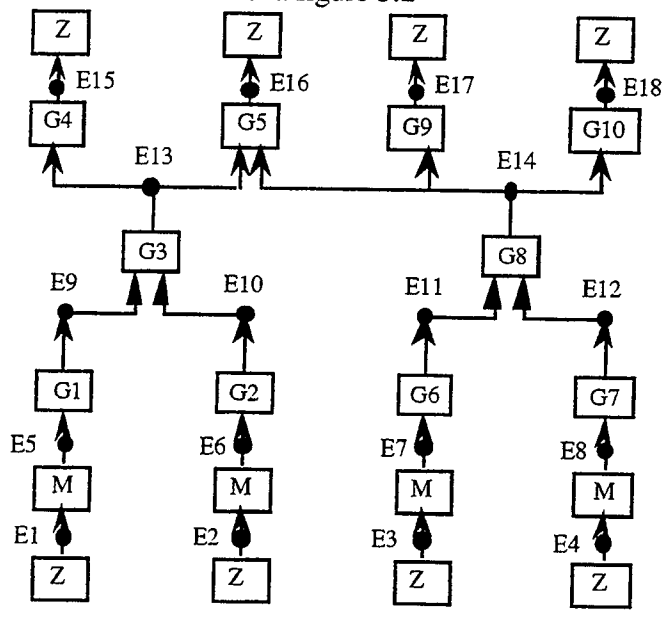


Figure 3.2 Réseau booléen

• Les entrées/sorties du réseau booléen sont appelées entrées/sorties primaires.
• Les entrées/sorties des points de mémoire sont appelées sorties/entrées secondaires.
• Un successeur ou un prédécesseur immédiat d'un nœud E (resp. N) est un nœud N (resp.E).
• Une porte F est dite successeur d'une porte G s'il existe une équipotentielle E telle que F est un successeur immédiat de E et G est un prédécesseur immédiat de E.
• Une porte F est dite prédécesseur d'une porte G s'il existe une équipotentielle E telle que F est un prédécesseur immédiat de E et G est un successeur immédiat de E.
• Comme dans un graphe quelconque, le cône prédécesseur d'un nœud est composé de l'ensemble des prédécesseurs de ce nœud.

Réseau booléen k-faisable

Les réseaux booléens considérés ici sont des réseaux dont les portes logiques sont implémentables sur des cellules du type LUT. On appelle ces nœuds des cellules.
Un réseau booléen k-faisable est un réseau booléen dont les cellules sont les LUT-k.

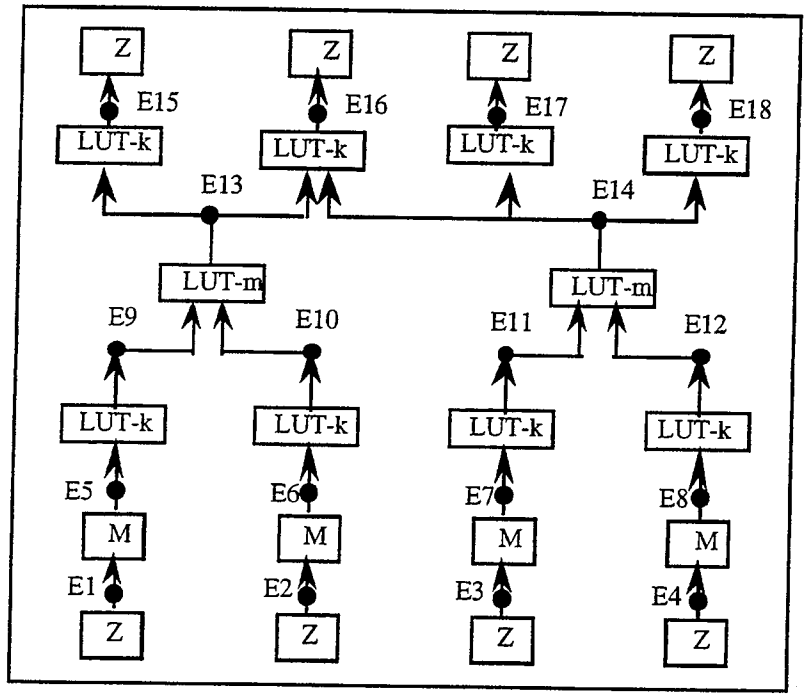


Figure 3.3 Réseau booléen k-faisable

Dans un réseau booléen, une cellule LUT-k est utilisée pour implémenter une fonction booléenne d'au plus k variables. On pourra lui associer bijectivement l'arborescence factorisée correspondant à la fonction booléenne précise implémentée (Figure 3.4)

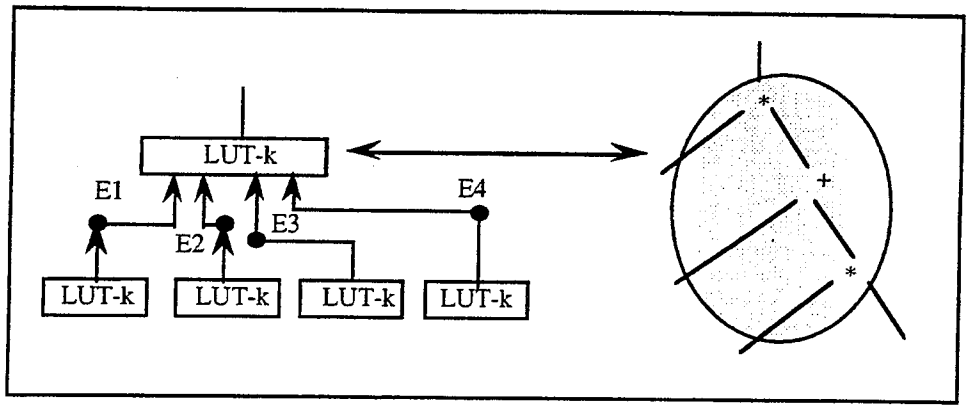


Figure 3.4 Arborescence factorisée associée à un LUT-k

II.2 Zones

On appelle zone l'ensemble de tous les chemins reliant un sous-ensemble de nœuds source et un sous-ensemble de nœuds destination et ne traversant que des LUT-k ou équipotentielles.

Zone sensible

L'optimisation temporelle d'un réseau booléen consiste à optimiser les délais de propagation dans une zone donnée. Cette zone est alors appelée zone sensible.
Les nœuds dans la zone sensible sont appelés nœuds sensibles.
Les entrées (sorties) de la zone sensible sont appelées entrées (sorties) sensibles.

Détection de la zone sensible

Une zone sensible peut être arbitrairement définie par un utilisateur.
De façon classique, pour optimiser un réseau booléen, on considère les zones suivantes:

* La zone (entrée/sortie) est l'ensemble de tous les chemins reliant les entrées et les sorties primaires et ne traversant que des équipotentielles et des LUT.
* La zone (entrée/bascule) est l'ensemble des chemins reliant les entrées primaires aux bascules et ne traversant que des équipotentielles et des LUT .
* La zone (bascule/bascule) est l'ensemble des chemins reliant les entrées et sorties de bascules (secondaires) et ne traversant que des équipotentielles et des LUT .
* La zone (bascule/sortie) est l'ensemble des chemins reliant les sorties de bascules (entrées secondaires) et les sorties primaires et ne traversant que des équipotentielles et des LUT.

Partie combinatoire du réseau booléen

On appelle partie combinatoire d'un réseau booléen l'union des 4 zones définiées ci-dessous (Figure 3.5)

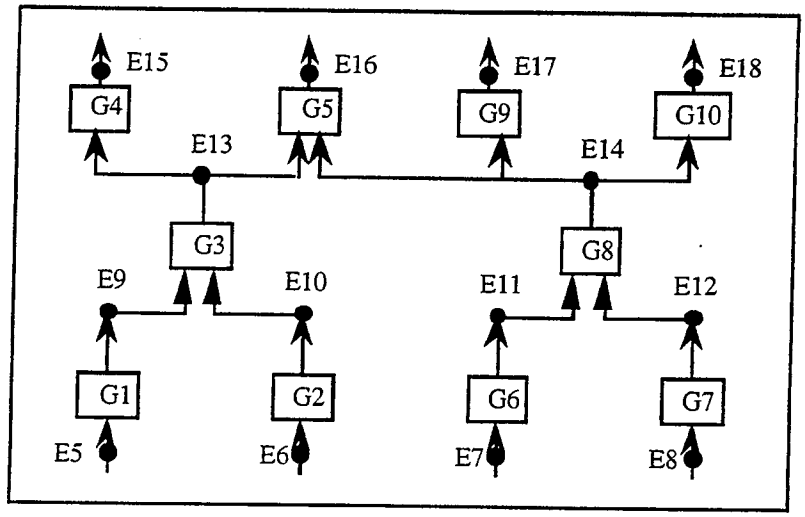


Figure 3.5 Partie combinatoire du réseau booléen dans la figure 3.4

Simplification de la partie combinatoire du réseau booléen

Pour la manipulation algébrique ultérieure, il n'est pas nécessaire de nommer les équipotentielles. Une équipotentielle sera remplacée dans la suite par un ensemble d'arcs traduisant la relation de dépendance fonctionnelle.

Exemple:

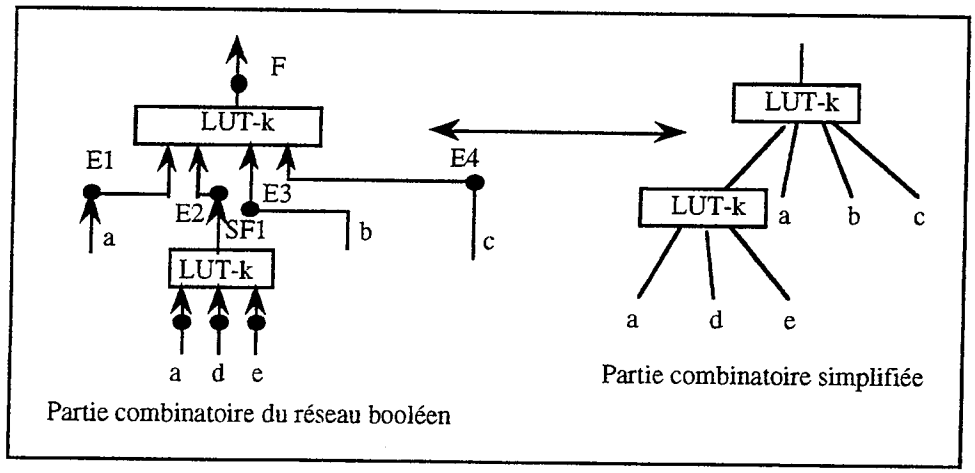


Figure 3.6 Simplification de la partie combinatoire du réseau booléen

Dorénavant, nous parlerons des zones simplifiées, ou plus brièvement, des réseaux de LUT-k.

III DETECTION DE LA ZONE CRITIQUE

Comme le modèle de délai joue un rôle très important dans l'optimisation temporelle, nous allons présenter tout d'abord les modèles de délai les plus couramment cités dans la littérature. Puis, nous proposerons un modèle adapté aux technologies des FPGA à base de LUT. Le calcul du délai dans le circuit fondé sur ce modèle permettra de détecter correctement la zone que l'utilisateur veut optimiser.

III.1 Modèle de délai

Le modèle de délai joue un rôle très important dans l'optimisation de performance temporelle des FPGAs à base de LUT. Un bon modèle de délai permet d'estimer au mieux le délai réel y compris l'impact du délai d'interconnexion dans le délai final. La zone critique estimée doit être proche de la zone critique réelle et ceci permettra une optimisation efficace sans trop augmenter la surface.

Pour l'optimisation de performance des FPGAs à base de LUT, nous passerons en revue les modèles les plus classiques:

III.1.1 Modèle de délai unitaire

Ce modèle de délai est beaucoup utilisé dans les premiers articles parus dans le domaine de l'optimisation de performance temporelle pour FPGAs à base de LUT [CONG92A], [FRA91B], ... Ce modèle ne traite que le délai à travers de blocs logiques en considérant que le délai de ces blocs est constant et sans tenir compte du délai d'interconnexion.

En réalité, ce modèle de délai n'est pas du tout suffisant pour décrire le délai dans les boîtiers FPGAs, surtout pour les technologies à base de SRAM dont le délai d'interconnexion contribue pour une part très importante au délai final.

Pourtant, dans le cas des FPGAs dont l'interconnexion est assurée par un bus, et non par des interrupteurs à base de RAM, le délai d'interconnexion est quasiment constant. Il peut être inclus dans le délai des cellules logiques comme dans le cas des FPGAs de Altera/Flex8000. Cependant, quand la sortance est assez grande, le modèle doit être modifié.

III.1.2 Modèle de délai nominal

Le modèle de délai nominal est défini comme suit:

$D(u) = \text{Max}(D(\text{pred}(u))) + \Delta t_{cellule} + \alpha * FO$

où:
- $\Delta t_{cellule}$ est le délai intrinsèque de la cellule
- α est le facteur de délai d'interconnexion
- FO est la sortance du nœud u

Ce modèle de délai permet de prendre en compte le délai d'interconnexion dans le délai total. Son application est relativement simple et peut donner de bons résultats.

Ce modèle est simplifié car il attribue le même délai à toute les entrées de portes attaquées par une équipotentielle.

III.1.3 Modèle de délai général [RUB83]

Ce modèle permet de calculer exactement le délai total y compris le délai d'interconnexion mais son calcul est trop compliqué. Cependant, le calcul du délai général exige des informations concernant la position des cellules. Ces informations ne peuvent être obtenues qu'après placement et routage. Ceci demande d'inclure les étapes de placement et de routage dans la boucle de synthèse. Par conséquence, la synthèse ne peut plus être faite automatiquement mais avec l'intervention de l'homme. De plus, après avoir extrait les informations de placement et de routage, on resynthétise le réseau. Par conséquent, la position des cellules est modifiée et l'estimation n'est plus valable.

Si le placement est prédit par la synthèse, ce placement sera invalidé par le placement propre au fabriquant.

Par ailleurs, l'algorithme du calcul de délai général est trop compliqué pour pouvoir être appelé plusieurs fois pendant la décomposition technologique.

III.1.4 Modèle de délai proposé pour les FPGAs à base de LUT

Le modèle général étant trop compliqué, nous préférons un autre modèle séparant les deux étapes de synthèse, logique et physique, et permettant donc d'avoir des résultats au cours de la synthèse.

En constatant que le délai d'interconnexion est réellement proportionnel à la sortance des nœuds, [CONG95A], nous allons utiliser un modèle proche du délai nominal.

III.2 Calcul du délai du chemin d'une entrée à un nœud dans le réseau

Le délai d'un chemin d'une entrée i à un nœud j dans le réseau est calculé d'après la formule:

$$D(i, j) = D(i, j-1) + \Delta t_j + \delta * FO_j$$

* j-1 est la cellule prédécesseur de j dans le chemin p (i, j)
* D (i, j-1) est le délai du chemin reliant l'entrée i au nœud j-1
* Δt_j est le délai intrinsèque de la cellule logique j.
* δ est le délai dû à une sortance
* FO_j est la sortance du nœud j.

La figure 3.7.a illustre le calcul du délai d'un chemin:

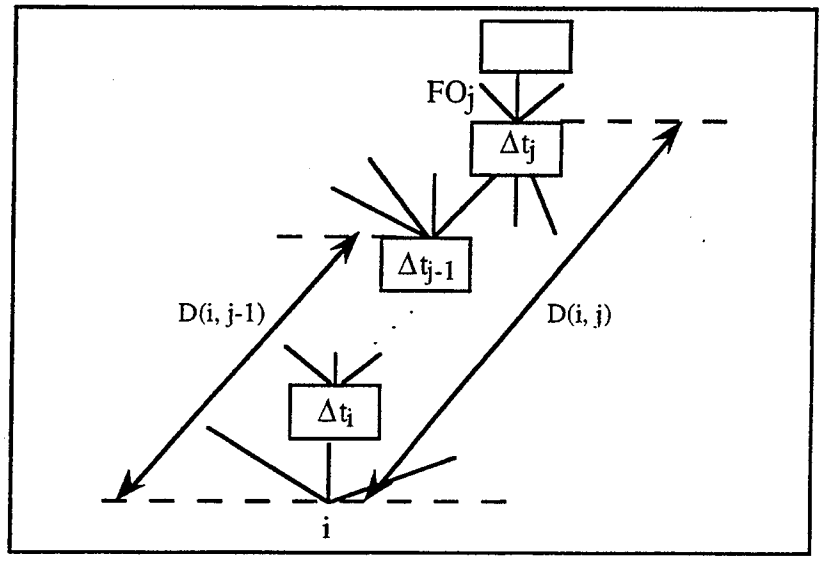


Figure 3.7.a Calcul du délai d'un chemin d'une entrée i à un nœud j

La formule peut être réécrite:

$$D(i,j) = \sum_{k=i}^{j} \Delta t_k + \delta * \sum_{k=i}^{j} FO_k$$

Δt_k est le délai intrinsèque de la cellule logique k dans le chemin p (i, j)

FO_k est la sortance du nœud k dans le chemin de l'entrée i au nœud j

Quand toutes les cellules logiques sont du même type, c'est-à-dire, $\Delta t_k = \Delta t_{cellule} = \Delta t$, nous avons:

$$D(i, j) = \Delta t * d(i, j) + \delta * \sum FO_k$$

où:

Δt est le délai intrinsèque d'une cellule logique

d (i. j) est le nombre de LUT-m sur le chemin de l'entrée i au nœud j

δ est le délai d'interconnexion dû à une sortance.

Ce modèle de délai est bien adapté aux technologies à base de LUT utilisant les SRAM et des architectures d'interconnexion du type Xilinx, AT&T. Le facteur $\delta * \sum FO_k$ reflète avec une certaine précision le délai d'interconnexion dans ce type de FPGAs.

Pour les technologies à base de LUT utilisant des bus d'interconnexion, le délai d'interconnexion est quasi-constant. Ce délai peut donc être inclus dans le délai de la cellule logique. Pourtant, quand la sortance est trop grande, le délai d'interconnexion varie et il est souhaitable de modifier la formule pour pouvoir tenir compte de cet effet.

La figure suivante illustre le calcul du délai:

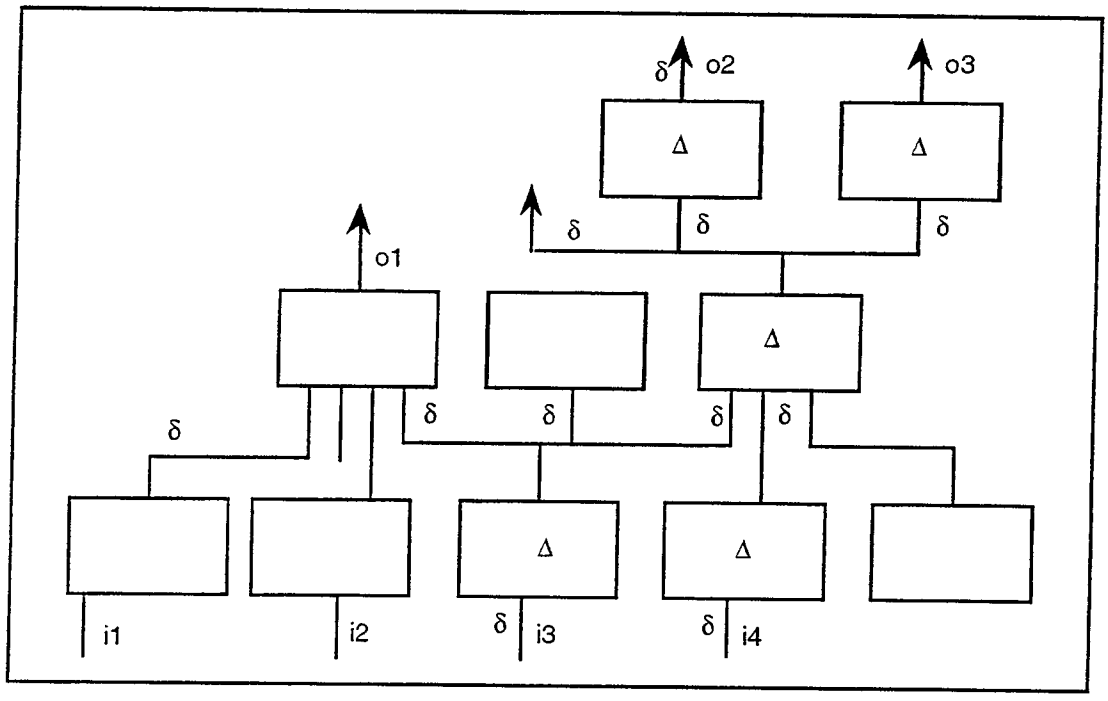


Figure 3.7.b Illustration du calcul du délai du chemin d'une entrée à un nœud

D (i3, o2) = 3 Δ + 8 δ

D (i4, o2) = 3 Δ + 6 δ

III.3 Détection de la zone critique

III.3.1 Définitions

Chemin critique du réseau booléen

Un chemin critique est un chemin ayant le délai le plus long parmi les chemins de la zone sensible. Ce délai est appelé D_{max}.

Les entrées (sorties) des chemins critiques sont appelées entrées (sorties) critiques.

Zone Critique

Une zone critique est définie comme les cônes prédécesseurs des sorties critiques.

Sortie ε-critique

Une sortie ε-critique est une sortie sensible dont le délai dans la zone sensible est supérieur à D_{Max} -ε.

S est une sortie ε-critique $\Leftrightarrow D(i_k, S) \geq D_{Max} - \varepsilon \quad \forall i_k \in$ entrées sensibles, S est une sortie sensible

Zone ε-critique

Une zone ε-critique est définie comme l'union des cônes prédécesseurs des sorties ε-critiques.

La figure 3.8 illustre la notion de la zone ε-critique.

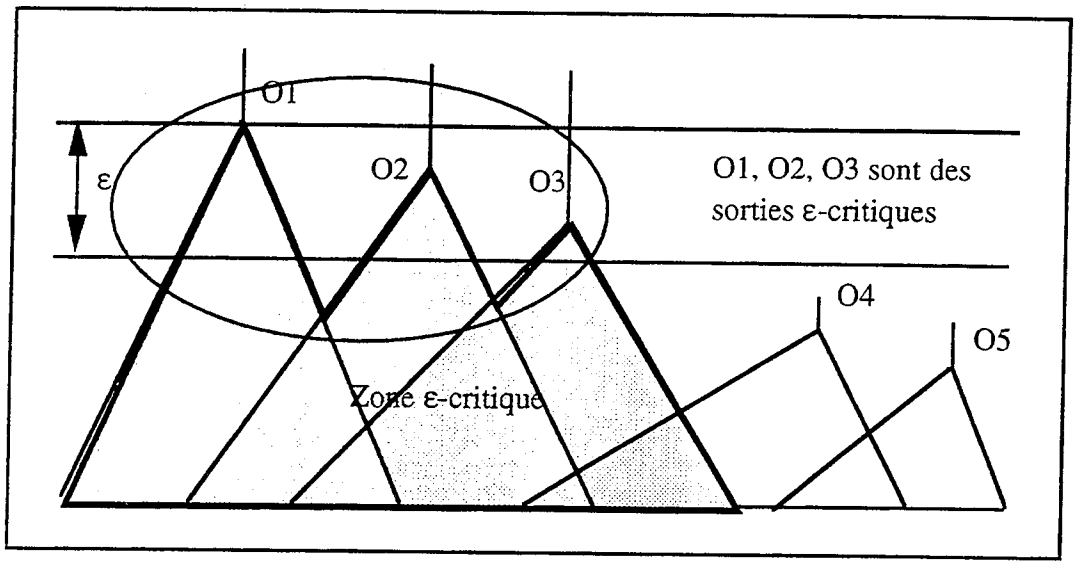


Figure 3.8 La zone ε-critique

• Les nœuds critiques sont les nœuds appartenant aux chemins critiques.

• Les nœuds quasi-critiques sont les nœuds dans la zone critique qui n'appartiennent pas aux chemins critiques mais dont le cône prédécesseur comporte au moins une entrée critique.

• Les nœuds non-critiques sont les nœuds qui n'appartiennent pas aux cas cités ci-dessus.

III.3.2 Méthode de détection de la zone critique/ε-critique

[SAW92], [SAW93] détectent les chemins critiques et ε-critiques. L'algorithme d'optimisation n'est appliqué que sur ces chemins qui forment une zone assez étroite. Cette technique d'optimisation ne permet pas une amélioration suffisante car la zone à optimiser est étroite et on ne peut pas profiter d'autres blocs logiques pour minimiser le délai des chemins critiques.

La zone critique qu'on a définie ci-dessus donne plus de liberté et d'efficacité pour la décomposition technologique orientée vitesse ultérieure.

De plus, en limitant la complexité de la zone de resynthèse, l'algorithme d'optimisation, appliqué soit dans la zone critique, soit dans la zone ε-critique, permet d'améliorer le résultat en gardant un temps de calcul raisonnable.

Pour détecter la zone critique, nous devons tout d'abord trouver les sorties critiques. Puis, nous recherchons des cônes prédécesseurs des sorties critiques. L'algorithme est le suivant:

Détection de la zone Critique/ε-critique

Extraction de la zone sensible
Extraction des entrées sensibles
Extraction des sorties sensibles
Calcul des délais des chemins entre des entrées et sorties sensibles
Extraction des chemins critiques ayant le délai maximal
Si (Réseau booléen est complexe)
  Extraction des sorties critiques
  La zone critique est composée des cônes prédécesseurs des sorties critiques
Sinon
  Extraction des sorties ε-critiques
  La zone ε-critique est composée des cônes prédécesseurs des sorties ε-critiques

Remarquons que nous ne traitons pas les faux chemins dans la détection de la zone critique.

IV DECOMPOSITION ORIENTEE VITESSE

Introduction

Après la réinjection, la zone critique n'est plus m-faisable. L'algorithme de la décomposition orientée vitesse va redécomposer le réseau en progressant des entrées vers les sorties et en garantissant que tous les nœuds prédécesseurs du nœud en traitement sont déjà m-faisable. Cette façon de décomposer permet d'estimer exactement le délai au nœud en cours de traitement car ce nœud ne dépend que des nœuds prédécesseurs qui sont m-faisables.

Remarquons que dans l'optimisation de performance temporelle, toutes les entrées n'ont pas la même priorité dans le calcul de délai. On peut remarquer que le calcul des chemins critiques ou l'estimation de la performance n'est basé que sur les chemins reliant des entrées critiques ou sensibles et les sorties sensibles ignorant les chemins partant des entrées non-critiques. Par exemple, le délai à un nœud est calculé en considérant les chemins partant des entrées sensibles et non de toutes les entrées.

Les chemins critiques, composés des nœuds critiques influent directement sur les performances temporelles; les nœuds sensibles sont susceptibles de devenir critiques tandis que les entrées non-critiques n'influent pas sur la performance. Cela implique que la fonction de coût affectée à chaque nœud doit être différente et refléter cette relation.

IV.1 Calcul du coût global associé à un nœud du réseau

A chaque nœud v du réseau est associé un coût-global, noté Coût_Global_Réseau (v), qui est calculé comme suit:

* Si v est une entrée critique

  Coût_Global_Réseau (v) = 0

* Si v est une entrée sensible

  Coût_Global_Réseau (v) = - $\alpha * \Delta t_{cellule}$ où α est un facteur de correction pour exprimer que cette entrée est moins importante que les entrées critiques.

* Si v est une entrée ni critique, ni sensible

  Coût_Global_Réseau (v) = -1

* Si v est un nœud critique:

  Le Coût_Global_Réseau de v est le délai maximal de tous les chemins partant des entrées sensibles et finissant par ce nœud v.

  Coût_Global_Réseau (v) = Max (D (i, v))

  où i ∈ l'ensemble des entrées sensibles

* Si v est un nœud sensible/quasi-critique:

  Le Coût_Global_Réseau de v est le délai maximal de tous les chemins partant des entrées sensibles et finissant par ce nœud v moins un facteur de correction $\alpha * \Delta t_{cellule}$.

  Coût_Global_Réseau (v) = Max (D (i, v)) - $\alpha * \Delta t_{cellule}$

  où i ∈ l'ensemble des entrées sensibles

  α est un facteur dépendant de la technologie.

* Si v n'est pas dans les cas ci-dessus:

  Le coût global est -1.

  Coût_Global_Réseau (v) = -1

Les exemples suivants illustrent la façon de calcul des coûts globaux dans le réseau. Pour simplifier ces exemples:

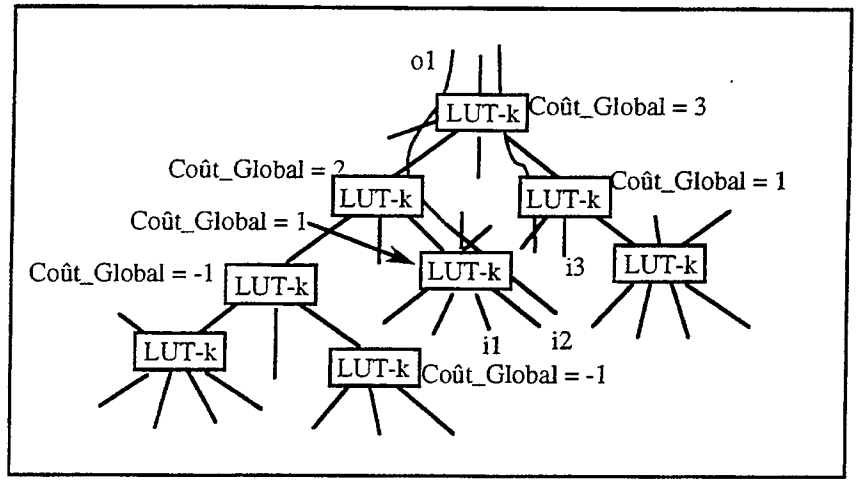
* Nous néglirerons le délai d'interconnexion, $\delta = 0$.

* Nous prendrons un délai unitaire intrinsèque, $\Delta = 1$.

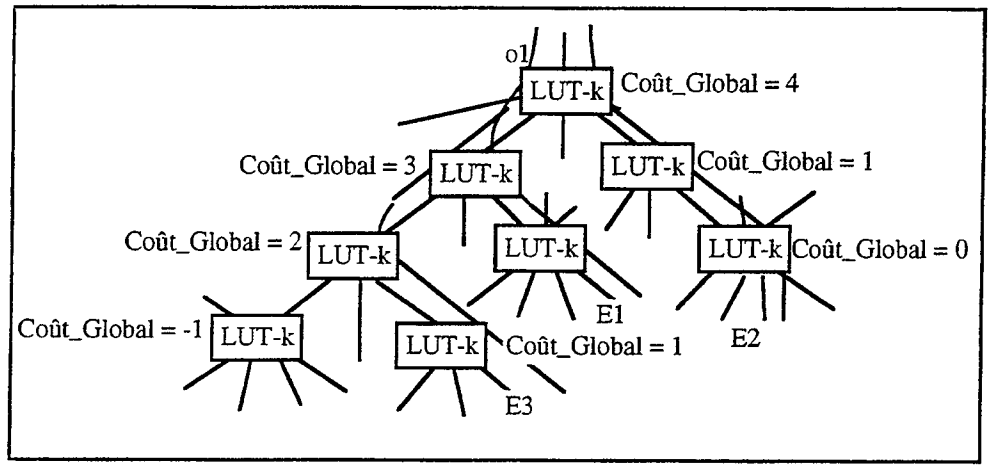
Dans la figure 3.9, le même exemple est illustré pour différents cas d'optimisation.

- Cas 1: Les entrées critiques sont choisies par l'utilisateur: i1, i2, i3 (figure 3.9a). Les chemins critiques sont (i1, o1), (i2, o1), (i3, o1).
- Cas 2: Supposons qu'une zone sensible ait été définie comme l'ensemble des chemins reliant les nœuds source (E1, E2, E3) et le nœud destination o1. Dans ce cas, le chemin critique déctecté est le chemin (E3, o1) (figure 3.9b).

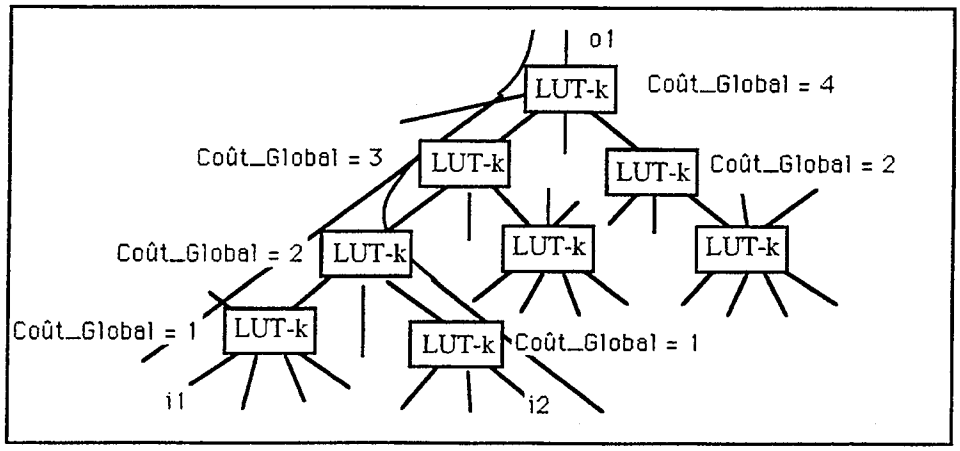
• Cas 3: On suppose ici que la zone sensible référencie toute la partie combinatoire, le chemin critique revient au chemin critique classique dont le délai est maximal (Figure 3.9.c).



a. Optimisation des zones indiquées par l'utilisateur



b. Optimisation des zones classiques



c) Optimisation d'un réseau pur combinatoire

Figure 3.9 Calcul du coût global dans la partie combinatoire

IV.2 Calcul de la fonction de coût dans une cellule

Considérons l'arborescence factorisée de la sous-fonction correspondant à une cellule.
La fonction de coût affectée à chaque nœud u dans l'arborescence associée à une cellule, notée Coût (u), est composée de 3 paramètres:
Coût (u) = (Coût_Global_Cellule (u), Coût_Local (u), Nombre_de_Var (u))
Chaque paramètre est défini comme suit:
• Coût_Global_Cellule (u):
Le Coût_Global_Cellule (u) est calculé à partir des Coût_Global_Cellule des entrées de son cône prédécesseur.
Le Coût_Global_Cellule d'une entrée correspond au Coût_Global_Réseau de la cellule prédécesseur défini dans la section IV.1 du chapitre 3.
Le Coût_Global_Cellule d'un nœud est le maximum des Coût_Global_Cellule des entrées de son cône prédécesseur.
Exemple:

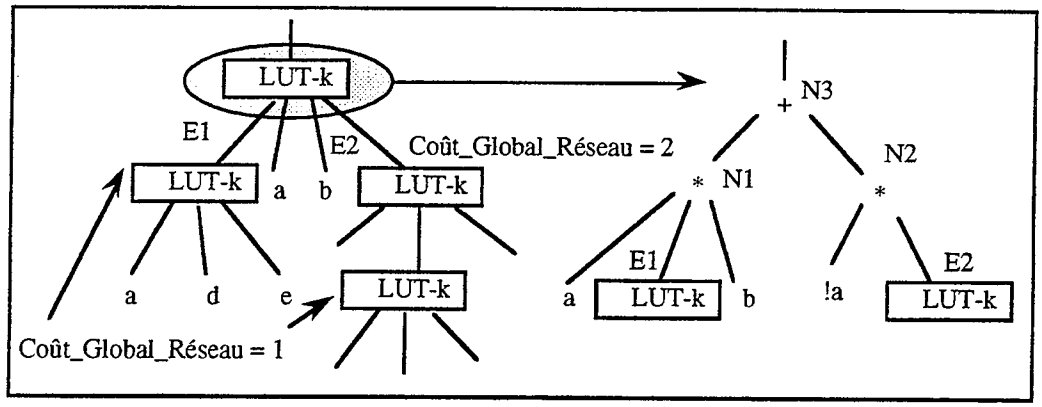


Figure 3.10a Calcul du Coût_Global_Cellule dans une cellule

Dans la figure 3.10a, le Coût_Global_Cellule de l'entrée E1 est 1, de E2 est 2. Le Coût_Global_Cellule de N1 est 1 et de N3 est 2.

• Coût_Local (u) est la profondeur de u dans l'arborescence factorisée associée à la cellule.
• Nombre_de_Var (u) est le nombre de variables d'entrée du cône prédécesseur de u dans l'arborescence factorisée associée à la cellule.

Le calcul de la fonction de coût est illustré en figure 3.10b
A des fins de simplification, les hypothèses suivantes seront faites:

* $\Delta = 1,\ \delta = 0$
* Coût (N1) = (3, 0, 1) (calculée précédemment)
* Les nœuds N2, N3 ne sont ni sensibles ni critiques => Coût_Global_Cellule = -1

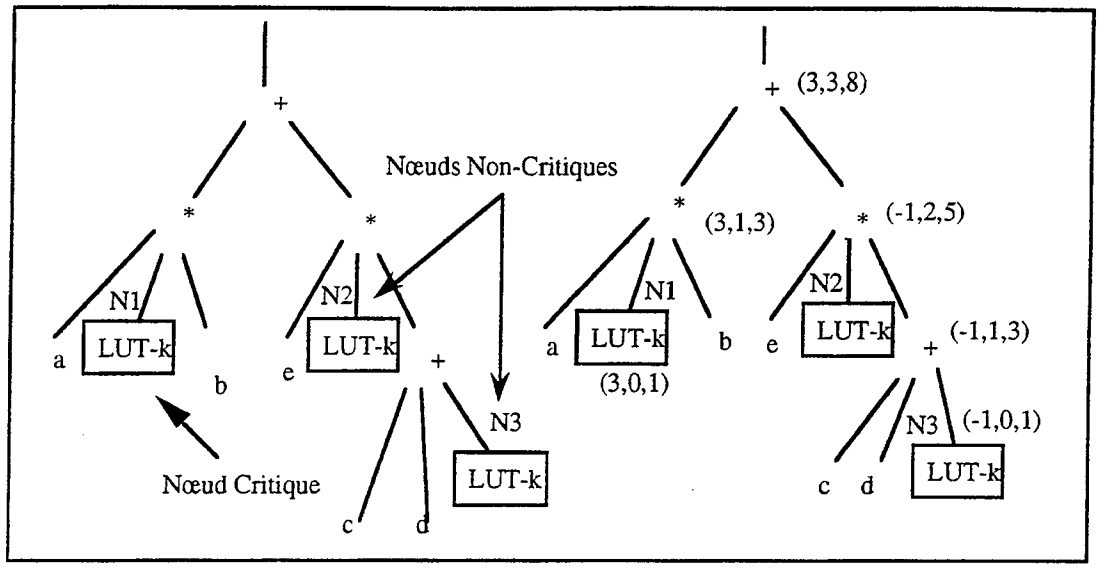


Figure 3.10b Calcul de la fonction de coût dans une cellule

IV.3 Relation d'ordre entre les fonctions de coût des nœuds dans une cellule

Nous définissons une relation entre les fonctions de coût des nœuds dans une cellule comme suit:

Le Coût_Global_Cellule est examiné pour déterminer une relation d'ordre entre deux fonctions de coût. Si les Coûts_Globaux_Cellule de deux fonctions de coût sont égaux, les Coûts_Locaux sont considérés. S'ils sont égaux, Nombre_de_Var (u) est pris en compte.

IV.4 Méthode de la décomposition technologique orientée vitesse

a) Prétraitement de la zone critique avant la décomposition technologique orientée vitesse

Soit k_0 le nombre d'entrée du LUT dans la technologie considérée.

La zone critique détectée dans l'étape précédente est composée en cellules LUT-k_0. Comme expliqué dans la section III.1, il est souhaitable de réinjecter certaines cellules dans la zone critique d'une manière contrôlable pour aider l'étape de décomposition technologique orientée vitesse.

* Toutes les cellules ayant une sortance unitaire sont candidates pour la réinjection spécifique à la famille FPGA à base de LUT (Voir le paragraphe III.2.4.b dans le chapitre 2). Le paramètre k sera choisi d'après la technologie considérée afin d'obtenir des cellules LUT-k d'une taille raisonnable pour la décomposition technologique orientée vitesse. k ne peut pas être très grand pour éviter le temps de calcul trop long dans l'étape de décomposition. Il ne peut pas non plus être trop petit car ceci ne permet pas une

amélioration suffisante. Par exemple, pour les familles XC4000 et Flex8000, k est choisi à 8 car le nombre de variables d'entrée de CLB de la famille XC4000 est de 3 (HMAP) ou 4 (FMAP) et le nombre de variables d'entrée de la cellule logique (LE) de Flex8000 est de 4. Cette valeur donne le meilleur résultat expérimentalement. Pour la technologie ORCA permettant des configurations de base de LUT-5 ou LUT-6, k sera plus grand.
Par ailleurs, quand la zone critique n'est pas très complexe [ABO92] (le nombre de littéraux est inférieur à 250), k est choisi à 11 pour obtenir une cellule plus grande permettant plus de liberté à la décomposition ultérieure.
Exemple:

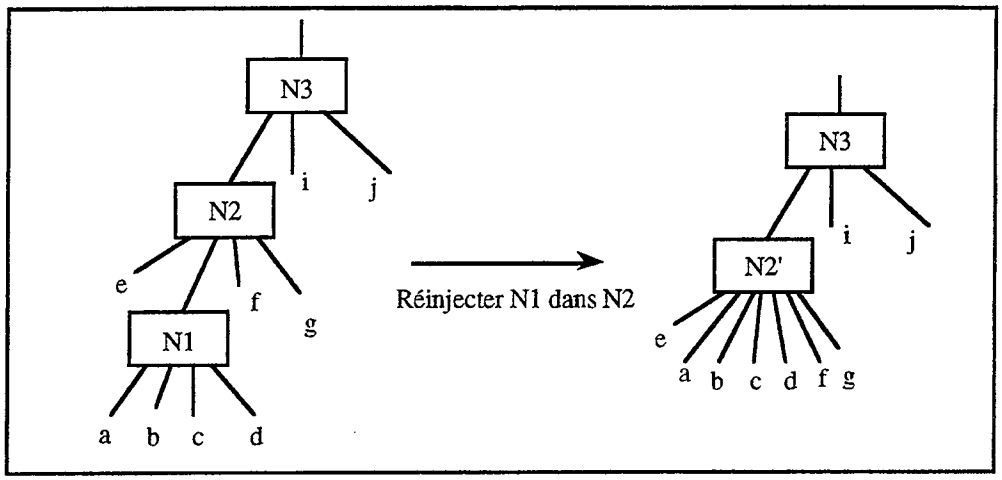


Figure 3.11 Combinaison de la réinjection des cellules non-partagée et de la réinjection spécifique aux familles FPGA à base de LUT

La figure 3.11 illustre la combinaison des types de réinjection des sous-fonctions non-partagées pour les FPGAs à base de LUT. Supposons que k soit égale à 8 pour cet exemple.
Dans la figure 3.11, la cellule N1 sera réinjectée dans N2 pour donner N2' car elle vérifie les conditions:

- N1 n'est pas partagée
- Le nombre de variables de la sous-fonction résultat SF2' est de $7 < 8 = k$

Ensuite, N2' sera examinée. N2' n'est pas partagée mais elle ne vérifie pas la condition:

- Le nombre de variables de la sous-fonction résultat est de $10 > 8 = k$

Donc, elle ne peut plus être réinjectée dans N3

* Si la cellule est partagée, nous allons combiner la réinjection en tenant compte du gain en terme de littéraux et la réinjection spécifique aux familles FPGA à base de LUT.
Si le gain de la cellule n'est pas important (inférieur à G dépendant de la technologie) et que cette cellule vérifie les conditions de réinjection spécifique aux famille à base de LUT, elle sera réinjectée uniquement dans la zone critique pour ne pas augmenter la surface.

Exemple:

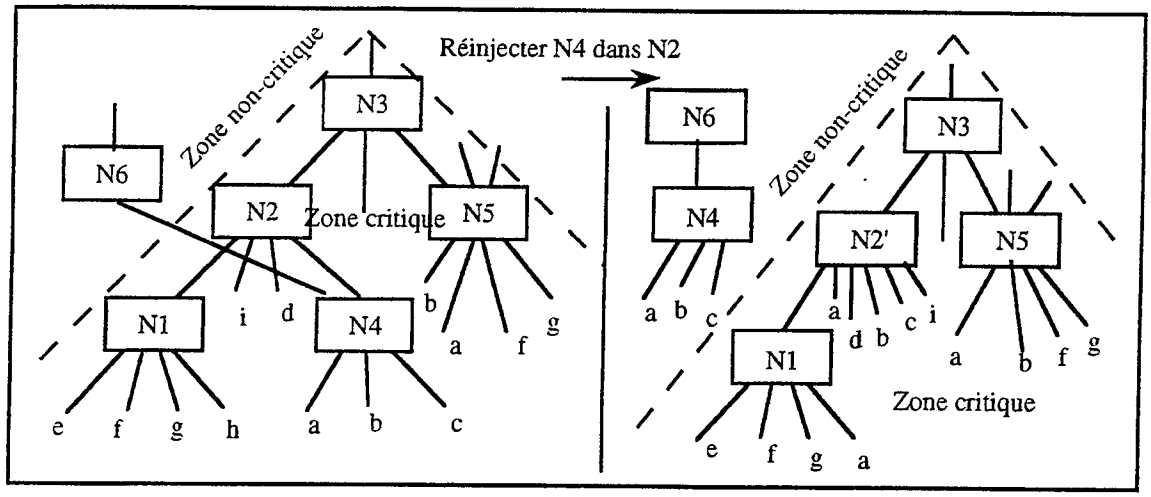


Figure 3.12 Réinjection des cellules dans la zone critique

Dans la figure 3.12, supposons que G soit égale à 3 et que k soit égal à 8. Supposons que le gain de N4 soit de 2. N4 vérifie les conditions pour la réinjection dans N2 mais pas dans N6 car N6 n'est pas dans la zone critique. N4 est donc réinjectée dans N2 pour obtenir la cellule résultat N2'.

Après la réinjection contrôlée, la zone critique devient un réseau de cellules LUT-k, $k>k_0$.

b) Décomposition technologique global de la zone critique

La décomposition technologique est faite en progressant des entrées vers les sorties dans la zone critique du réseau booléen.

Toutes les cellules de la zone critique seront tout d'abord ordonnées par ordre croissant de leur profondeur. Les cellules LUT-k sont traitées dans cet ordre en partant de la moins profonde.

Si la cellule est sensible ou critique, nous la traitons par une décomposition orientée vitesse. Si non, elle sera traitée par une décomposition orientée surface [BAB92].

Dans la figure 3.13, supposons que E1 soit une entrée sensible, et que E2 soit une entrée critique. Les cellules critiques ou sensibles N1, N2, N3, N5, N6, N9 seront traitées par une décomposition orientée vitesse tandis que les cellules N4, N7, N8 sont traitées par une décomposition orientée surface.

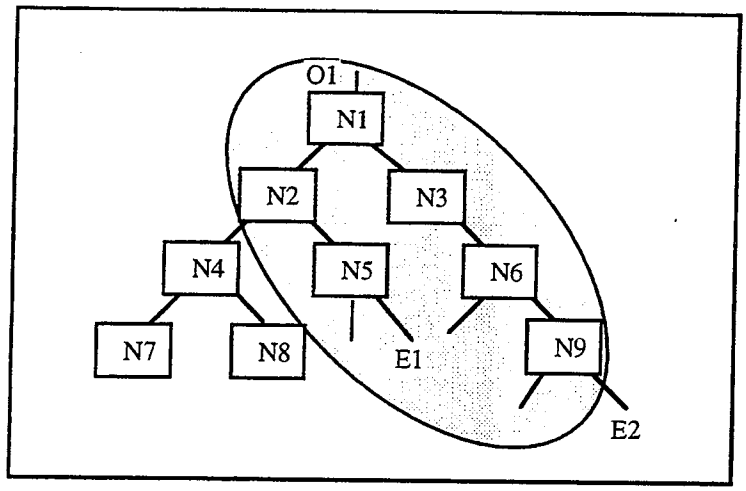


Figure 3.13 La décomposition technologique orientée vitesse dans la zone pointillée

c) Décomposition technologique orientée vitesse pour chaque cellule LUT-k

Dans l'arborescence factorisée de chaque cellule, la décomposition s'effectue des entrées vers la sortie.

La décomposition orientée vitesse dans chaque cellule progresse des feuilles vers la racine. Elle commence par la feuille ayant la fonction de coût minimale. L'approche monte dans l'arborescence factorisée de cellule jusqu'au nœud v où le nombre de variables du cône prédécesseur de v est supérieur à k_0 ou le Coût_Global_Cellule de v est supérieur au Coût_Global_Cellule de son prédécesseur. Puis, les prédécesseurs de v sont ordonnés par ordre croissant de fonction de coût. Parmi ces prédécesseurs, nous allons sélectionner des candidats pour créer une nouvelle cellule LUT-k_0 en limitant l'augmentation de la fonction de coût au nœud en traitement.

Si tous les prédécesseurs de v ont la même fonction de coût, ils sont tous candidats.

Si non, les prédécesseurs de v qui ont la fonction de coût inférieur à la fonction de coût maximale sont sélectionnés comme candidats.

Exemple:

Dans la figure 3.14, le nœud courant est N1, supposons que ses prédécesseurs aient les coûts globaux comme suit:

Coût_Global_Cellule (N2) = 4; Coût_Global_Cellule (N3) = 2; Coût_Global_Cellule (N4) = 3; Coût_Global_Cellule (N5) = 4; Coût_Global_Cellule (N6) = 6; Coût_Global_Cellule (N7) = 6;

Donc, les candidats sélectionnés sont N2, N3, N4, N5.

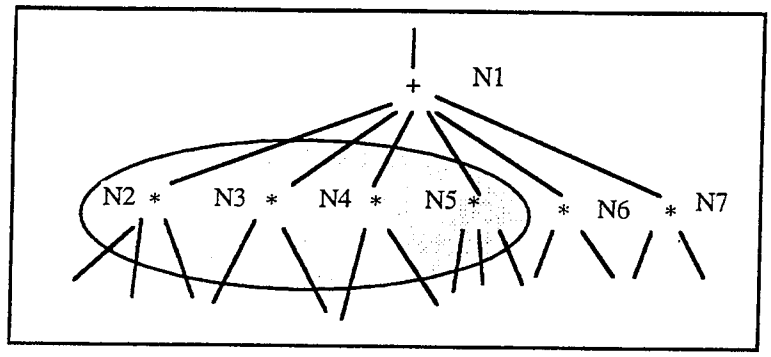


Figure 3.14 Chercher les candidats pour créer une nouvelle LUT-k_0

Les cône prédécesseurs des candidats sont regroupés avec les règles suivants:

- Le nombre de variables d'entrée de cet ensemble de cônes n'est pas supérieur à k_0.
- La priorité est donnée aux candidats ayant la plus petite fonction de coût.

Cet ensemble des cônes prédécesseurs forme une nouvelle cellule LUT-k_0 et sera remplacé par une nouvelle entrée dans l'arborescence.

Exemple:

Considérons le même exemple que précédent.

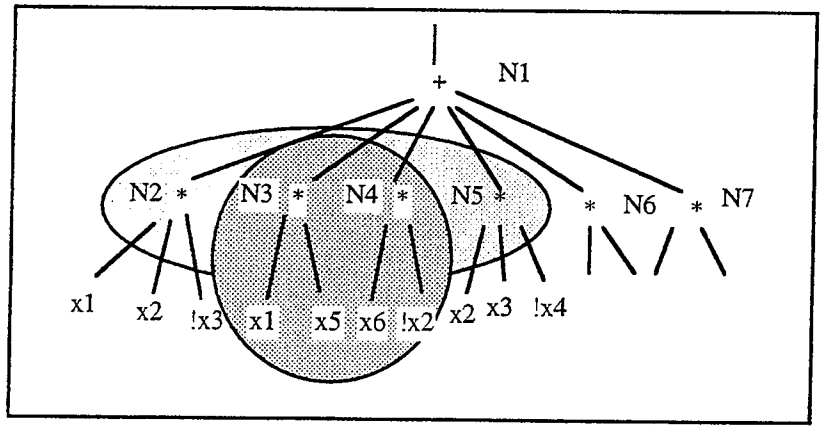


Figure 3.15a Regroupement des cônes prédécesseurs des candidats

Dans la figure 3.15a, les cônes prédécesseurs des candidats détectés dans l'étapes précédente (N2, N3, N4, N5) seront candidats pour le regroupement. Les cône prédécesseurs de N3 et de N4 ayant la fonction de coût minimale (3) seront choisis d'abord. Ces 2 cônes vérifient les conditions de regroupement (le nombre de variables d'entrée de ces 2 cônes n'est pas supérieur à 4). Les candidats N2 et N5 ne peuvent plus être ajoutés dans l'ensemble de ces cône car le nombre de variables d'entrée dépassera 4. Les cônes N3. N4 seront remplacés par une nouvelle variable d'entrée x7.

Nous obtenons donc le résultat illustré dans la figure 3.15b

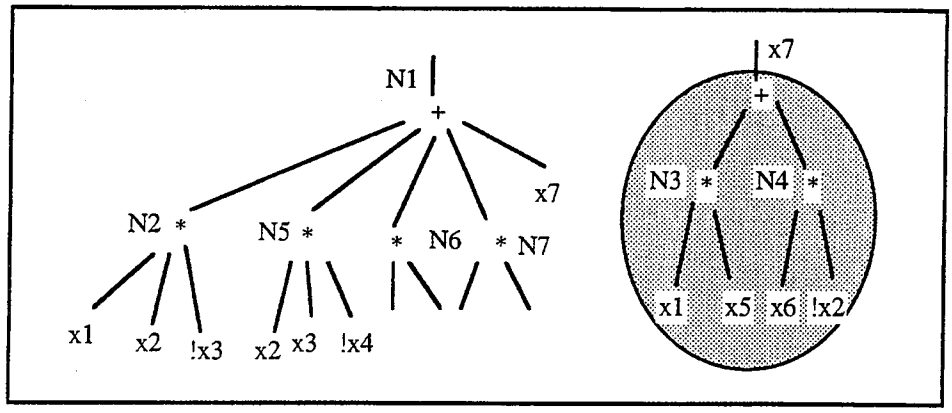


Figure 3.15b Résultat du regroupement orienté vitesse

L'approche recommence avec la feuille ayant la fonction de coût minimale jusqu'à ce que le nombre de variables d'entrée de la cellule en traitement ne soit pas supérieur à k_0.
La figure 3.16 illustre la décomposition technologique orientée vitesse dans une cellule. Pour simplifier l'illustration, nous prenons seulement deux composants de la fonction de coût: coût global et coût local et des hypothèses de IV.2. Le Coût_Global_Cellule de la cellule résultat est 4 tandis que le coût global de la cellule résultat dans la décomposition technologique orientée surface est 5 (figure 3.17).

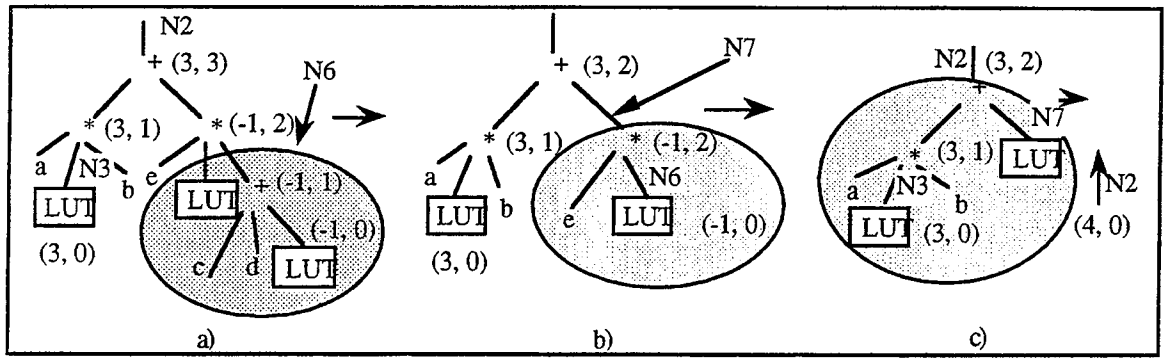


Figure 3.16 Décomposition technologique orientée vitesse

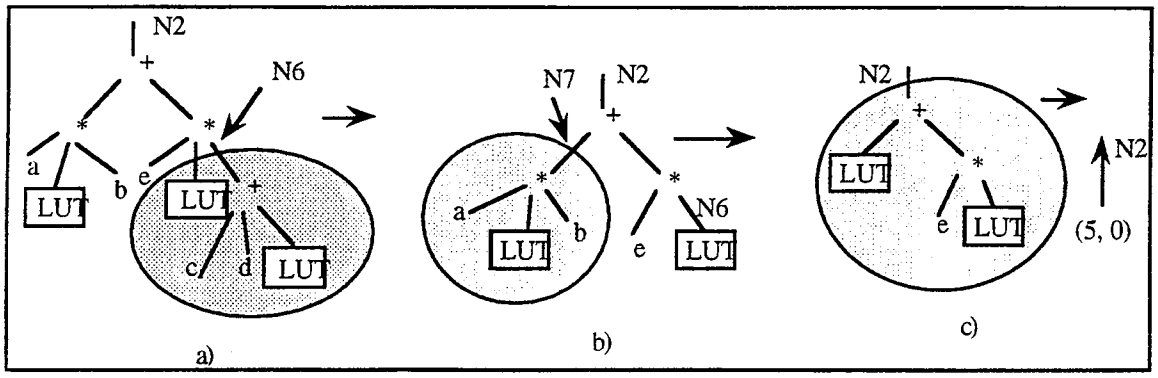


Figure 3.17 Décomposition technologique orientée surface

Correction du réseau après chaque décomposition technologique locale pour une cellule:

Le nombre de variables d'entrée de la cellule après la décomposition peut être inférieur à k_0. Pour utiliser le mieux possible le LUT-k_0 et optimiser la vitesse, la cellule ayant le nombre de variables d'entrée inférieur à k_0 sera candidat à la réinjection. Les critères de sélection sont:

- Si cette cellule est critique, elle sera réinjectée dans tous ses successeurs dans les chemins critiques.
- Si la sortance de la cellule est inférieur à m et le gain en terme du nombre de littéraux de la sous-fonction associée à la cellule est inférieur à n (m et n dépendent de la technologie), la cellule sera réinjectée dans tous ses successeurs.

Les algorithmes sont présentés comme suit:

Décomposition technologique globale (Zone Critique)
* Ordonner toutes les cellules de la zone critique par ordre croissant de leur profondeur dans le réseau.
* La méthode de décomposition technologique locale orientée vitesse ***Décomp_techno_locale*** est appliquée à chaque cellule de la zone critique en commençant par la cellule la moins profond.
* Après la décomposition pour chaque cellule, la sous-fonction associée peut être réinjectée dans les chemins critiques si elle vérifie des conditions de réinjection (voir l'algorithme ci-dessous)

Décomposition_technologique_globale (Zone Critique)

Ordonner les sous-fonction SF de la zone critique par ordre croissant en terme de profondeur;
Pour (toutes les cellules LUT-k de la zone critique)
Tant que (nombre de variables de la cellule est supérieur à k_0)
Décomposition_technologique_locale (cellule);
Si ((le nombre de variables de la cellule après la décomposition est inférieur à k_0) et (la cellule est critique) et (le gain de la cellule est inférieur à n))
/* n dépend de technologie */
Réinjecter la cellule dans la zone critique.

De même manière, la décomposition technologique orientée vitesse locale appliquée à chaque sous-fonction commence par des feuilles. L'algorithme est le suivant:

Décomposition_technologique_locale d'une cellule

* Si la cellule est critique ou sensible:
  - Ordonner les entrées de la cellule par ordre croissant de fonction de coût.
  - Regroupement orientée vitesse pour la cellule, commençant par l'entrée ayant la fonction de coût la plus petite..
* Sinon
  - Regroupement orientée surface pour la cellule.

Décomposition_technologique-locale d'une cellule

Si (la cellule est critique ou sensible)

Ordonner les entrées de la cellule par ordre croissant en terme de fonction de coût;

i_{Min} = entrée de la cellule et sa fonction de coût est minimale;

Regroupement orienté vitesse (i_{Min}, cellule)

Sinon

Regroupement orienté surface;

Regroupement_Orienté_Vitesse (i_{Min}, cellule)

Dans l'arborescence factorisée associée à la cellule:

* i_{Min} est une entrée dont la fonction de coût est minimale.
* Le **Nœud_Courant** u est le succeseur de i_{Min}.
* Tant que le nombre de variables du **Nœud_Courant** u n'est pas supérieur à k_0 et les **Coût_Global_Cellule** du **Nœud_Courant** et de ses prédécesseurs sont égaux

  Nœud_Courant u = successeur du **Nœud_Courant**;
* Ordonner les prédécesseurs du **Nœud_Courant** u dans l'arborescence factorisée de la cellule par ordre croissant de fonction de coût.
* Chercher les candidats à regrouper, notés **Nœuds_à_Regrouper**, dans ces prédécesseurs du Nœud_Courant.
* Regrouper des nœuds dans **Nœuds_à_Regrouper** pour créer une cellule LUT-m (la procédure ***Regrouper***) en minimisant la fonction de coût.

Regroupement_Orienté_Vitesse (i_{Min}, cellule)

Nœud_Courant = succ (i_{Min}) / Coût (i_{Min}) = Min (Coût (i_k)) $\forall$ i_k sont les entrées de la cellule

Tant que ((le nombre de variables du Nœud_Courant est inférieur à k_0) et
(Coût_Global_Cellule du Nœud_Courant est égal à Coût_Global_Cellule maximal des prédécesseurs du Nœud_Courant))
Nœud_Courant = successeur du Nœud_Courant ;

Ordonner les prédécesseurs du Nœud_Courant par ordre croissant de fonction de coût;

Chercher les candidats Nœuds_à_Regrouper parmi les prédécesseurs du Nœud_Courant pour être regroupés;

Regrouper (Nœud_Courant, Nœuds_à_Regrouper)

Créer une nouvelle LUT.

Chercher les candidats (prédécesseurs du Nœud_Courant)

* Si tous les prédécesseurs du **Nœud_Courant** ont la même fonction de coût, ils sont tous sélectionnés comme candidats à regrouper.
* Sinon
  - Ordonner les prédécesseurs du **Nœud_Courant** u par ordre croissant de fonction de coût.
  - Soit C_{Max} la fonction de coût maximale des prédécesseurs du nœud courant.

  Les prédécesseurs sélectionnés à regrouper sont ceux qui ont la fonction de coût inférieur à C_{Max}.

Chercher les candidats (Prédécesseurs du Nœud_Courant)

Ordonner les prédécesseurs p_i du Nœud_Courant par ordre croissant de fonction de coût:

Pour (Tous les prédécesseurs p_i du Nœud_Courant)
  Si (Max (Coût-Global_Cellule (p_i)) > Min (Coût_Global_Cellule (p_i)))
    Nœuds_à_Regrouper =
      {p_0 / p_0 est un prédécesseur du Nœud_Courant,
      Coût_Global_Cellule(p_0) = Min (Coût_Global_Cellule(p_i) }
    Tant que (Coût_Global_Cellule(p_j) < Max (Coût_Global_Cellule(p_i))
      Nœuds_à_Regrouper = Nœuds_à_Regrouper ∪ {p_j};
  Sinon Si (Max (Coût_Local (p_i)) > Min (Coût_Local (p_i)))
    Nœuds_à_Regrouper =
      {p_0 / p_0 est un prédécesseur du Nœud_Courant,
      Coût_Local (p_0) = Max (Coût_Local (p_i) }
    Tant que (Coût_Local (p_j) > Min (Coût_Local (p_i))
      Nœuds_à_Regrouper = Nœuds_à_Regrouper ∪ {p_j};
  Sinon Si (Max(Nombre_de_Variables de cône prédécesseur de p_i) >
      Min (Nombre_de_Variables de cône prédécesseur de p_i))
    Nœuds_à_Regrouper =
      {p_0 / p_0 est un prédécesseur du Nœud_Courant,
      Nombre_de_Variables (p_0) = Max (Nombre_de_Variables (p_i) }
    Tant que (Nombre_de_Variables (p_j)>Min (Nombre_de_Variables (p_i))
      Nœuds_à_Regrouper = Nœuds_à_Regrouper ∪ {p_j};
  Sinon
    Nœuds_à_Regrouper = tous les prédécesseurs du Nœud_Courant;

Regrouper (Nœud_Courant, Nœuds_à_Regrouper)

* Ordonner les nœuds candidats de **Nœuds_à_Regrouper** par ordre croissant de fonction de coût.

* p1 est un candidat dont la fonction de coût est minimale.

* **Ensemble_des_Fils** = {p1}

* Ajouter dans **Ensemble_des_Fils** d'autres candidats tant que sup (ensemble_des_fils) $\leq k_0$. Choisir en priorité les candidats dont la fonction de coût est minimale.

* Insérer un point de coupure sous le **Nœud_Courant** en regroupant les nœuds dans **Ensemble_des_Fils** .

* Le sous-arbre contenant les cône prédécesseurs des nœuds dans **Ensemble_des_Fils** sous le point de coupure est remplacé par une feuille correspondant à la nouvelle cellule dans le réseau booléen.

Regrouper (Nœud_Courant, Nœuds_à_Regrouper)

Ordonner les nœuds p_j de Nœuds_à_Regrouper par ordre croissant en terme de fonction de coût;

Coût (p1) = Min (Coût (n_i)) $\forall$ $n_i \in$ Nœuds_à_Regrouper

Ensemble_des_Fils = {p1};

tant que (/sup (Ensemble_des_Fils $\cup$ {n_i})/ $\leq$ m)

Ensemble_des_Fils = Ensemble_des_Fils $\cup$ {n_i}

Insérer un point de coupure sous le nœud u;

Le sous-arbre comprenant des cônes prédécesseurs des nœuds dans Ensemble_des_Fils sous le point de coupure est remplacé par une nouvelle feuille.

V DECOMPOSITION TECHNOLOGIQUE SPECIFIQUE AUX CIBLES

La décomposition technologique orientée vitesse pour les FPGAs à base de LUT a été présentée ci-dessus. Dans l'étape suivante, nous ajoutons des techniques supplémentaires à ces algorithmes pour les adapter aux différentes cibles en tenant compte de la configuration de leur cellule de base et de leur architecture d'interconnexion. Les cibles que nous traitons dans cette thèse sont la famille 4000 de Xilinx et Flex8000 d'Altera. Nous allons présenter les algorithmes spécifiques à chaque cible.

V.1 Décomposition technologique orientée vitesse pour la famille XC4000

En analysant la structure du CLB et l'architecture d'interconnexion de la famille XC4000, nous avons remarqué que:

* Le délai interne de la connexion entre les blocs logiques FMAP/GMAP et HMAP dans le CLB est beaucoup plus petit que celui du réseau d'interconnexion. Pour améliorer la performance temporelle, la décomposition technologique devrait favoriser la génération des combinaisons FMAPs et HMAPs.

Les conditions à vérifier pour créer un HMAP sont:

- Le nombre de variables d'entrée du nœud considéré est au plus 3.
- Deux parmi les trois entrées de ce nœud viennent des sorties de FMAPs.
- Au moins l'un de ces deux FMAPs a une sortance unitaire.

* La connexion par le réseau d'interconnexion entre les bascules/portes trois états et les buffers de sortie/entrée peut être évitée par l'intégration de ces bascules/portes trois états dans les blocs entrée/sortie. Ceci permet d'améliorer considérablement la performance temporelle du circuit.

V.1.1 Modèle de délai

Le modèle de délai que nous utilisons pour la famille XC4000 est le modèle de délai nominal car ce modèle prend en compte correctement le délai d'interconnexion qui est proportionnel à la sortance des signaux.

Considérons qu'un générateur de fonctions (FMAP ou HMAP) correspond à une cellule du réseau booléen, le délai du chemin d'une entrée i à un nœud j est calculé d'après la formule suivante:

$$D(i, j) = D(i, j-1) + \Delta t_j + \delta * FO_{j-1}$$

* j-1 est le nœud prédécesseur de j dans le chemin p (i, j)
* D (i, j-1) est le délai de l'entrée i à nœud j-1
* Δt_j est le délai intrinsèque de la cellule logique j.
* δ est le délai dû à une sortance
* FO_{j-1} est la sortance du nœud j-1.

Le temps de traversée Δt_j d'un générateur de fonctions dépend du type de générateur (LUT-4/FMAP ou LUT-3/HMAP) et de la performance du boîtier (speed grade)

$\Delta t_j = \Delta t_j$ (FMAP/HMAP, speed grade)

$\delta = \delta$ (speed grade)

δ est choisi expérimentalement. Les valeurs numériques Δt_j des boîtiers de la famille Xilinx sont données dans l'annexe.

V.1.2 Méthode de la décomposition technologique orientée vitesse dédiée à la famille XC4000

Tous les algorithmes de la décomposition technologique orientée vitesse pour les FPGAs à base de LUT présentés ci-dessus sont réutilisés pour la famille XC4000 en ajoutant des traitements supplémentaires pour améliorer les résultats. Le modèle de délai prend en

compte le délai d'interconnexion et la différence entre les blocs logiques FMAPs et HMAPs. Ceci permet d'estimer plus précisément la performance temporelle du circuit. La décomposition technologique orientée vitesse favorise la génération des combinaisons FMAPs/HMAPs afin d'éviter l'interconnexion qui augmente considérablement le délai du circuit. La modification de la décomposition technologique orientée vitesse est présentée comme suit:

Nous allons appliquer tout d'abord la décomposition technologique orientée vitesse dans une cellule pour $k_0 = 3$. Si la nouvelle cellule vérifie les conditions pour créer une cellule HMAP, elle sera marquée HMAP. Si non, l'approche recommence avec $k_0 = 4$ pour créer un FMAP.

La figure 3.18 illustre la génération d'un HMAP. Dans la figure 3.18a, la cellule N3 vérifie les conditions pour créer un HMAP. Par contre, dans la figure 3.18b, les 3 entrées N1, N2, N5 de N3 ont une sortance de deux. Elles ne vérifient pas les conditions pour créer une cellule HMAP. Donc N3 ne peut pas être créée comme un HMAP.

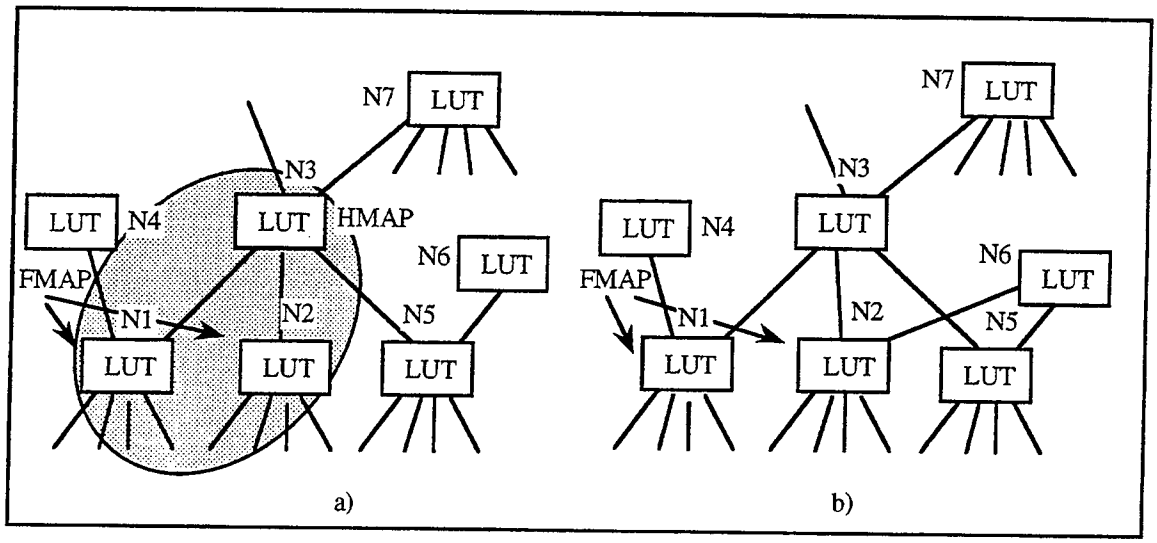


Figure 3.18 Génération d'un bloc HMAP (LUT-3)

Regrouper_XC4000 (Nœuds_à_Regrouper)
1) Ordonner les nœuds de Nœuds_à_Regrouper par ordre croissant de fonction de coût.
1) Regrouper les nœuds de Nœuds_à_Regrouper (algorithme ***Regrouper*** pour $k_0 = 3$) pour obtenir tout d'abord un ensemble **Ensemble_des_Fils** d'au plus 3 variables.
2) Si les variables d'entrée des cônes prédécesseurs des nœuds dans **Ensemble_des_Fils** vérifient les conditions pour créer une combinaison de FMAP/HMAP, créer une nouvelle cellule de trois variables correspondant à un HMAP.
3) Sinon, appliquer l'approche générale ***Regrouper*** avec $k_0 = 4$ pour créer un nouveau FMAP.

Avant d'appeler cette procédure, nous affectons $k_0 = 3$, une valeur correspond au nombre de variables d'entrée de la cellule HMAP.

Regrouper_XC4000 (Nœuds_à_Regrouper)

$k_0 = 3$; /* $k_0 = 3$ correspond à la cellule HMAP */
Regrouper (Nœud_Courant, Nœuds_à_Regrouper)
Si (les entrées de la nouvelle cellule vérifient les conditions de création d' une cellule HMAP)
    Créer une cellule HMAP;
Sinon
    $k_0 = 4$;
    Regrouper (Nœud_Courant, Nœuds_à_Regrouper)

V.1.3 Passage de contraintes

Le format XNF de Xilinx donne aux concepteurs la possibilité d'indiquer certaines contraintes d'optimisation de performance temporelle à l'outil de placement et routage de Xilinx (PPR). Nous utilisons cette caractéristique pour améliorer la performance temporelle.
Parmi les contraintes que nous pouvons passer à l'outil PPR de Xilinx, l'attribut BLKNM et les signaux critiques sont les plus intéressants car ils permettent de diminuer directement le délai.

* **BLKNM:** PPR (de Xilinx) va regrouper les FMAPs et HMAPs qui ont le même attribut BLKNM dans un même CLB. Ceci permet de raccourcir le délai entre ces blocs logiques grâce à leur connexion interne dans le CLB.
Pourtant, ces attributs ne peuvent être appliqués qu'aux FMAPs et HMAPs dans les chemin critiques pour satisfaire les exigeances de la synthèse sans compliquer le problème de routage. La figure 3.19 montre la combinaison des blocs dans le même CLB.

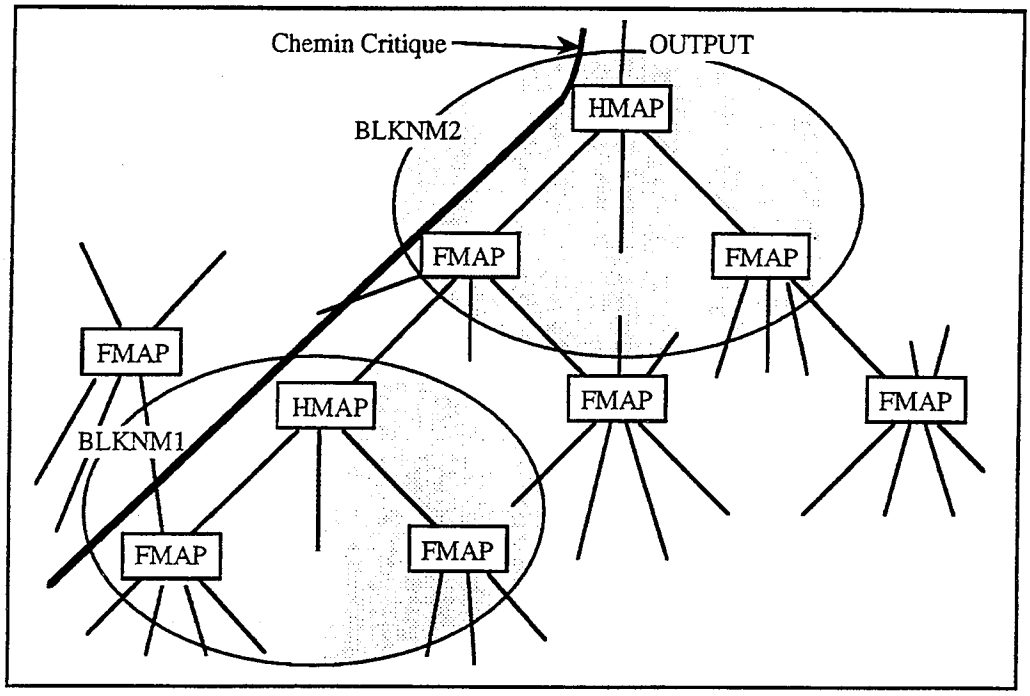


Figure 3.19 Regroupement des blocs du même attribut BLKNM

*** Signaux critiques:**

Après l'optimisation de la performance temporelle, les chemins critiques sont bien déterminés. Nous pouvons passer ces informations à l'outil PPR de Xilinx pourqu'il puisse améliorer encore la performance temporelle de la conception en favorisant les connexions des chemins critiques pendant le placement et le routage.

Remarque:

Le gain obtenu par le passage des informations concernant les chemins critiques et les BLKNMs n'est pas toujours positif car ces contraintes que nous imposons à l'outil PPR de Xilinx peuvent compliquer le problème de placement et de routage, surtout quand la ressource de routage est limitée.

*** Intégration des bascules ou des portes trois états dans les blocs d'entrée/sortie:**

Dans le cas où des bascules ou des portes trois états sont connectées aux buffers d'entrée/sortie, l'intégration de ces éléments dans les blocs entrée/sortie améliore considérablement la performance du circuit.

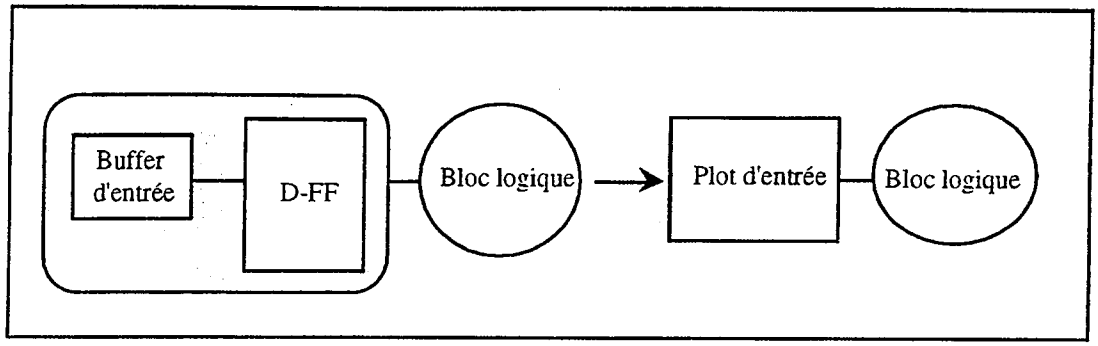


Figure 3.20 Intégration des bascules dans les blocs entrée/sortie

V.2 Décomposition technologique orientée vitesse pour la famille Flex8000 d'Altera

Grâce à l'analyse de l'architecture du boîtier Flex8000 présentée dans le chapitre 1 d'un point de vue de l'optimisation de performance temporelle, nous pouvons remarquer que:

* L'utilisation des chaînes CASCADE permet d'intégrer une mi-couche (porte ET) dans un élément logique (LE) sans quasiment augmenter le délai. De plus, la chaîne CASCADE utilise le réseau de connexion locale qui est plus performant que le réseau d'interconnexion général. Ceci améliore encore la performance du circuit.
Les conditions pour qu'une porte ET puisse être intégrée comme chaîne CASCADE dans l'élément logique LE sont:
  - Les deux nœuds correspondant à deux entrées de cette porte ET n'ont qu'un successeur pour chaque nœud.
  - Les blocs correspondant aux deux entrées de cette porte ET ne peuvent pas être en même temps les sorties d'autres CASCADEs.

* L'utilisation des chaînes CARRY dans les opérateurs arithmétiques augmente beaucoup la vitesse de ces opérateurs.

* L'intégration des bascules à l'entrée ou à la sortie dans les blocs d'entrée/sortie minimise la surface et diminue également le délai.

* Nous distinguons deux types de blocs logiques:
- Bloc LUT-4 correspondant à la partie combinatoire d'un élément logique: le délai de ce bloc LUT-4 est noté Δt_{LUT}.
- Bloc CASCADE correspondant à la chaîne CASCADE (porte ET) dont le délai appelé Δt_{CAS} est beaucoup plus faible que Δt_{LUT}.

V.2.1 Modèle de délai

Comme le délai d'interconnexion dans les boîtiers Flex8000 peut être considéré comme invariant, le modèle de délai est simplifié et ramené à la formule suivant:

$$D(i,j) = \sum_{k=i}^{j} \Delta t'_k$$

Où:

* D (i, j) est le délai du chemin de l'entrée i à un nœud j.
* Si k est un nœud correspondant à un bloc LUT-4:

$\Delta t'_k = \Delta t_{LUT} + \Delta t_{INT}$

* Si k est un nœud correspondant à un bloc CASCADE:

$\Delta t'_k = \Delta t_{CAS} + \Delta t_{INT}$

Δt_{INT} représente le délai d'interconnexion

V.2.2 Méthode de la décomposition technologique orientée vitesse dédiée à la famille Flex8000

Tous les algorithmes de décomposition technologique orientée vitesse pour les FPGAs à base de LUT sont réutilisés pour la famille Flex8000 en ajoutant des traitements supplémentaires pour tenir compte des particularités (tels que CASCADE, CARRY...) de cette famille. Le modèle de délai est simplifié sans perdre l'efficacité et la généralité. L'approche favorise la génération des chaînes CASCADE dans la région critique pour réduire le délai des chemins critiques. Les chaînes CARRY sont utilisées dans les opérateurs arithmétiques pour implémenter des circuits arithmétiques à haute vitesse.
Dans la figure 3.21, l'utilisation des portes CASCADE permet d'intégrer les portes ET dans les LEs N1, N2 et N3 sans augmenter le délai et diminuer la surface à la fois.

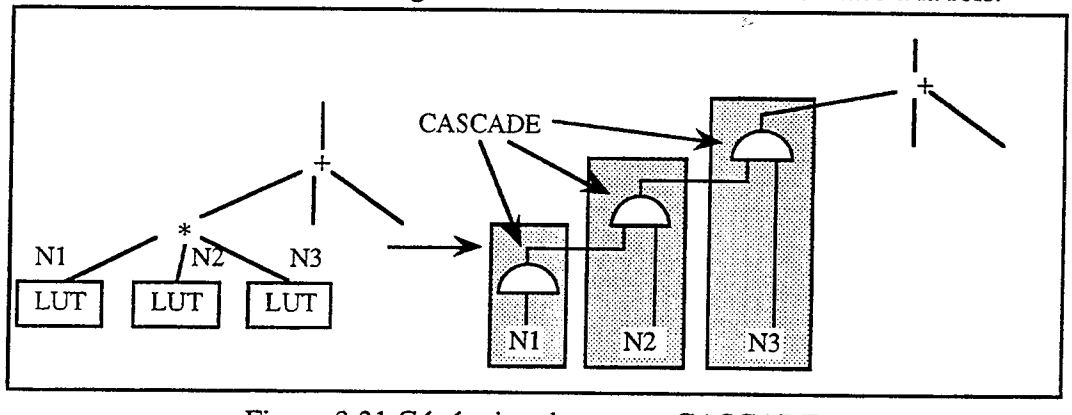


Figure 3.21 Génération des portes CASCADE

La modification de la procédure ***Regrouper*** est présentée comme suit::

Regrouper_Orienté_Vitesse_Flex8000 (i_{Min}, cellule)

i_{Min} est une entrée dont la fonction de coût est minimale.

* Si le successeur de i_{Min} est un nœud ET

    Si ce successeur vérifie les conditions pour créer une chaîne CASCADE

        Créer une cellule CASCADE

* Sinon

    $k_0 = 4$;

    Regrouper_Orienté_Vitesse (i_{Min}, cellule)

CHAPITRE 4

RESULTATS ET EVALUATIONS

INTRODUCTION

Des techniques de décomposition technologique et d'optimisation temporelle des réseaux booléens sur FPGAs à base de LUT (en particulier les deux familles FPGAs XC4000 de Xilinx et Flex8000 d'Altera) ont été présentées dans le chapitre précédent. Toutes ces méthodes sont implantées dans l'outil commercial de synthèse logique ASYL+.

Pour évaluer ces techniques d'optimisation, nous allons présenter tout d'abord la comparaison des performances temporelles pour deux options d'optimisation dans ASYL+: optimisation orientée surface et vitesse.

Ensuite, nous comparerons les résultats de notre approche d'optimisation temporelle avec les outils de synthèse logique connus dans le commerce tels que ceux d'Exemplar Logic Inc., de Maxplus2 fourni par Altera et de XACT de Xilinx.

Cette comparaison est faite à la fois sur un ensemble d'exemples réels conçus sur FPGA provenant de différentes sociétés et sur des benchmarks académiques internationaux MCNC [MCN89] et ISCAS [ISC89].

Comme ces outils commerciaux n'optimisent que soit la fréquence de fonctionnement des circuits séquentiels soit le délai maximal du circuit ombinatoire, la comparaison ne porte que sur la fréquence pour les circuits séquentiels ou sur le délai maximal pour les circuits combinatoires alors que notre approche est capable d'optimiser d'autres options.

Les résultats (fréquence ou délai) dans les tableaux sont donnés par l'outil de placement et de routage PPR de Xilinx pour l'optimisation sur XC4000 et par l'outil de placement et de routage Maxplus2 pour l'optimisation sur Flex8000.

Ces comparaisons sont faites sur Sparc-10 pour ASYL+, Maxplus2, XACT et sur PC Pentium 90Mhz avec 16 Mo RAM pour Exemplar. Pour tous les outils, nous avons utilisé les options qui donnent la meilleur performance.

I. COMPARAISON DES RESULTATS DES OPTIONS D'OPTIMISATION DANS ASYL+

Nous utilisons des conventions suivantes:

• La colonne #LC (resp. #CLB) présente le nombre de cellules (resp. blocs logiques: Configurable Logic Block) utilisées pour la technologie Flex8000 (resp. XC4000).

• La colonne Fréq. (Mhz) montre la fréquence de fonctionnement du circuit séquentiel en Mhz.

• La colonne Délai (ns) indique le délai maximal du circuit combinatoire en nanosecond.

• La colonne Prof. présente la profondeur du chemin critique.

• La colonne G1(V/S) présente le gain en performance temporelle après le placement et le routage en pourcentage de l'option vitesse sur l'option surface.

• La colonne G2 (V/S) indiquer le gain en profondeur du chemin critique en pourcentage de l'option vitesse sur l'option surface.

• La colonne A (V/S) signifie l'augmentation en pourcentage du nombre de cellules utilisées avec l'option vitesse par rapport à l'option surface.

I.1 Pour la famille Flex8000

Les gains moyens en pourcentage en vitesse de l'optimisation temporelle sur l'optimisation de surface dans ASYL+ pour les benchmarks ISCAS, MCNC et les conceptions réelles sont respectivement de 35,79%, 30,13% et 32,74%. En terme de profondeur, les gains sont encore plus importants 45,85%, 46,27% et 56,56%. La différence entre les gains en terme de profondeur et le gain réel confirme que la profondeur en nombre de LUT n'est pas le seul critère pour évaluer la performance temporelle comme expliqué dans le chapitre 3.

D'ailleurs, l'augmentation en pourcentage du nombre de cellules utilisées avec l'option vitesse par rapport à l'option surface n'est que de respectivement 35,5%, 20% et 27% pour les benchmarks. Cela confirme l'efficacité et la cohérence des phases de traitements dans notre approche (réinjection contrôlée, détection de la zone critique, bon modèle de délai et la décomposition technologique orientée vitesse à base du calcul des fonctions de coût).

Noms	Optimisation orientée Surface			Optimisation Temporelle			Gain %		
	#LCs	Fréq. (Mhz)	Prof.	#LCs	Fréq. (Mhz)	Prof.	G1(V/S)	G2(V/S)	A(V/S)
s1196	220	12,37	9	344	27,02	4	118,43	125	56,36
s1238	243	10,86	10	349	24,63	5	126,80	100,00	43,62
s1423	181	3,67	32	459	9,16	12	149,59	166,67	153,59
s208	23	28,24	4	26	38,91	3	37,78	33,33	13,04
s27	5	63,69	2	5	63,69	2	0,00	0,00	0,00
s298	35	29,94	4	37	33	3	10,22	33,33	5,71
s344	44	20,74	6	62	21,55	5	3,91	20,00	40,91
s349	44	20,2	6	62	20,12	5	-0,40	20,00	40,91
s382	46	22,27	5	46	22,27	5	0,00	0,00	0,00
s5378	498	10,48	9	520	13,94	7	33,02	28,57	4,42
s9234	450	9,35	13	488	10,06	12	7,59	8,33	8,44
s953	165	12,88	8	248	16,07	5	24,77	60,00	50,30
	Délai (ns)			Délai (ns)					
c1355	83	87,9	8	82	74,4	6	18,15	33,33	-1,20

c1908	116	132,5	14	141	115,2	11	15,02	27,27	21,55
c432	85	183,3	20	117	141,8	13	29,27	53,85	37,65
c499	81	85	8	82	76,1	6	11,70	33,33	1,23
c880	104	146,9	15	236	119,8	11	22,62	36,36	126,92
Gain moyen (%)							35,7	45,85	35,50

Tableau 4.1 Comparaison des résultats des options d'optimisation orientée surface et vitesse sur le benchmark ISCAS pour la famille Flex8000

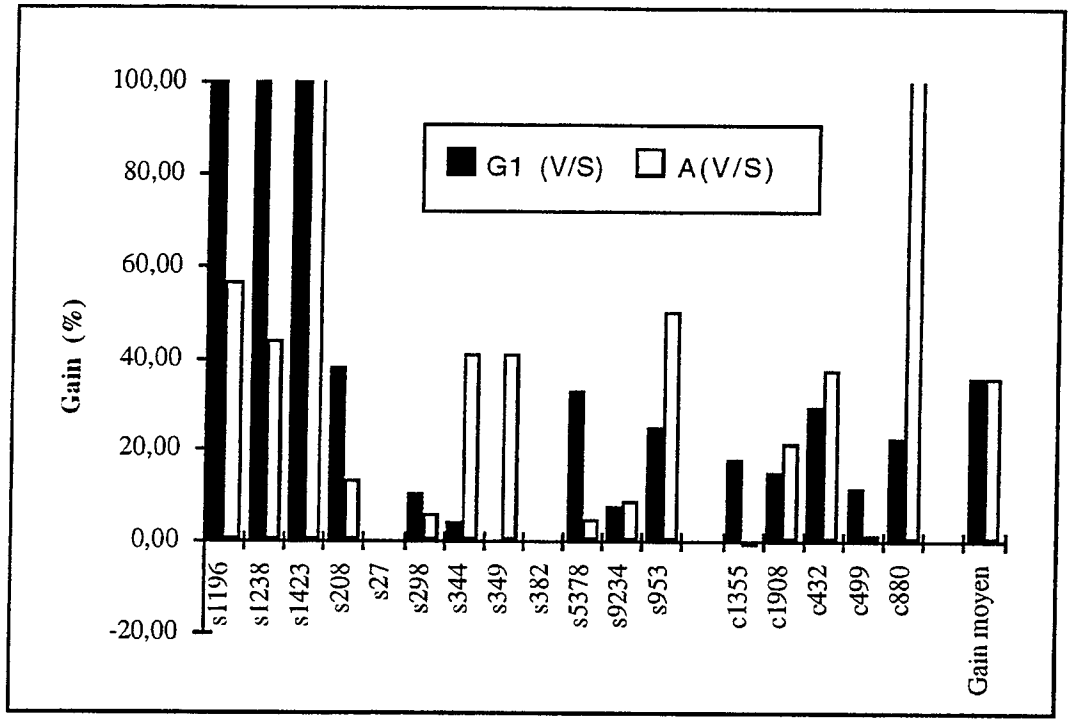


Figure 4.1 Comparaison des résultats des options d'optimisation orientée surface et vitesse sur le benchmark ISCAS pour la famille Flex8000

Noms	Optimisation Orientée Surface			Optimisation Temporelle			Gain (%)		
	Prof.	#LCs	Fréq. (Mhz)	Prof.	#LCs	Fréq. (Mhz)	G1(V/S)	G2 (V/S)	A(V/S)
actdec	5	43	20,7	3	65	28,9	39,61	66,67	51,16
androx	6	32	18,93	3	64	31,15	64,55	100,00	100,00
bases	5	73	22,32	4	89	24,09	7,93	25,00	21,92
cmd_stat	4	77	22,62	4	77	22,62	0,00	0,00	0,00
cnt16s	7	63	14,66	5	71	20,87	42,36	40,00	12,70
dps	6	72	16,33	4	103	23,98	46,85	50,00	43,06
eobpal	4	30	28,81	2	40	35,21	22,21	100,00	33,33
itiming	6	50	17,33	3	72	29,67	71,21	100,00	44,00
len1	6	36	19,37	4	54	22,02	13,68	50,00	50,00
lorain	4	81	33,67	2	89	52,35	55,48	100,00	9,88
meanf	2	16	53.76	2	18	53,76	0,00	0,00	12,50
rx1	3	21	34,84	2	25	52,79	51,52	50,00	19,05
seqs	4	37	27,39	3	33	32,25	17,74	33,33	-10,81
stat	5	76	23,14	4	76	25,77	11,37	25,00	0,00

tc3da	7	142	17,51	3	155	30,03	71,50	133,33	9,15
tx4dctp	5	32	23,2	4	36	25,9	11,64	25,00	12,50
txdata2	5	35	20,08	3	47	31,44	56,57	66,67	34,29
txf	5	42	25,51	3	39	33,55	31,52	66,67	-7,14
wordctr	5	80	20	3	114	27,32	36,60	66,67	42,50
			Délai(ns)			Délai (ns)			
hexdec	2	14	28,3	2	14	28,3	0,00	0,00	0,00
horizont	6	91	65,4	3	135	45,5	43,74	100,00	48,35
pdec2	8	21	66,2	5	35	53,4	23,97	60,00	66,67
vertical	5	60	59,3	3	86	41,5	42,89	66,67	43,33
vi_con	4	16	44	3	19	35,8	22,91	33,33	18,75
Gain Moyen							32,74	56,60	27,30

Tableau 4.2 Comparaison des résultats des options d'optimisation orientée surface et vitesse sur les conceptions réelles pour la famille Flex8000

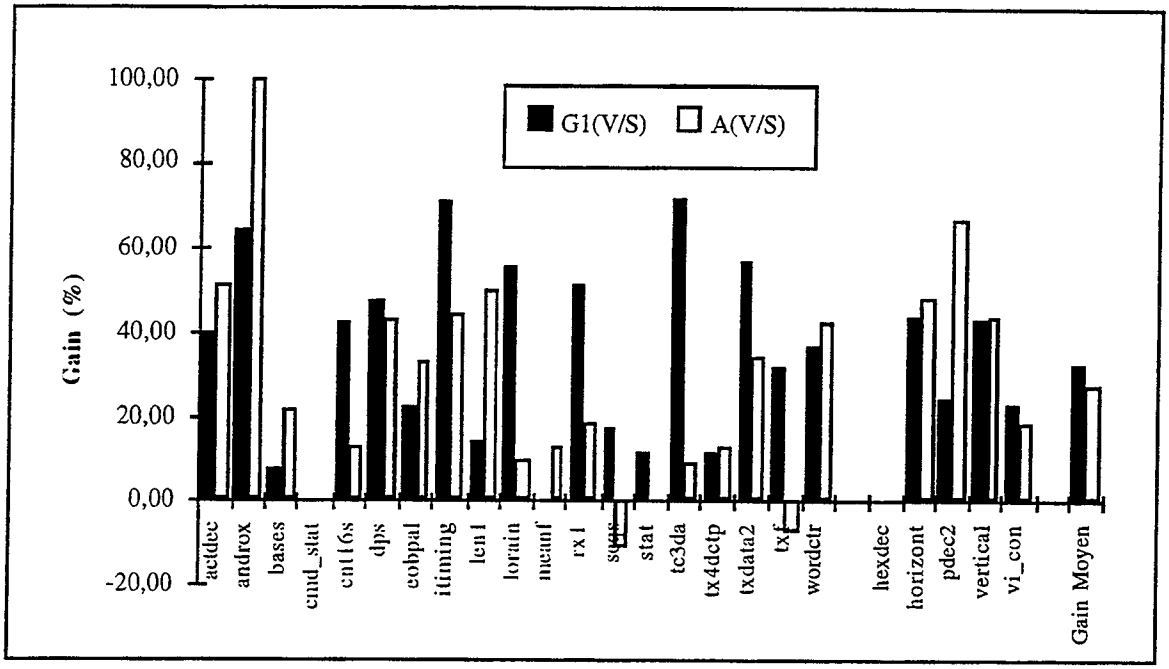


Figure 4.2 Comparaison des résultats des options d'optimisation orientée surface et vitesse sur les conceptions réelles pour la famille Flex8000

	Optimisation Orientée Surface			Optimisation Temporelle			Gain (%)		
Noms	#LCs	Délai (ns)	Prof.	#LCs	Délai (ns)	Prof.	G1(V/S)	G2(V/S)	A (V/S)
5xp1	23	38,9	3	23	38,3	3	1,57	0,00	0,00
9sym	18	64,7	7	52	56,9	5	13,71	40,00	188,89
9symml	18	64,7	7	51	56,9	5	13,71	40,00	183,33
bw	62	41,3	2	62	35,2	2	17,33	0,00	0,00
clip	39	56,4	6	55	45,9	4	22,88	50,00	41,03
count	39	72,3	7	61	50,4	4	43,45	75,00	56,41
duke2	131	94,1	9	232	81,1	6	16,03	50,00	77,10
e64	92	264,6	28	354	51,9	4	409,83	600,00	284,78

f51m	20	38,3	3	20	38,3	3	0,00	0,00	0,00
misex1	17	34	3	17	33,1	3	2,72	0,00	0,00
misex2	37	51,1	5	43	47,3	4	8,03	25,00	16,22
o64	43	48	4	43	42,7	4	12,41	0,00	0,00
rd53	8	27,7	2	8	27,7	2	0,00	0,00	0,00
rd73	11	34,8	3	11	34,8	3	0,00	0,00	0,00
rd84	24	55	5	44	46,8	4	17,52	25,00	83,33
sao2	44	56,5	5	44	54	5	4,63	0,00	0,00
vg2	27	55,5	6	29	53,2	5	4,32	20,00	7,41
z4ml	10	35,6	3	10	34,7	3	2,59	0,00	0,00
misex3	266	216	24	363	130	16	65,77	50,00	36,47
alu2	93	66,6	6	93	62,9	6	5,88	0,00	0,00
alu4	203	132,3	14	226	121	11	9,61	27,27	11,33
apex3	601	115,8	9	914	105	7	10,08	28,57	52,08
apex7	75	80	8	98	72,1	6	10,96	33,33	30,67
Gain Moyen(%)							30,13	46,27	46,48

Tableau 4.3 Comparaison des résultats des options d'optimisation orientée surface et vitesse sur les benchmarks MCNC pour la famille Flex8000

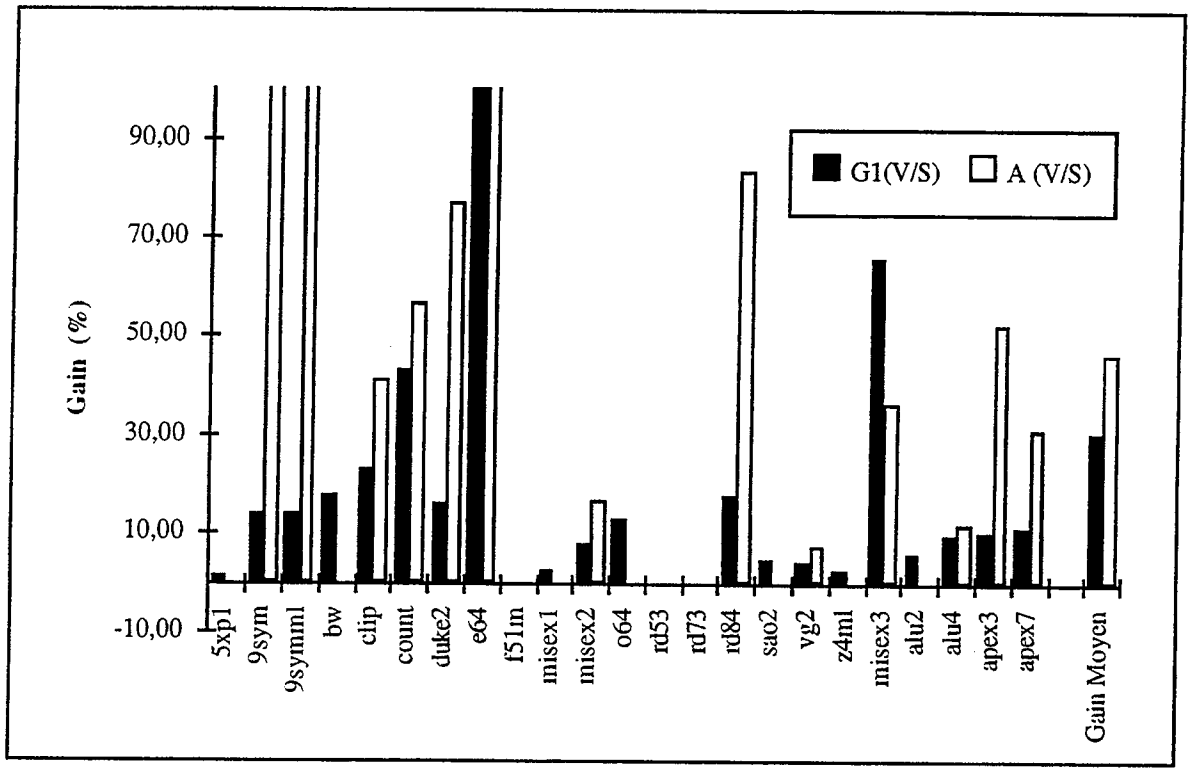


Figure 4.3 Comparaison des résultats des options d'optimisation orientée surface et vitesse sur les benchmarks MCNC pour la famille Flex8000

I.2 Pour la famille XC4000

Les gains moyens en pourcentage en performance de l'optimisation temporelle sur l'optimisation de surface dans ASYL+ pour les 3 benchmarks ISCAS, MCNC et les conceptions réelles sont respectivement de 14,27%, 19,37% et 19,37%.

D'ailleurs, l'augmentation en pourcentage du nombre de cellules utilisées de l'option vitesse par rapport à l'option surface n'est que respectivement de 18,4%, 16,4% et 17,3% pour les 3 benchmarks.

Nous pouvons remarquer que le gain de notre approche pour la famille XC4000 est moins important que celui pour la famille Flex8000. Cette différence du gain peut être expliquée comme suit:

Dans le boîtier de la famille Flex8000, l'interconnexion est assurée par le réseau de bus, "FastTrack Interconnect", dont le délai est quasiment constant et quasi-prédictible. Ceci permet d'avoir une estimation de performance temporelle proche de réalité. De plus, l'augmentation de surface n'influe pas beaucoup sur le routage comme dans le cas de la famille XC4000. Pour la famille Flex8000, l'approche d'optimisation temporelle augmente la surface en moyenne de 30% avec un gain en vitesse de 30% à 35%.

Dans le boîtier de la famille XC4000, l'interconnexion est assurée par un réseau de lignes horizontales et verticales avec des interrupteurs contrôlés par les SRAM. Le routage à travers des interrupteurs par l'outil propre au fabriquant rend imprédictible l'estimation de performance temporelle pendant la synthèse. De plus, le délai causé par l'interconnexion dans le délai total est égal ou même plus grand que le délai des blocs logiques. Quand le nombre de cellules augmente beaucoup en faveur de vitesse, elle empêche le routeur de trouver une optimale solution pour le routage. Donc, la performance sera dégradée. Cette remarque implique la surface ne doit pas trop augmenter pendant l'optimisation temporelle.

Pour la famille XC4000, c'est beaucoup plus difficile à obtenir un résultat similaire à celui pour Flex8000. Nous pouvons constater en regardant les résultats de Chortle-d [FRA91B] et de MIS-PGA [MUR91A]. Chortle-d donne un meilleur résultat en profondeur (2,9%) mais MIS-PGA gagne en vitesse après le placement et le routage (16%). Notre approche améliore la performance en surveillant prudemment la surface. Le gain en vitesse de l'optimisation temporelle est en moyenne de 17% avec une augmentation de surface à peu près de 15%.

Remarquons que le gain en profondeur pour la famille XC4000 (24% et 46%) est encore plus important que le gain après le placement et le routage (resp. 14% et 19%). Rappelons que les chiffres correspondants pour la famille Flex8000 sont respectivement 45%, 46% et 56% (resp. 35%, 32% et 30%). Ce grand écart entre des gains en profondeur et des gains

après le placement et le routage peut être expliqué par le délai important du à l'interconnexion dans les FPGAs dont l'interconnexion est basée sur des interrupteurs.

	Optimisation Orientée Surface			Optimisation Temporelle			Gain (%)		
Noms	CLBs	Prof.	Fréq. (Mhz)	CLBs	Prof.	Fréq. (Mhz)	G1 (V/S)	G2 (V/S)	A(V/S)
s1196	93	10	11,5	112	10	11,5	0,00	0,00	20,43
s1238	99	16	9,2	135	9	12,1	31,52	77,78	36,36
s1423	75	28	4,9	88	15	7,5	53,06	86,67	17,33
s208	8	3	23,6	8	3	23,6	0,00	0,00	0,00
s27	2	2	35,6	2	2	35,6	0,00	0,00	0,00
s298	14	3	21,8	15	3	26,7	22,48	0,00	7,14
s344	16	5	17,8	20	5	18,9	6,18	0,00	25,00
s349	17	5	17,4	20	5	21,3	22,41	0,00	17,65
s382	19	4	18,6	21	3	21,8	17,20	33,33	10,53
s5378	199	8	11,4	221	7	12,6	10,53	14,29	11,06
s953	65	7	13,5	90	5	14,6	8,15	40,00	38,46
			Délai(ns)			Délai(ns)			
c1355	33	4	75,7	33	4	75,7	0,00	0,00	0,00
c1908	47	13	140	72	11	130	7,69	18,18	53,19
c3540	155	18	174,7	155	15	162,3	7,64	20,00	0,00
c432	34	19	176,6	36	12	131,3	34,50	58,33	5,88
c499	36	7	102,9	36	5	93,3	10,29	40,00	0,00
c880	40	13	123,3	68	11	111,1	10,98	18,18	70,00
Gain Moyen							14,27%	23,93%	18,41%

Tableau 4.4 Comparaison des résultats des options d'optimisation orientée surface et vitesse sur les benchmarks ISCAS pour la famille XC4000

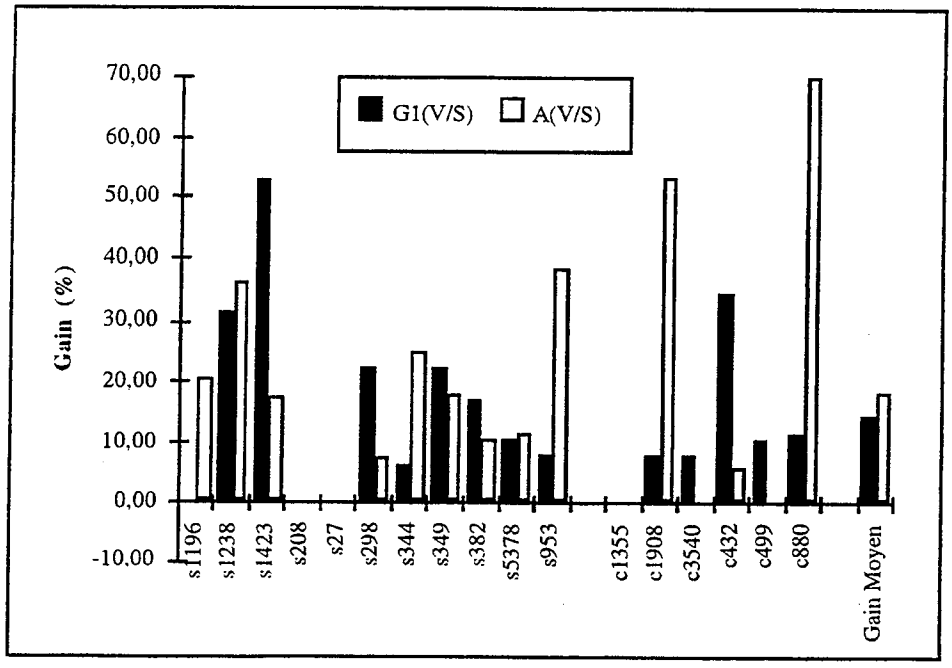


Figure 4.4 Comparaison des résultats des options d'optimisation orientée surface et vitesse sur les benchmarks ISCAS pour la famille XC4000

	Optimisation Surface			Optimisation Temporelle			Gain (%)		
Noms	CLBs	Prof.	Freq	CLBs	Prof.	Freq.	G1(V/S)	G2 (V/S)	A(V/S)
actdec	19	6	16,6	42	4	26,2	57,83	50,00	121,05
androx	13	5	18,5	18	3	29,3	58,38	66,67	38,46
bases	28	4	20,9	27	4	21,8	4,31	0,00	-3,57
cmd_stat	32	3	19,5	32	3	19,5	0,00	0,00	0,00
cnt16s	23	6	33,5	3	6	33,6	0,30	0,00	-86,96
dps	31	5	15,9	37	5	16,1	1,26	0,00	19,35
eobpal	13	3	14	13	3	15,8	12,86	0,00	0,00
itiming	19	5	22,3	25	3	25,9	16,14	66,67	31,58
len1	16	10	11,2	18	4	18,6	66,07	150,00	12,50
lorain	34	4	23	34	3	24,3	5,65	33,33	0,00
meanf	6	2	28,8	5	2	34,8	20,83	0,00	-16,67
rx1	8	3	21,4	8	3	22,4	4,67	0,00	0,00
seqs	14	3	26,2	15	3	26,3	0,38	0,00	7,14
stat	30	4	15,7	42	3	17,5	11,46	33,33	40,00
tc3dma	50	6	20,2	59	4	21,9	8,42	50,00	18,00
tx4dctmp	12	7	17,3	14	3	25,1	45,09	133,33	16,67
txdata2	13	4	23,3	21	3	27,5	18,03	33,33	61,54
txmf	15	4	23,9	14	3	28,8	20,50	33,33	-6,67
wordctr	43	3	23,7	51	3	26	9,70	0,00	18,60
			Délai (ns)			Délai (ns)			
hexdec	3	2	29,7	3	1	29,6	0,34	100,00	0,00
horizont	38	5	67,9	43	2	54,4	24,82	150,00	13,16
pmdec2	8	7	71,9	16	4	57,3	25,48	75,00	100,00
ras0en	10	8	84,8	12	6	67,2	26,19	33,33	20,00
vertical	26	4	59,4	29	2	47,1	26,11	100,00	11,54
Gain Moyen							19,37	46,18	17,32

Tableau 4.5 Comparaison des résultats des options d'optimisation orientée surface et vitesse sur les conceptions réelles pour la famille XC4000

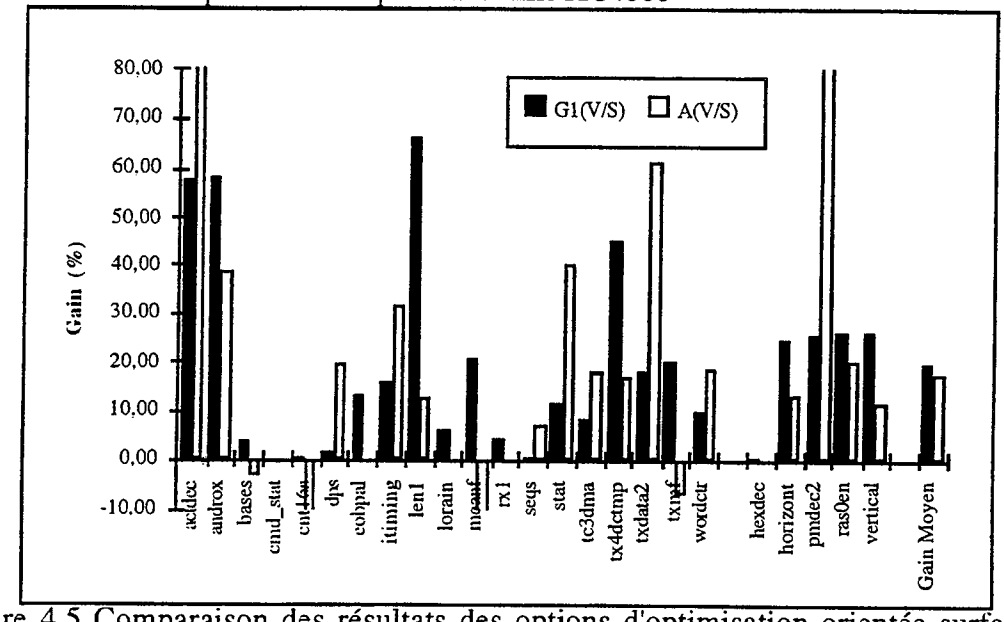


Figure 4.5 Comparaison des résultats des options d'optimisation orientée surface et vitesse sur les conceptions réelles pour la famille XC4000

	Optimisation Orientée Surface		Optimisation Temporelle		Gain (%)	
Noms	CLBs	Délai(ns)	CLBs	Délai (ns)	G1(V/S)	A(V/S)
5xp1	9	38,5	9	38,5	0,00	0,00
9sym	7	67,7	7	65,5	3,36	0,00
9symml	7	67,7	7	65,5	3,36	0,00
bw	24	44,1	24	43,9	0,46	0,00
clip	17	56,6	17	56,6	0,00	0,00
count	19	85,1	21	59,4	43,27	10,53
duke2	56	82,5	64	71,9	14,74	14,29
e64	33	207,5	78	61,2	239,05	136,36
f51m	8	42,8	8	42,6	0,47	0,00
misex1	6	35,4	6	35,4	0,00	0,00
misex2	16	51,3	20	44,6	15,02	25,00
o64	21	56,4	21	50,9	10,81	0,00
rd53	2	28,8	2	28,8	0,00	0,00
rd73	6	47,9	6	41,5	15,42	0,00
rd84	9	50,2	9	50	0,40	0,00
sao2	19	53,5	19	53,4	0,19	0,00
vg2	12	58,6	12	58	1,03	0,00
z4ml	3	36,3	3	35,9	1,11	0,00
Gain Moyen					19,37	10,36

Tableau 4.6 Comparaison des résultats des options d'optimisation orientée surface et vitesse sur les benchmarks MCNC pour la famille XC4000

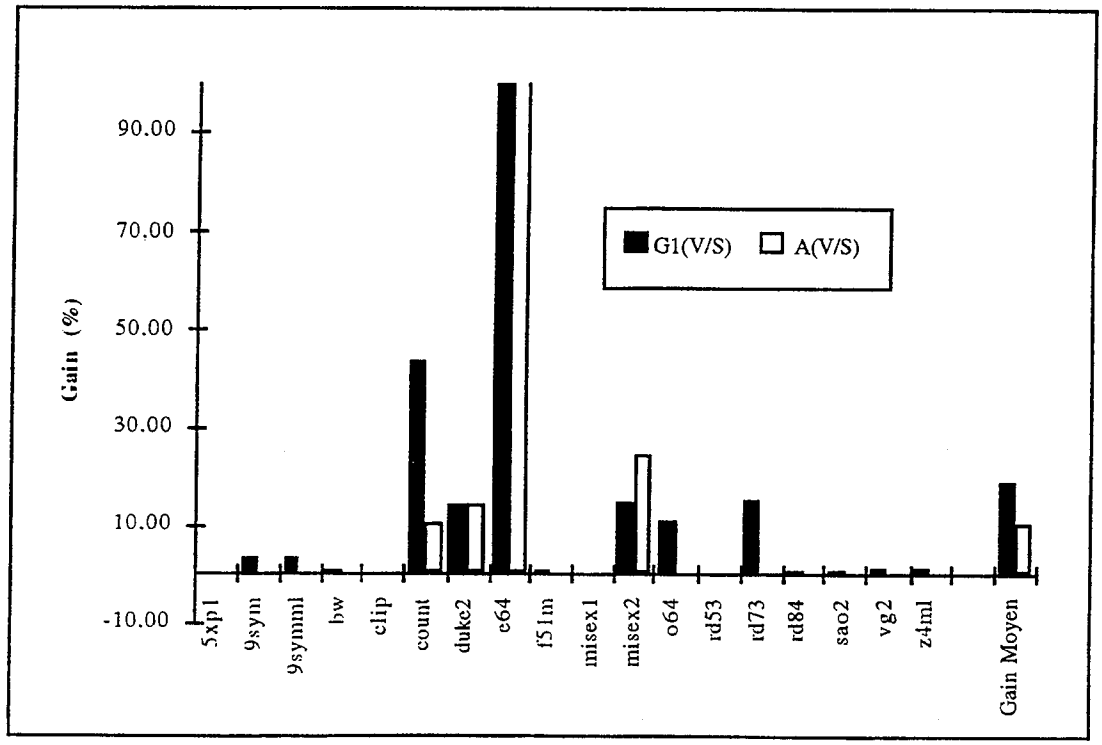


Figure 4.6 Comparaison des résultats des options d'optimisation orientée surface et vitesse sur les benchmarks MCNC pour la famille XC4000

II. COMPARAISON DES RESULTATS D'ASYL+ AVEC LES OUTILS COMMERCIAUX

• La colonne GV(A/M) (resp. GS(A/M)) présente le gain en performance (resp. en surface) en pourcentage d'ASYL+ par rapport à Maxplus2.

• La colonne GV(A/E) (resp. GS(A/E)) présente le gain en performance (resp. en surface) en pourcentage d'ASYL+ par rapport à Exemplar.

• La colonne GV(A/X) (resp. GS(A/X) indique le gain en performance (resp. en surface) en pourcentage d'ASYL+ par rapport à XACT.

II.1 Pour la famille Flex8000

Les gains moyens en pourcentage en performance temporelle (resp. en surface) d'ASYL+ par rapport à Exemplar pour les 2 benchmarks ISCAS, MCNC sont respectivement de 52,26%et 31,70% (resp. 3,8% et 35,09%).

En comparant à Maxplus2, les gains moyens en pourcentage en performance temporelle (resp. en surface) d'ASYL+ pour les 3 benchmarks ISCAS, MCNC et les conceptions réelles sont respectivement de 32,56%, 12,34% et 11,98% (resp. -20,9%, 53,64% et 4,8%).

Nous pouvons remarquer qu'ASYL gagne à la fois en vitesse et en surface sur Exemplar et Maxplus2 (sauf pour le benchmark ISCAS évoqué plus loin). Quoique notre approche ait comme premier but l'optimisation temporelle, elle a bien optimisé aussi la surface. Ceci confirme l'efficacité de notre approche: une méthode efficace de décomposition technologique orientée vitesse, fondée sur la notion de fonction de coût, appliquée à une bonne zone critique, en surveillant l'augmentation de la surface, donne un bon résultat.

Pour le benchmark ISCAS, nous avons une perte de 20% en surface par rapport à Maxplus2 (bien que nous gagnions en moyenne 32% en vitesse). Ceci apparaît normal car, quand on optimise la performance temporelle, la surface n'est pas considérée comme le critère principal. D'autre part, cette perte vient du fait que la combinaison de la partie combinatoire et des bascules pour maximiser l'utilisation des celules n'est pas bien traitée dans ASYL. Ceci dégrade le résultat au niveau de la surface pour les benchmarks ayant beaucoup de bascules comme le benchmark ISCAS..

Les tableaux de résultats sont donnés ci-dessous:

Noms	Asyl+		Maxplus2		Exemplar		Gain en Fréq (%)		Gain en surface (%)	
	#LCs	Fréq.	#LCs	Fréq.	#LCs	Fréq.	GV (A/M)	GV (A/E)	GS (A/M)	GS (A/E)
s1196	344	27,02	210	16,63	267	11,29	62,48	139,33	-38,95	-22,38
s1238	349	24,63	232	14,24	280	10,74	72,96	129,33	-33,52	-19,77
s1423	459	9,16	171	4,93	337	5,32	85,80	72,18	-62,75	-26,58
s208	26	38,91	29	26,45	22	30,76	47,11	26,50	11,54	-15,38
s27	5	63,69	6	44,05	7	45,45	44,59	40,13	20,00	40,00
s298	37	33	33	25,38	46	23,69	30,02	39,30	-10,81	24,32
s344	62	21,55	31	27,32	60	17,33	-21,12	24,35	-50,00	-3,23
s349	62	20,12	31	27,32	62	17,33	-26,35	16,10	-50,00	0,00
s382	46	22,27	49	15,36	65	22,17	44,99	0,45	6,52	41,30
s5378	520	13,94	491	11,19	747	9,38	24,58	48,61	-5,58	43,65
s9234	488	10,06	444	9,57	(*)	(*)	5,12		-9,02	
s953	248	16,07	180	13,33	199	11,6	20,56	38,53	-27,42	-19,76
Gain moyen %							32,56	52,26	-20,83	3,83

Tableau 4.7 Comparaison des résultats d'ASYL+/Exemplar/Maxplus2 sur le benchmark ISCAS pour la famille Flex8000

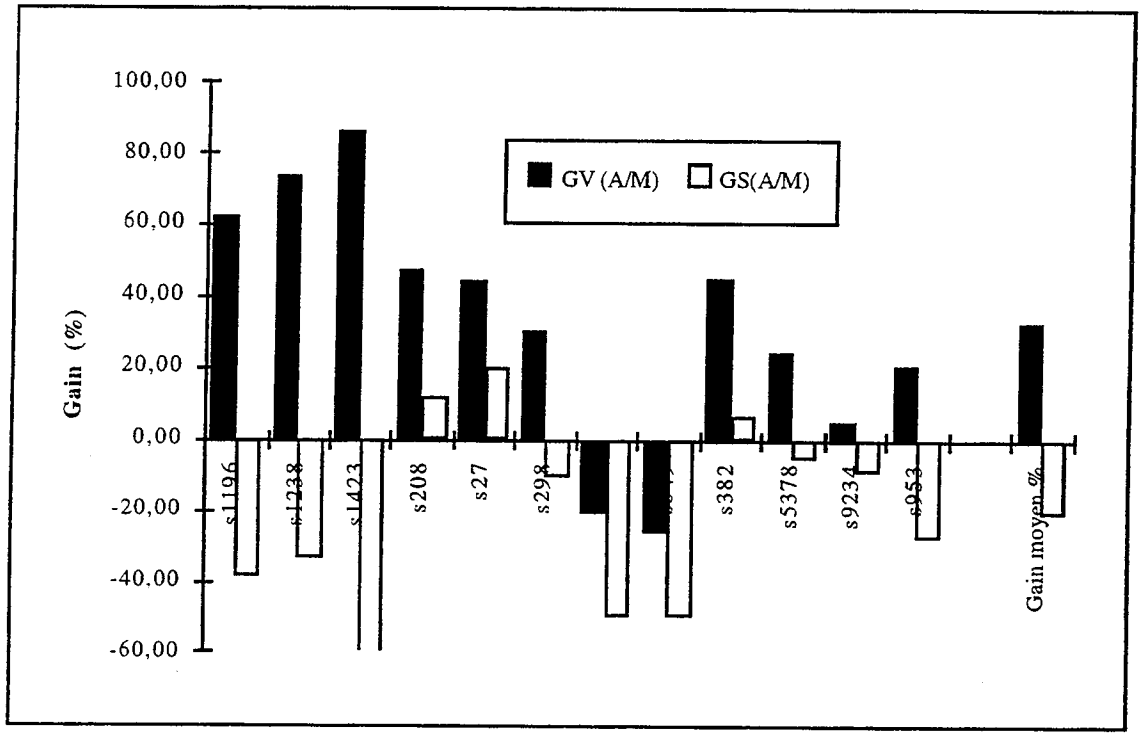


Figure 4.7a Comparaison des résultats d'ASYL+/Maxplus2 sur le benchmark ISCAS pour la famille Flex8000

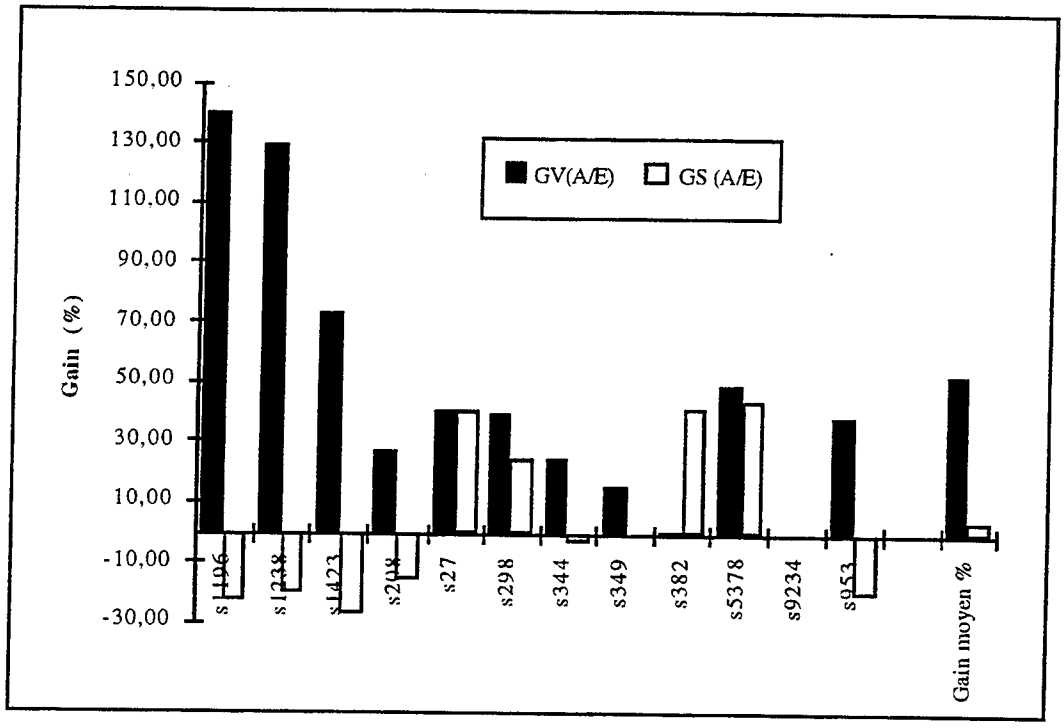


Figure 4.7b Comparaison des résultats d'ASYL+/Exemplar sur le benchmark ISCAS pour la famille Flex8000

Noms	Asyl+		Maxplus2		Gain (%)	
	#LCs	Freq.	#LCs	Freq.	GV(A/M)	GS(A/M)
actdec	65	28,9	80	20,53	40,77	23,08
androx	64	31,15	38	24,63	26,47	-40,63
bases	89	24,09	77	20,28	18,79	-13,48
cmd_stat	77	22,62	75	25,9	-12,66	-2,60
cnt16s	71	20,87	126	23,14	-9,81	77,46
dps	103	23,98	72	19,6	22,35	-30,10
eobpal	40	35,21	35	36,49	-3,51	-12,50
itiming	72	29,67	51	27,17	9,20	-29,17
len1	54	22,02	38	24,69	-10,81	-29,63
lorain	89	52,35	147	42,73	22,51	65,17
meanf	18	53,76	15	52,91	1,61	-16,67
rom_cs	11	63,69	15	53,47	19,11	36,36
rx1	25	52,79	21	33,67	56,79	-16,00
seqs	33	32,25	34	32,25	0,00	3,03
stat	76	25,77	134	23,86	8,01	76,32
tc3da	155	30,03	134	25,18	19,26	-13,55
tcadrctr	132	26,66	176	24,75	7,72	33,33
tx4dctp	36	25,9	47	27,32	-5,20	30,56
txdata2	47	31,44	36	27,93	12,57	-23,40
txf	39	33,55	43	28,08	19,48	10,26
u112	75	26,73	43	25,64	4,25	-42,67
wordctr	114	27,32	137	23,41	16,70	20,18
Gain Moyen %					11,98	4,79

Tableau 4.8 Comparaison des résultats d'ASYL+/Maxplus2 sur les conceptions réelles pour la famille Flex8000

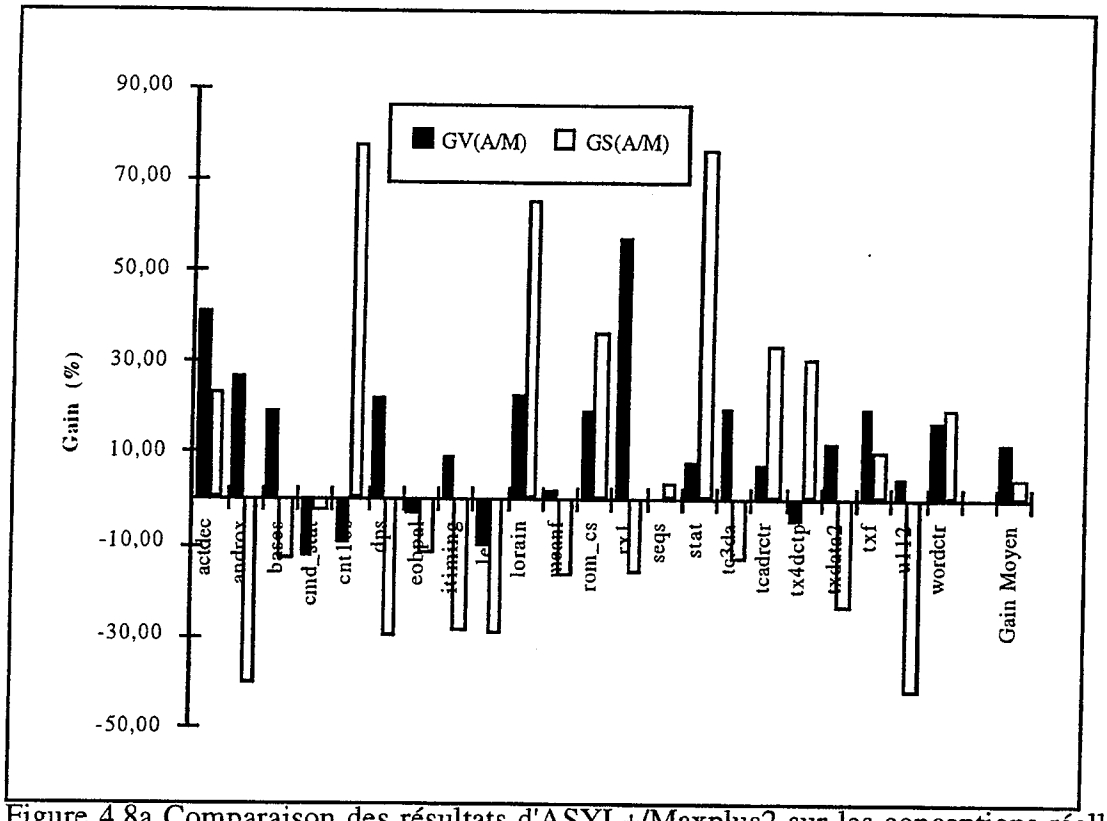


Figure 4.8a Comparaison des résultats d'ASYL+/Maxplus2 sur les conceptions réelles pour la famille Flex8000

	Asyl_Speed		Maxplus2		Exemplar		Gain en vitesse (%)		Gain en surface (%)	
Noms	#LCs	délai (ns)	#LCs	délai (ns)	#LCs	delay	GV(A/M)	GV(A/E)	GS (A/M)	GS(A/E)
5xp1	23	38,3	44	50	42	48,9	30,55	27,68	91,30	82,61
9sym	52	56,9	89	63,9	82	71,7	12,30	26,01	71,15	57,69
9symml	51	56,9	89	63,9	82	71,7	12,30	26,01	74,51	60,78
bw	62	35,2	57	36,3	66	51,4	3,12	46,02	-8,06	6,45
clip	55	45,9	75	65,9	61	63,3	43,57	37,91	36,36	10,91
count	61	50,4	104	44,9			-10,91		70,49	
duke2	232	81,1	228	84,8	263	84,4	4,56	4,07	-1,72	13,36
e64	354	51,9	642	57,9	128	103,5	11,56	99,42	81,36	-63,84
f51m	20	38,3	45	48,4	50	62,2	26,37	62,40	125,00	150,00
misex1	17	33,1	20	37,3	21	35,8	12,69	8,16	17,65	23,53
misex2	43	47,3	51	41,3	37	44,7	-12,68	-5,50	18,60	-13,95
o64	43	42,7	39	41,3	43	84	-3,28	96,72	-9,30	0,00
rd53	8	27,7	12	41,8	10	32,2	50,90	16,25	50,00	25,00
rd73	11	34,8	11	33,6	36	55,3	-3,45	58,91	0,00	227,27
rd84	44	46,8	93	68,7			46,79		111,36	
sao2	44	54	47	59,4	52	59	10,00	9,26	6,82	18,18
vg2	29	53,2	87	46,4	30	52,1	-12,78	-2,07	200,00	3,45
z4ml	10	34,7	13	34,9	6	33,3	0,58	-4,03	30,00	-40,00
Gain Moyen (%)							12,34	31,70	53,64	35,09

Tableau 4.9 Comparaison des résultats d'ASYL+/Exemplar/Maxplus2 sur le benchmark MCNC pour la famille Flex8000

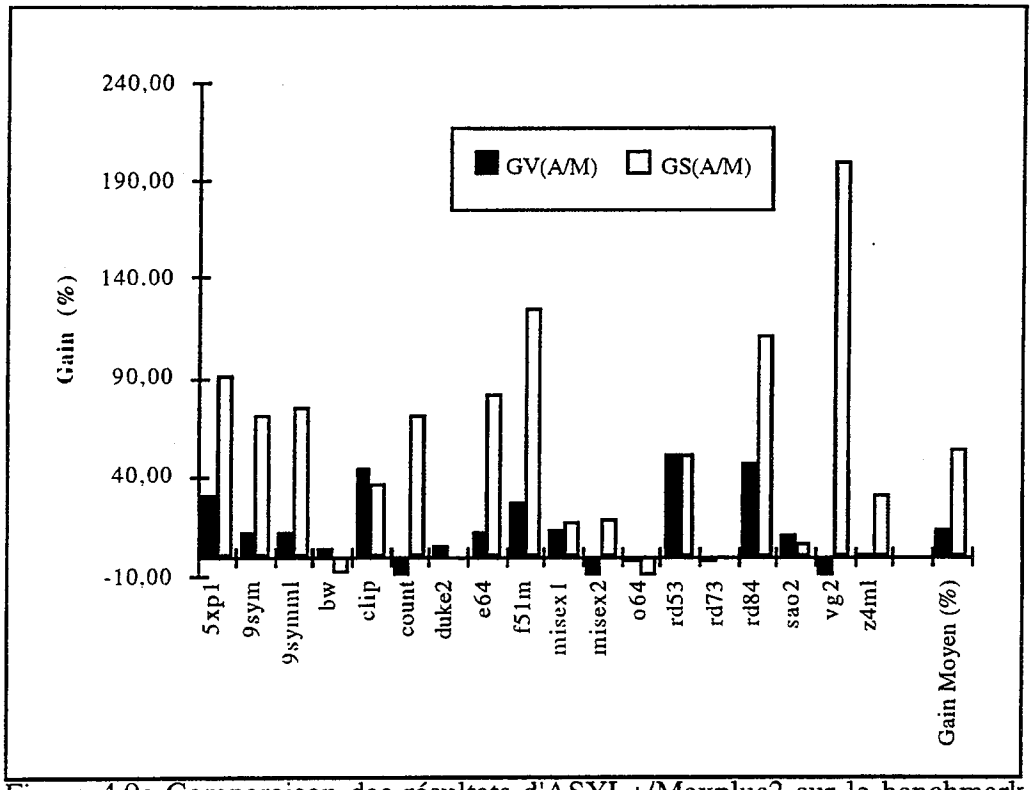


Figure 4.9a Comparaison des résultats d'ASYL+/Maxplus2 sur le benchmark MCNC pour la famille Flex8000

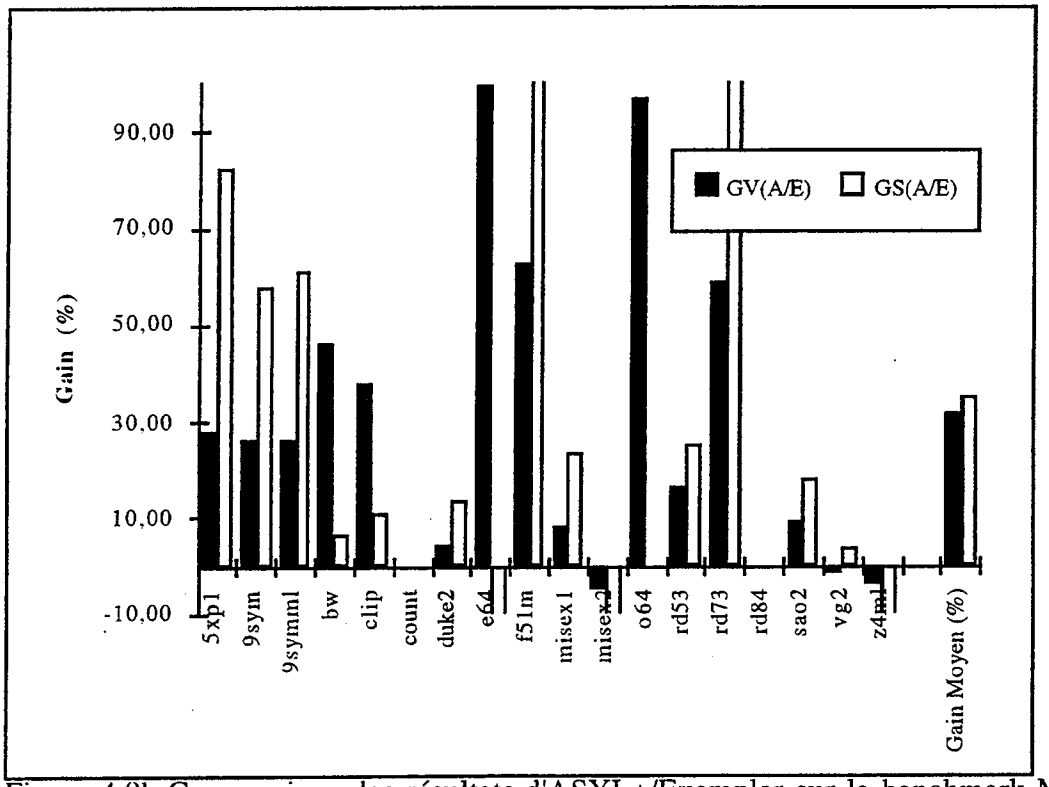


Figure 4.9b Comparaison des résultats d'ASYL+/Exemplar sur le benchmark MCNC pour la famille Flex8000

II.2 Pour la famille XC4000

Dans les tableaux des résultats 4.10, 4.11 et 4.12, les gains moyens en pourcentage en performance temporelle (resp. en surface) d'ASYL+ par rapport à Exemplar pour les 2 benchmarks ISCAS, MCNC sont respectivement de 6,4% et 25,13% (resp.17,24% et 104,24%).

En comparant à XACT, les gains moyens en pourcentage en performance temporelle (resp. en surface) d'ASYL+ pour les benchmarks ISCAS, MCNC et les conceptions réelles sont respectivement de 13,63%, 25,04% et 12,34% (resp. 1,55%, 13,13% et 1,59%).

Nous pouvons remarquer que le gain d'ASYL sur Exemplar et sur XACT pour la famille XC4000 est moins important que celui obtenu pour la famille Flex8000 pour les raisons que nous avons expliquées dans le paragraphe I.2 du chapitre 4.

Ceci révèle que l'optimisation temporelle pour les familles de FPGAs à base de LUT, dont l'architecture d'interconnexion est basée sur les interrupteurs, est très difficile à cause du délai imprédictible de l'interconnexion et de contraintes très fortes entre vitesse et surface. La prédiction temporelle pour ce type de FPGA reste encore un problème ouvert.

Les tableaux des résultats sont donnés ci-dessous:

	Asyl+		XACT		Exemplar		Gain en Vitesse (%)		Gain en Surface (%)	
Noms	CLBs	Fréq. (Mhz)	CLBs	Fréq. (Mhz)	CLBs	Fréq. (Mhz)	GV (A/X)	GV (A/E)	GS (A/X)	GS (A/E)
s1196	112	11,5	103	14	120	11,6	-17,86	-0,86	-8,04	7,14
s1238	135	12,1	118	12,4	121	10,3	-2,42	17,48	-12,59	-10,37
s1423	88	7,5	84	3,9	120	6,4	92,31	17,19	-4,55	36,36
s208	8	23,6	9	23,6	10	20,2	0,00	16,83	12,50	25,00
s27	2	35,6	3	35,4	2	33,1	0,56	7,55	50,00	0,00
s298	15	26,7	16	25,6	22	28,6	4,30	-6,64	6,67	46,67
s344	20	18,9	18	17,3	29	21,6	9,25	-12,50	-10,00	45,00
s349	20	21,3	19	17	29	22,8	25,29	-6,58	-5,00	45,00
s382	21	21,8	20	21	28	22,5	3,81	-3,11	-4,76	33,33
s953	90	14,6	78	13,5	85	14,8	8,15	-1,35	-13,33	-5,56
		Délai (ns)		Délai (ns)		Délai (ns)				
c1355	33	75,7	42	95,6	37	79,5	26,29	5,02	27,27	12,12
c1908	72	130	55	131	70	142,5	0,77	9,62	-23,61	-2,78
c3540	155	162,3	182	211,1			30,07		17,42	
c432	36	131,3	39	157,4	48	180,5	19,88	37,47	8,33	33,33
c499	36	93,3	41	98,7	41	103,6	5,79	11,04	13,89	13,89
c880	68	111,1	48	124,4	54	116,4	11,97	4,77	-29,41	-20,59
Gain Moyen (%)							13,63	6,39	1,55	17,24

Tableau 4.10 Comparaison des résultats d'ASYL+/Exemplar/XACT sur le benchmark ISCAS pour la famille XC4000

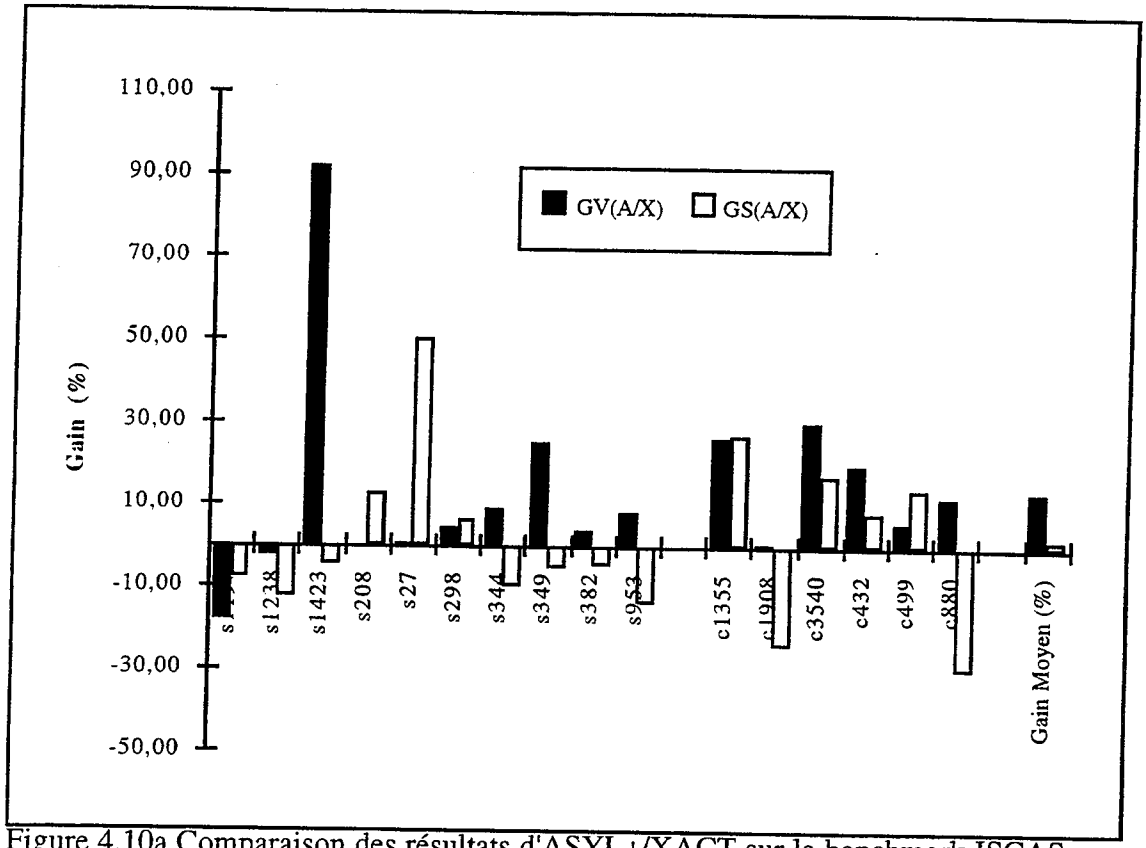


Figure 4.10a Comparaison des résultats d'ASYL+/XACT sur le benchmark ISCAS pour la famille XC4000

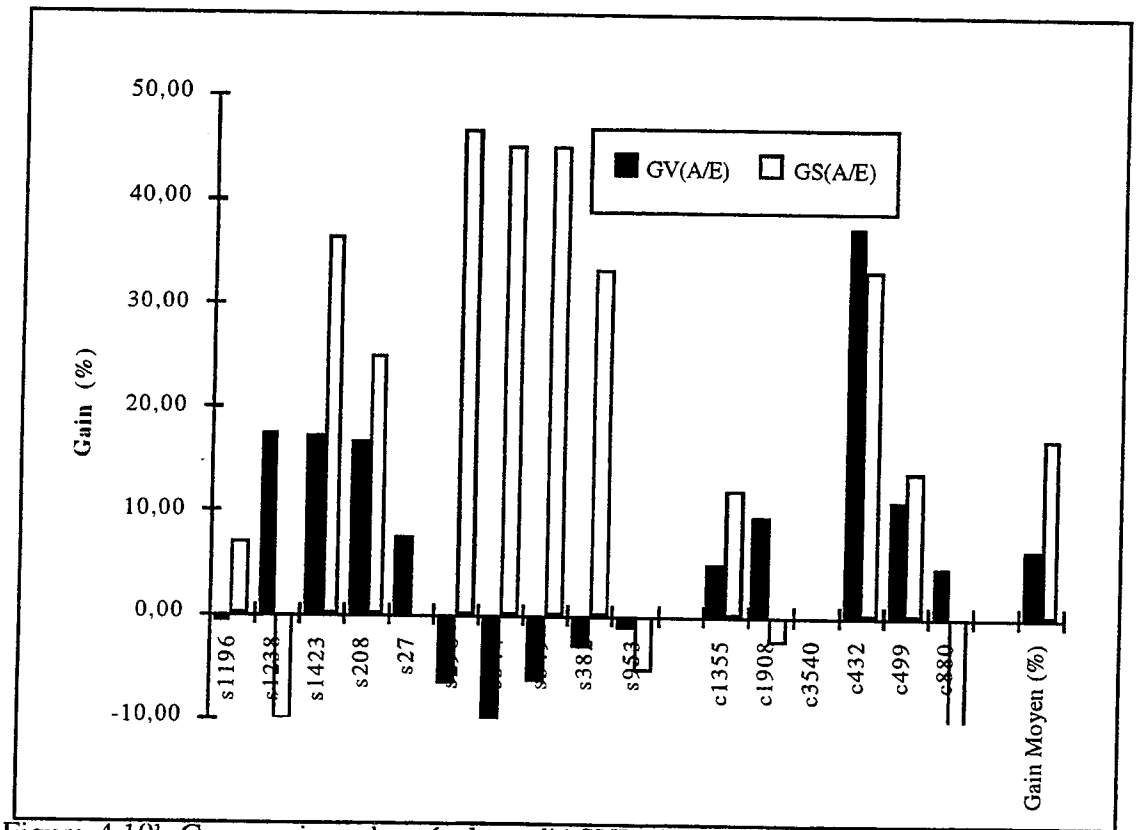


Figure 4.10b Comparaison des résultats d'ASYL+/Exemplar sur le benchmark ISCAS pour la famille XC4000

	Asyl+		XACT		Gain (%)	
Noms	CLBs	Fréq (Mhz)	CLBs	Fréq (Mhz)	GV (A/X)	GS (A/X)
actdec	42	26,2	20	22,8	14,91	-52,38
androx	18	29,3	16	26,6	10,15	-11,11
bases	27	21,8	30	19,5	11,79	11,11
cmd_stat	32	19,5	34	19,6	-0,51	6,25
cnt16s	26	33,6	29	28,6	17,48	11,54
dps	37	16,1	34	15,5	3,87	-8,11
eobpal	13	15,8	14	14,4	9,72	7,69
itiming	25	25,9	22	21,8	18,81	-12,00
len1	18	18,6	16	11	69,09	-11,11
lorain	34	24,3	36	28	-13,21	5,88
meanf	5	34,8	6	34,3	1,46	20,00
rx1	8	22,4	11	22,5	-0,44	37,50
seqs	15	26,3	16	25,5	3,14	6,67
stat	42	17,5	37	16,8	4,17	-11,90
tc3dma	59	21,9	55	20,4	7,35	-6,78
txdata2	21	27,5	15	22,6	21,68	-28,57
txmf	14	28,8	15	27,5	4,73	7,14
		Délai(ns)		Délai(ns)		
hexdec	3	29,6	7	35,8	20,95	133,33
horizont	43	54,4	40	72,8	33,82	-6,98
pmdec2	16	57,3	8	67,5	17,80	-50,00
ras0en	12	67,2	10	66	-1,79	-16,67
vertical	29	47,1	30	54,9	16,56	3,45
Gain Moyen (%)					12,34	1,59

Tableau 4.11 Comparaison des résultats d'ASYL+/Exemplar/XACT sur les conceptions réelles pour la famille XC4000

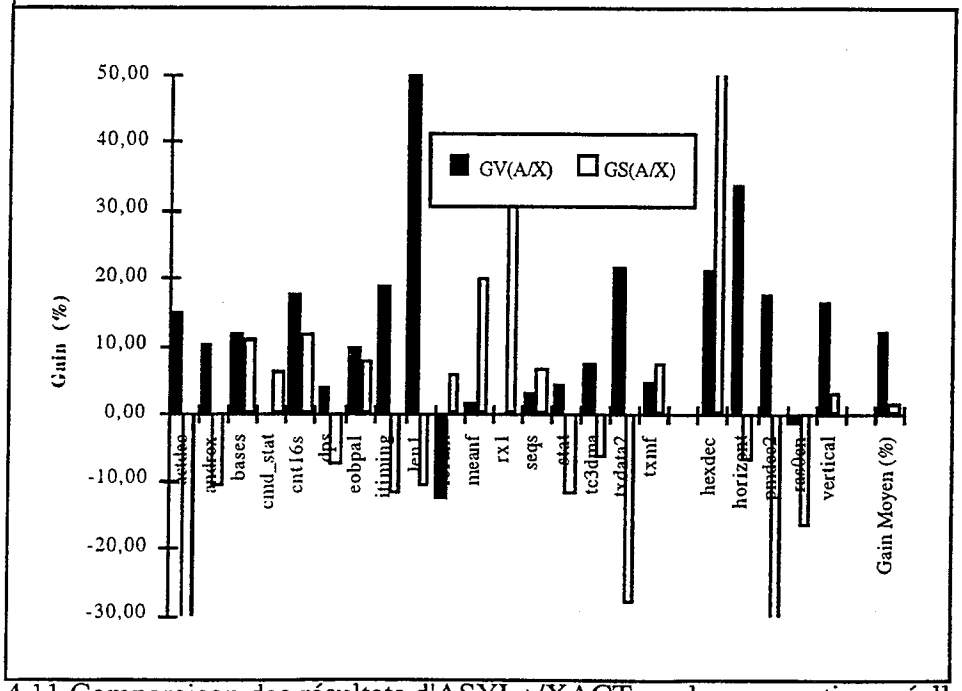


Figure 4.11 Comparaison des résultats d'ASYL+/XACT sur les conceptions réelles pour la famille XC4000

	Asyl+		XACT		Exemplar		Gain en vitesse (%)		Gain en surface (%)	
Noms	CLBs	Délai (ns)	CLBs	Délai (ns)	CLBs	Délai (ns)	GV (A/X)	GV (A/E)	GS (A/X)	GS (A/E)
5xp1	9	38,5	11	43,9	20	52,4	14,03	36,10	22,22	122,22
9sym	7	65,5	9	64,4	37	70,5	-1,68	7,63	28,57	428,57
9symml	7	65,5	9	64,4	37	70,5	-1,68	7,63	28,57	428,57
bw	24	43,9	25	42	29	54	-4,33	23,01	4,17	20,83
clip	17	56,6	20	57,1	28	67,2	0,88	18,73	17,65	64,71
count	21	59,4	20	73,3	22	68,9	23,40	15,99	-4,76	4,76
duke2	64	71,9	62	97,8	111	86,7	36,02	20,58	-3,13	73,44
e64	78	61,2	46	233,4			281,37		-41,03	
f51m	8	42,6	10	43,8	23	62,2	2,82	46,01	25,00	187,50
misex1	6	35,4	8	38,2	9	36,3	7,91	2,54	33,33	50,00
misex2	20	44,6	17	55,6	18	47,4	24,66	6,28	-15,00	-10,00
o64	21	50,9	22	51,7	22	109,3	1,57	114,73	4,76	4,76
rd53	2	28,8	3	32,5	5	37,3	12,85	29,51	50,00	150,00
rd73	6	41,5	8	44,8	16	52,6	7,95	26,75	33,33	166,67
rd84	9	50	11	53			6,00		22,22	
sao2	19	53,4	22	64,4	22	57,2	20,60	7,12	15,79	15,79
vg2	25	58	12	65,1	15	75,7	12,24	30,52	-52,00	-40,00
z4ml	3	35,9	5	38,1	3	39,1	6,13	8,91	66,67	0,00
Gain Moyen (%)							25,04	25,13	13,13	104,24

Tableau 4.12 Comparaison des résultats d'ASYL+/Exemplar/XACT sur le benchmark MCNC pour la famille XC4000

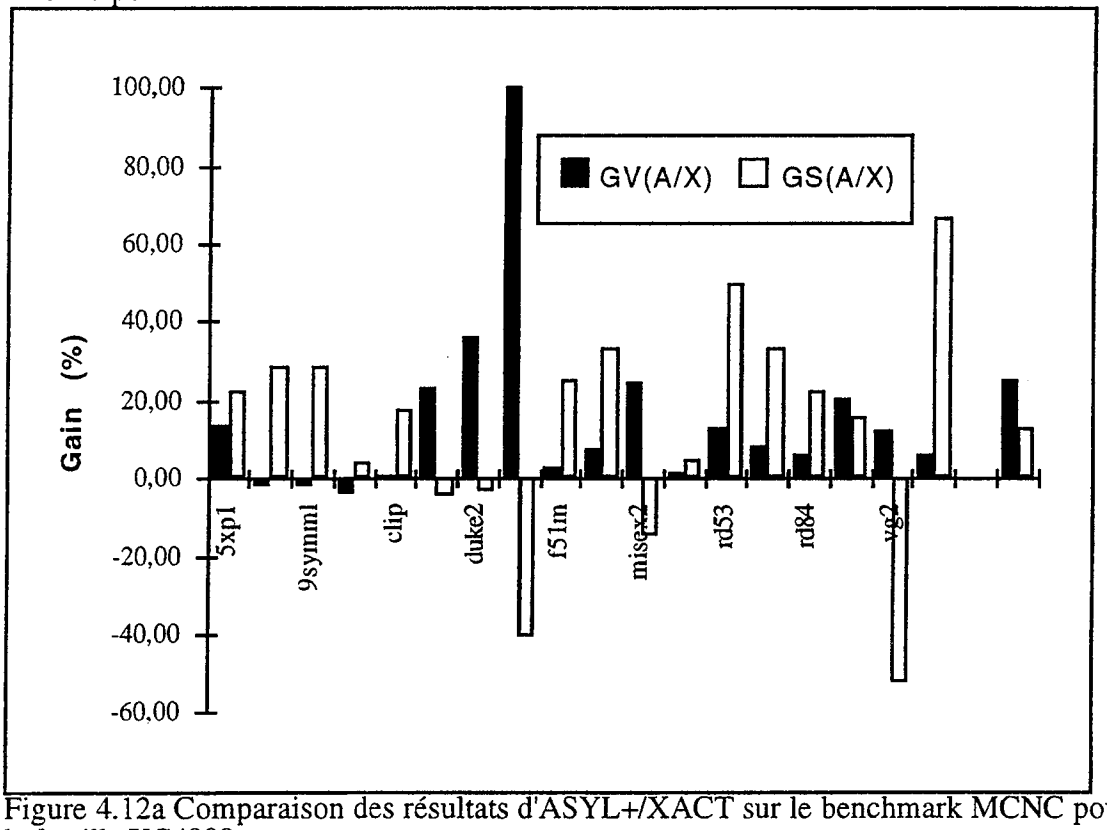


Figure 4.12a Comparaison des résultats d'ASYL+/XACT sur le benchmark MCNC pour la famille XC4000

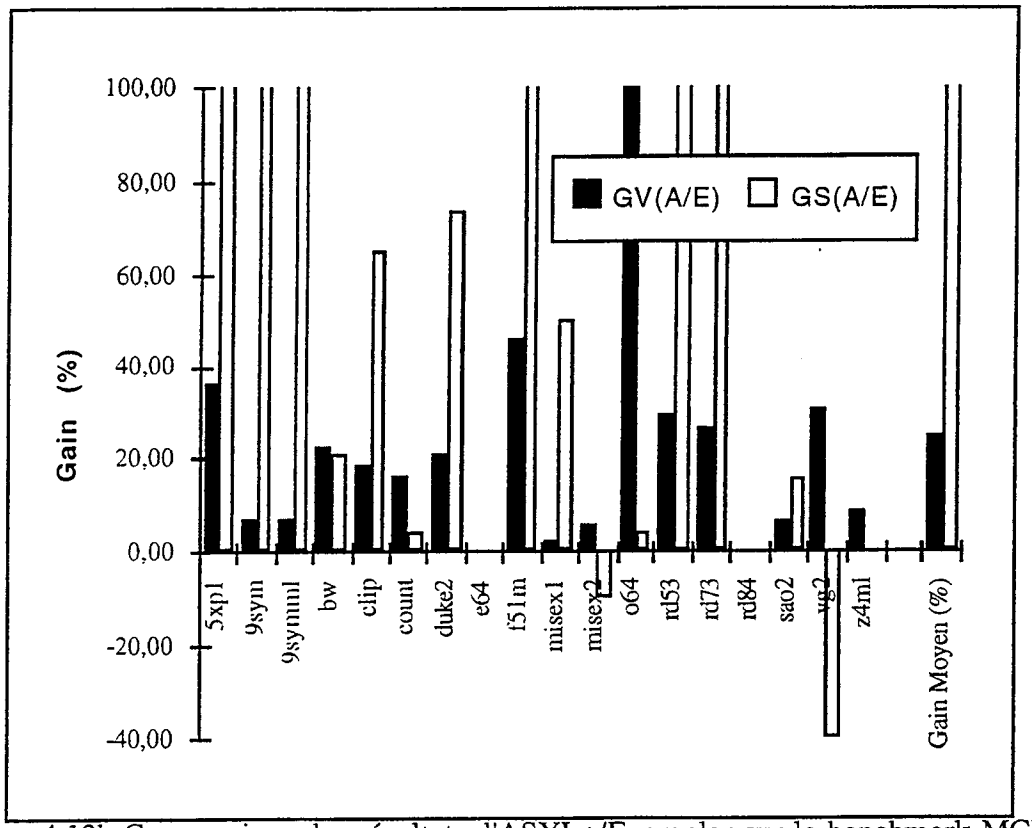


Figure 4.12b Comparaison des résultats d'ASYL+/Exemplar sur le benchmark MCNC pour la famille XC4000

CONCLUSION GENERALE

Cette thèse a abordé un problème extrêmement difficile, à savoir la prédiction de performances temporelles au cours de la synthèse avant la réalisation du circuit. Cette étude a plus précisement concerné les réseaux programmables à base de LUT (XC4000 de Xilinx et Flex8000 d'Altera) et ceux-ci illustrent bien les difficultés du problème.

De façon générale, la prédiction de performances temporelles est fondée sur des modèles de temps de traversée de cellules et de délai de connexion. Si le temps de traversée de cellule peut être raisonnablement connu ou approché, il en va autrement du délai d'interconnexion. En effet, les outils de placement/routage non seulement conduisent à des résultats non prévisibles mais de plus le temps de traversées des dispositifs de connexion est important par rapport au temps de traversée des portes. Dans cette thèse, nous avons proposé un modèle raisonnable du temps de traversée des cellules et un modèle très approximatif pour les connexions prenant en compte la sortance des cellules et en aucun cas la longueur des connexions ou, par exemple, les éléments de connexions (switch) traversées. On peut alors se poser deux questions :

- Le modèle proposé a-t-il été approché pour l'obtention de bons résultats en optimisation temporelle de circuits ?

- Le modèle proposé permet-it de prédire réellement les caractéristiques temporelles du circuit ?

Nous allons essayer de répondre à ces deux questions :

Pour la première question, il est claire que la thèse a clairement démontré que ce premier objectif était atteint. En effet, les résultats pratiques mesurés précisement après le placement et le routage montrent que le chemin critique a été amélioré de façon significative pendant la phase d'optimisation et que l'approche proposée est parfaitement compétitive avec les outils du commerce. Ceci signifie que malgré l'imprécision de la prédiction, l'outil détecte bien la zone critique à l'intérieur de la zone candidate à l'optimisation déclarée par les concepteurs.

Ce travail a aussi montré que les différentes fonctions de coût, même si elles sont relativement grossières, aident bien à redécomposer de façon intélligente le réseau en LUT à des fins d'optimisations temporelles. En résumé les modèles et les

méthodes proposés sont efficaces et ne font pas faire d'erreurs grossières de choix de zone ou de redécomposition technologique.

La deuxième question concerne la validité réelle de ces prédictions temporelles. Il s'agit d'un problème important car la dernière génération des outils de synthèse propose aux concepteurs une optimisation temporelle guidée par des valeurs à atteindre pour des chemins du circuit et non simplement l'indication d'une zone à optimiser. Sur ce dernier point, il faut reconnaître que les différences entre les valeurs réelles et les valeurs prédites sont importantes. Les seules techniques actuellement utilisées consistent à recueillir les contraintes de l'utilisateur, à en tenir compte au mieux pendant la phase de synthèse, puis à les communiquer au placeur/routeur. L'outil de synthèse dans ce cas prépare de son mieux et de façon très imprécise le réseau booléen, et l'outil de placement, au cours du déroulement des algorithmes de placement/routage, cherche à trouver des solutions correspondant au mieux aux objectifs. La prochaine génération d'outils de CAO verra sans doute la fusion des outils de synthèse et de placement/routage. L'évaluation temporelle se fera en associant à l'outil de synthèse un outil de "floorplan" et un tracé global de connexions. Dans ce cas, l'évaluation prédictive des délais de connexion sera fondée sur les longueurs obtenues par l'outil de préplacement/routage et les données de sortie seront d'une part le réseau booléen et d'autre part le "floorplan" du circuit.

Bien entendu, il reste toujours la solution de "back-annotation" et de méthodes itératives de synthèse.

De plus, nous pouvons citer "retiming" qui est une méthode très puissante pour améliorer la fréquence du circuit. C'est un problème ouvert, surtout quand les bascules ont des entrées "set/reset".

En attendant cette prochaine génération d'outils, nous pensons que cette thèse aura apporté une contribution au problème de l'évaluation prédictive des performances pour les réseaux à base de LUT aussi que des méthodes efficaces de décomposition.

REFERENCES BIBLIOGRAPHIQUES

[ABO92] Pierre Abouzeid - Thèse "Méthode de Factorisation Algébrique Dédiées aux Circuits Intégrés Complexes" - 18/12/1992 - INPG/CSI - Grenoble, France.

[ACT95] Actel - Data book - 1995.

[ALT93] Altera - AHDL - 1993.

[ALT95A] Altera - Data Book - 1995.

[AMD95] AMD - Data Book - 1995.

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

[BAB92] B. Babba, M. Crastes - "Automatic synthesis on the Table Lookup-based PGAs" - EUROASIC 92

[BAB95] B. Babba - Thèse "Synthèse optimisée sur les réseaux programmables de la famille Xilinx" - 20 juin 1995 - INPG/CSI

[BES92] Thierry Besson, H. Bouzouzou, M. Crastes, G. Saucier " Synthesis on Multiplexer-based Programmable Devices using (Ordered) Binary Decision Diagrams" - EDAC 1992 - pp. 8-13.

[BES93] T.Besson, H. Bouzouzou, Van Viet.LE, G.Saucier - "Use of Reduced Ordered Binary Dceision Diagrams for FPGA Mapping" - IFIP Workshop on Logic and Architecture Synthesis - pp. 133-144, December 6-7-8, 1993 - INPG, France

[BES94] T.Besson, H. Bouzouzou, Van Viet.LE, S.Tixier, G.Saucier - "Use of Binary Decision Diagrams for FPGA mapping" - FPGA'94 2nd International ACM/SIGDA Workshop on Field Programmable Gate Arrays

[BHA92] N.B.Bhat, D.D.Hill - "Routable Technology Mapping for LookUp Table-Based FPGAs" - EDAC 1992 - pp.95-98

[BOU91] Laurent Bouchet - Thèse "Synthèse sur réseaux programmables" - 1991 - INPG/CSI - Grenoble, France.

[BRA87] R. Brayton, R. Rudell, A. Wang,A. Sangiovanni-Vincentelli - "MIS: a multiple-level logic optmization system" - IEEE Transactions on CAD, p. 1062-1081, November 1987.

[BRY86] R. Bryan - "Graph-based algorithms for Boolean function manipulation" - IEEE transactions on Computers, Vol C35, n°8, Août 1986, pp. 677-691.

[CAL91] Calazans N., Jacobi R., Zhang Q., Trullemans C. - "Improving Binary Decision Diagram Through Incremental Reduction and Improved Heuristics" - Custom Integrated Circuits Conference. 1991

[CHA89] Pak K.Chan. Kevin Karplus - " Computing signal delay in general RC network by Tree/Link Partitioning" - ACM/IEEE - 26th DAC 1989, Las Vegas Convention Center

[CHAN93] P.K.Chan. M.Schlag, J.Y.Zien - "On Routability Prediction for FPGA's" - 30th ACM / IEEE Design Automation Conference 1993 - pp.326-330

[CHEN93] C.S.Chen. Y.W.Tsay, T.T.Hwang, A.C.H.Wu, Y.L.Lin - "Combining Technology Mapping and Placement for Delay Optimization in FPGA Design" - ICCAD 1993 - pp.123-127

[CONG92A] Jason Cong, Y.Ding, A.B. Kahng, P. Trajmar - "DAG-Map: Graph-Based FPGA Technology Mapping for Delay Optimization" - Design&Test of Computers September 1992 - pp.7-20

[CONG92B] Jason Cong, Y.Ding - "An Optimal Technology Mapping Algorithm for Delay Optimization in LookUp-Table-Based FPGA Designs" - ICCAD 1992 - pp.48-53

[CONG93A] Jason Cong, Y.Ding - "Beyond the Combinatorial Limit in Depth Minimization for LookUp-Table-Based FPGA Designs" - ICCAD 1993 - pp.110-114

[CONG93B] J. Cong, Y. Ding - "On ARea/Depth Trade-Off in LUT-Based FPGA Technology Mapping" -30th ACM / Design Automation Conference 1993 - pp. 213-218

[CONG94A] Jason Cong, Y.Ding - "FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in LookUp-Table-Based FPGA Designs" - IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, Vol 13, No. 1, Jan 1994 - pp.1-12

[CONG95A] J. Cong. Y.Ding - "On Nominal Delay Minimization in LUT-Based FPGA Technology Mapping" - FPGA'95, ACM/SIGDA International Symposium on FPGAs- pp 82-88.

[CONG95B] J. Cong. Y. Y. Hwang - "Simultaneous Depth and Area Minimization in LUT-Based FPGA Mapping" - FPGA'95, ACM/SIGDA International Symposium on FPGAs- pp 68-74

[CRA89] M.Crastes, C.Duff, G.Saucier - "ASYL: a Logic and Architecture Design Automation System" - EURO ASIC 1989, pp. 183-209, jan 1989.

[CRA92] M. Crastes, B.Babba, G. Saucier - "Input Driven Synthesis on PLDs and PGAs" - EDAC92, pp. 48-52, March 1992.

[DET87] E. Detjens, G. Gannot, R. Rudell, A. Sangovani-Vincentelli and A. Wand - "Technology Mapping in MIS" - ICCAD 87, pp. 116-119, November 1987.

[FIL91] D.Filo, J.C.Y.Yang, F.Mailhot, G.De Micheli - "Technology Mapping for a Two-Output RAM-Based FPGAs" - EDAC 1991 - pp.534-538

[FRA90] R.J.Francis, J.Rose, K. Chung - "Chortle: A Technology Mapping Program for Look-Up Table-Based Field Programmable Gate Arrays" - 27th ACM / IEEE Design Automation Conference 1990 - pp.613-619

[FRA91A] R.J.Francis, J.Rose, Z. Vranesic - "Chortle-crf: Fast Technology Mapping for Look-Up Table-Based Field Programmable Gate Arrays" - 28th ACM / IEEE Design Automation Conference 1991 - pp.227-233

[FRA91B] R.J.Francis, J.Rose, Z. Vranesic - "Technology Mapping of Look-Up Table-Based Field Programmable Gate Arrays for Performance" - ICCAD 1991 - pp.568-571

[FRI87]. Friedman S., Supowit K. - "Finding the Optimal Variable Ordering for Binary Decision Diagram" 24th DAC, IEEE/ACM, 1987, p. 348.

[ISC89] ISCAS International Benchmarks - 1989.

[KAP94] B.Kapoor - "An Efficient Graph-Based Technology Mapping Algorithm for FPGAs Using Lookup Tables" - FPGA'94 2nd International ACM/SIGDA Workshop on Field Programmable Gate Arrays

[KAR91] Kevin Karplus - "Xmap: a Technology Mapper for Table-lookup FPGAs" - 28th ACM / IEEE Design Automation Conference 1991 - pp.240-243

[KUN68] J. Kuntzmann - "Algèbre de Boole" - Edition Dunod, Paris, 1968.

[LE95A] Van Viet LE, Gabrièle Saucier - "A New Performance Optimization Algorithm for LUT-based FPGAs" - SASIMI'95, p176-183.

[LE95B] Van Viet LE, Thierry Besson, A. Abbara, D.Barsen, H.Bogushevitsh, G.Saucier, M.Crastes - "ASIC Prototyping with Area Oriented Mapping For ALTERA/Flex Devices" - SASIMI'95, p176-183.

[LE95C] Van Viet.LE, G.Saucier - "A Performance Driven Look-up Table Mapping" - IFIP Workshop on Logic and Architecture Synthesis - pp. 86-93, December 18-19, 1995 - INPG, France

[LE96A] Van Viet LE, Gabrièle Saucier, M. Crastes "Performance Optimization for LUT-based FPGAs" - Will be published in ED&TC96 - March 96 - Paris, France.

[LE96B] Van Viet LE, Gabrièle Saucier, M. Crastes "Performance Optimization under user's constraints for LUT-based FPGAs" - Will be published in FPGA'96 ACM/SIGDA - Monterey, California - February 11-13, 1996.

[LE96C] Van Viet LE, H. Bogushevitsh, A. Abbara, D.Brasen, G.Saucier - "ASIC Prototyping for Altera FPGA Devices on ICUBE Programmable Interconnect" - Will be published in ED&TC 96 - March 96 - Paris, France.

[LIN84] Tzu Mu Lin, Carver A.Mead "Signal delay in general RC network" - IEEE Trans on CAD, Oct 1984

[MAT94] A.Mathur, C.L.Liu - "Performance Driven Technology Mapping for Lookup table Based FPGA using the General Delay Model" - FPGA'94 2nd International ACM/SIGDA Workshop on Field Programmable Gate Arrays

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

[MUR90] R. Murgai, Y. Nishizaki, N. Shenoy, R.K. Brayton, A.S. Vincentelli - "Logic Synthesis for Programmable Gate Arrays" - 27th ACM/IEEE Design Automation Conference 1990 - pp.620-625

[MUR91A] R. Murgai, N. Shenoy, R.K. Brayton, A.S. Vincentelli - "Performance Directed Synthesis for Table Look Up Programmable Gate Arrays" - ICCAD 1991 - pp.572-575

[MUR91B] R. Murgai, N. Shenoy, R.K. Brayton, A.S. Vincentelli - "Improved Logic Synthesis Algorithms for Table Look Up Architectures" - ICCAD 1991 - pp.564-567

[MUR92] R. Murgai, R.K. Brayton, A.S. Vincentelli - "Sequential Synthesis for Table Look Up PGA's" - EUROASIC 1992 - pp.32-37

[MUR93A] R. Murgai, R.K. Brayton, A.S. Vincentelli - "Sequential Synthesis for Table Look Up Programmable Gate Arrays" - 30th ACM/IEEE Design Automation Conference 1993 - pp.224-229

[MUR93B] R. Murgai, R.K. Brayton, A.S. Vincentelli - "Cube-packing and Two-level Minimization" - ICCAD 1993 - pp.115-122

[MUR94] R. Murgai, R.K. Brayton, A.S. Vincentelli - "Optimum Functional Decomposition Using Encoding" - 31th ACM/IEEE Design Automation Conference 1994 - pp.408-414

[ORC95] ORCA - AT&T Data Book - 1995

[POI90] F.Poirot, P.Abouzeid, G.Saucier - " Lexicographical factorization minimizing the critical path and the routing factor of multilevel logic" - IFIP Working Conference on Logic and Architecture Synthesis, PAris, France, June 1990.

[QUI95] QuickLogic - Data Book - 1995

[RUB83] JorgeRubinstein, Paul Penfield, Mark A.Horowitz - " Signal delay in RC Tree Network" - IEEE Trans On CAD, July 1983

[SAK93] Khalid Sakouti - Thèse "Synthèse et Optimisation de Réseaux Booléens" - 26/11/1993 - INPG/CSI - Grenoble, France.

[SAU90] G.Saucier, P. Abouzeid et al - "Multilevel Synthesis Minimizing the routing factor" 27th DAC, pp 365-368, june 1990

[SAU92A] G.Saucier - "Analysis of the Trends in Logic Synthesis" - SASIMI'92, Kobe, Japan, April 1992.

[SAU92B] G.Saucier, P.Abouzeid, T.Besson, J.Fron, K.Sakouti - " A coherent Area/Timing Optimization Strategies for Multilevel Logic" - TAU'92, Workshop on Timing Issues in the Specification and Synthesis, Princeton University, USA, March 1992.

[SAW92] P. Sawkar, D. Thomas - "Area and Delay Mapping for Table-Look-Up Based Field Programmable Gate Arrays" - 29th ACM / IEEE Design Automation Conference 1992 - pp.368-373

[SAW93] P. Sawkar, D. Thomas - "Performance Directed Technology Mapping for Table-Look-Up Based Field Programmable Gate Arrays" - 30th ACM / IEEE Design Automation Conference 1993 - pp.208-212

[SCH94] E.Schubert, U.Kebschull, W.Rosenstiel - "Functional Decision Diagrams for Technology Mapping to LookUp Table FPGA" - FPGA'94 2nd International ACM/SIGDA Workshop on Field Programmable Gate Arrays - pp.1-10

[SCHL94] M.Schlag, J.Kong, P.K.Chan - "Routability-Driven Technology Mapping for Lookup Table-Based FPGA's" - IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, Vol 13, No. 1, Jan 1994 - pp.13-26

[SIN90] K. J. Singh and A. Sangovani-Vincentelli - "A Heuristic Algorithm for the Fanout Problem" - DAC. pp. 357-360, June 1990.

[TIW93] V.Tiwari, P. Ashar, S.Malik- "Technology Mapping for Low Power" - 30th ACM / Design Automation Conference 1993 - pp.74-79

[TOG94] N.Togawa, M.Sato, T.Ohtsuki - "Maple: A Simultaneous Technology Mapping, Placement, and Global Routing Algorithm for FPGAs" - ICCAD 1994 - pp.156-163

[TOU91A] H. J. Touati, Hamid Savoj, R. K. Brayton - "Delay Optimization of combinational logic circuits by clustering and partial collapsing" - ICCAD 1991 -pp. 188-191.

[TOU91] H. J. Touati - Doctoral Thesis - "Performance-Oriented Technology Mapping" Berkeley University - 1991

[TRI92] S. Trimberger, M.R.Chene - "Placement-Based Partitioning for LookUp Table-Based FPGAs" - EDAC 1992 - pp.91-94

[TSUI93] C.Y.Tsui, M.Pedram, A.M.Despain- "Technology Decomposition and Mapping Targeting Low Power Dissipation" - 30th ACM / Design Automation Conference 1993 - pp.74-79

[WAL90] - "High Level Delay Estimation for Technology-Independent Logic Equations" - ICCAD, pp. 188-191, 1990.

[WAN92] Wei Wan, M.A. Perkowski - "A New Approach to the Decomposition of Incompletely Specified Multi-Output Functions Based on Graph Coloring and Local Transformations and its Application to FPGA Mapping" - EURO-DAC 1992 - pp.230-234

[WEI94] U. Weinmann, W. Rosenstiel - "Logic Module Independent Mapping for LookUp Table FPGA" - FPGA'94 2nd International ACM/SIGDA Workshop on Field Programmable Gate Arrays

[WOO91] Nam Sung Woo - "A Heuristic Method for FPGA Technology Mapping Based on the Edge Visibility" - 28th ACM / IEEE Design Automation Conference 1991 - pp.248-251

[XIL94] XILINX - The Programmable Logic Data Book - 1994

[YANG94] H.Yang, D.F.Wong - "Edge-Map: Optimal Performance Driven Technology Mapping for Iterative LUT Based FPGA Designs" - ICCAD 1994 - pp.150-155

[YOS91] K. Yoshikawa et al. - "Timing Optimization on mapped circuits" - 28th DAC, pp. 112-117, Juin 1991.

ANNEXES

EPF8282 Internal Timing Parameters *Note (1)*

EPF8282 I/O Element Timing Parameters											
Parameter	A-2 Speed Grade		A-3 Speed Grade		A-4 Speed Grade		-2 Speed Grade		-3 Speed Grade		
	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	Unit
t_{IOD}		0.7		0.8		0.9		1.0		2.0	ns
t_{IOC}		1.7		1.8		1.9		1.0		2.0	ns
t_{IOE}		1.7		1.8		1.9		1.0		2.0	ns
t_{IOCO}		1.0		1.0		1.0		1.0		1.0	ns
t_{IOCOMB}		0.3		0.2		0.1		0.0		0.0	ns
t_{IOSU}	1.4		1.6		1.8		2.0		2.0		ns
t_{IOH}	0.0		0.0		0.0		0.0		0.0		ns
t_{IOCLR}		1.2		1.2		1.2		1.3		1.3	ns
t_{IN}		1.5		1.6		1.7		1.8		2.8	ns
t_{OD1}		1.1		1.4		1.7		2.5		3.0	ns
t_{OD2}		1.6		1.9		2.2		–		–	ns
t_{OD3}		4.6		4.9		5.2		6.5		7.0	ns
t_{XZ}		1.4		1.6		1.8		2.5		3.0	ns
t_{ZX1}		1.4		1.6		1.8		2.5		3.0	ns
t_{ZX2}		1.9		2.1		2.3		–		–	ns
t_{ZX3}		4.9		5.1		5.3		6.5		7.0	ns

EPF8282 Interconnect Timing Parameters											
Parameter	A-2 Speed Grade		A-3 Speed Grade		A-4 Speed Grade		-2 Speed Grade		-3 Speed Grade		
	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	Unit
$t_{LABCASC}$		0.3		0.3		0.4		0.5		0.9	ns
$t_{LABCARRY}$		0.3		0.3		0.4		0.5		0.6	ns
t_{LOCAL}		0.5		0.6		0.8		1.0		1.0	ns
t_{ROW}		4.2		4.2		4.2		4.2		4.2	ns
t_{COL}		2.5		2.5		2.5		2.5		2.5	ns
t_{DIN_C}		5.0		5.0		5.5		6.0		7.0	ns
t_{DIN_D}		7.2		7.2		7.2		7.2		8.2	ns
t_{DIN_IO}		5.0		5.0		5.5		7.0		8.0	ns

EPF8282 Logic Element Timing Parameters

Parameter	A-2 Speed Grade		A-3 Speed Grade		A-4 Speed Grade		-2 Speed Grade		-3 Speed Grade		
	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	Unit
t_{LUT}		2.0		2.5		3.2		4.1		5.1	ns
t_{CLUT}		0.0		0.0		0.0		0.2		1.0	ns
t_{RLUT}		0.9		1.1		1.5		1.9		3.4	ns
t_{GATE}		0.0		0.0		0.0		0.0		0.0	ns
t_{CASC}		0.6		0.7		0.9		1.1		2.0	ns
t_{CICO}		0.4		0.5		0.6		0.7		1.1	ns
t_{CGEN}		0.4		0.5		0.7		0.7		1.4	ns
t_{CGENR}		0.9		1.1		1.5		1.7		2.4	ns
t_{C}		1.6		2.0		2.5		2.8		3.1	ns
t_{CH}	1.7		1.7		2.7		3.5		4.3		ns
t_{CL}	1.7		1.7		2.7		3.5		4.3		ns
t_{CO}		0.4		0.5		0.6		0.4		0.9	ns
t_{COMB}		0.4		0.5		0.6		0.4		0.9	ns
t_{SU}	0.8		1.1		1.2		1.5		1.7		ns
t_{H}	0.9		1.1		1.5		2.3		4.0		ns
t_{PRE}		0.6		0.7		0.8		0.7		1.2	ns
t_{CLR}		0.6		0.7		0.8		0.7		1.2	ns

EPF8282 External Reference Timing Characteristics *Note (1)*

Parameter	A-2 Speed Grade		A-3 Speed Grade		A-4 Speed Grade		-2 Speed Grade		-3 Speed Grade		
	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	Unit
t_{DRR}		15.8		19.8		24.8		29.3		35.5	ns

Note:
(1) Internal and external timing parameters for EPF8282A devices are preliminary.

IOB Switching Characteristic Guidelines

Testing of the switching parameters is modeled after testing methods specified by MIL-M-38510/605. All devices are 100% functionally tested. Since many internal timing parameters cannot be measured directly, they are derived from benchmark timing patterns. The following guidelines reflect worst-case values over the recommended operating conditions. For more detailed, more precise, and more up-to-date timing information, use the values provided by the XACT timing calculator and used in the simulator.

	Speed Grade	-6		-5		-4		
Description	**Symbol**	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**	**Units**
Input								
Propagation Delays								
Pad to I1, I2	T_{PID}		4.0		3.0		2.8	ns
Pad to I1, I2, via transparent latch (fast)	T_{PLI}		8.0		7.0		6.0	ns
Pad to I1, I2, via transparent latch (with delay)	T_{PDLI}		26.0		24.0		**	ns
Clock (IK) toI1, I2, (flip-flop)	T_{IKRI}		8.0		7.0		6.0	ns
Clock (IK) to I1, I2 (latch enable, active Low)	T_{IKLI}		8.0		7.0		6.0	ns
Set-up Time (Note 3)								
Pad to Clock (IK), fast	T_{PICK}	7.0		6.0		4.0		ns
Pad to Clock (IK) with delay	T_{PICKD}	25.0		24.0		**		ns
Hold Time (Note 3)								
Pad to Clock (IK), fast	T_{IKPI}	1.0		1.0		1.0		ns
Pad to Clock (IK) with delay	T_{IKPID}	neg		neg		neg		ns
Output								
Propagation Delays								
Clock (OK) to Pad (fast)	T_{OKPOF}		7.5		7.0		6.5	ns
same (slew rate limited)	T_{OKPOS}		11.5		10.0		9.5	ns
Output (O) to Pad (fast)	T_{OPF}		9.0		7.0		5.5	ns
same (slew-rate limited)	T_{OPS}		13.0		10.0		8.5	ns
3-state to Pad begin hi-Z (slew-rate independent)	T_{TSHZ}		9.0		7.0		6.5	ns
3-state to Pad active and valid (fast)	T_{TSONF}		13.0		10.0		9.5	ns
same (slew -rate limited)	T_{TSONS}		17.0		13.0		12.5	ns
Set-up and Hold Times								
Output (O) to clock (OK) set-up time	T_{OOK}	8.0		6.0		5.5		ns
Output (O) to clock (OK) hold time	T_{OKO}	0		0		0		ns
Clock								
Clock High or Low time	T_{CH}/T_{CL}	5.0		4.0		4.0		ns
Global Set/Reset								
Delay from GSR net through Q to I1, I2	T_{RRI}		14.5		13.5		13.5	ns
Delay from GSR net to Pad	T_{RPO}		18.0		17.0		14.0	ns
GSR width*	T_{MRW}	21.0		18.0		18.0		ns

* Timing is based on the XC4005. For other devices see XACT timing calculator.
** See preceding page

Notes:
1. Timing is measured at pin threshold, with 50 pF external capacitive loads (incl. test fixture). **Slew rate limited** output rise/fall times are approximately two times longer than fast output rise/fall times. **A maximum total external capacitive load for simultaneous fast mode switching in the same direction is 200 pF per power/ground pin pair. For slew-rate limited outputs this total is two times larger. Exceeding this maximum capacitive load can result in ground bounce of >1.5 V amplitude, <5 ns duration, which might cause problems when the LCA drives clocks and other asynchronous signals.**
2. Voltage levels of unused (bonded and unbonded) pads must be valid logic levels. Each can be configured with the internal pull-up or pull-down resistor or alternatively configured as a driven output or be driven from an external source.
3. Input pad setup times and hold times are specified with respect to the internal clock (IK). To calculate system setup time, subtract clock delay (clock pad to IK) from the specified input pad setup time value, but do not subtract below zero. Negative hold time means that the delay in the input data is adequate for the **external system hold time** to be zero, provided the input clock uses the Global signal distribution from pad to IK.

CLB Switching Characteristic Guidelines

Testing of the switching parameters is modeled after testing methods specified by MIL-M-38510/605. All devices are 100% functionally tested. Since many internal timing parameters cannot be measured directly, they are derived from benchmark timing patterns. The following guidelines reflect worst-case values over the recommended operating conditions. For more detailed, more precise, and more up-to-date timing information, use the values provided by the XACT timing calculator and used in the simulator.

Description	Symbol	-6 Min	-6 Max	-5 Min	-5 Max	-4 Min	-4 Max	Units
Combinatorial Delays								
F/G inputs to X/Y outputs	T_{ILO}		6.0		4.5		4.0	ns
F/G inputs via H' to X/Y outputs	T_{IHO}		8.0		7.0		6.0	ns
C inputs via H' to X/Y outputs	T_{HHO}		7.0		5.0		4.5	ns
CLB Fast Carry Logic								
Operand inputs (F1,F2,G1,G4) to COUT	T_{OPCY}		7.0		5.5		5.0	ns
Add/Subtract input (F3) to COUT	T_{ASCY}		8.0		6.0		5.5	ns
Initialization inputs (F1,F3) to COUT	T_{INCY}		6.0		4.0		3.5	ns
C_{IN} through function generators to X/Y outputs	T_{SUM}		8.0		6.0		5.5	ns
C_{IN} to C_{OUT}, bypass function generators.	T_{BYP}		2.0		1.5		1.5	ns
Sequential Delays								
Clock K to outputs Q	T_{CKO}		5.0		3.0		3.0	ns
Set-up Time before Clock K								
F/G inputs	T_{ICK}	6.0		4.5		4.5		ns
F/G inputs via H'	T_{IHCK}	8.0		6.0		6.0		ns
C inputs via H1	T_{HHCK}	7.0		5.0		5.0		ns
C inputs via DIN	T_{DICK}	4.0		3.0		3.0		ns
C inputs via EC	T_{ECCK}	7.0		4.0		3.0		ns
C inputs via S/R, going Low (inactive)	T_{RCK}	6.0		4.5		4.0		ns
C_{IN} input via F'/G'	T_{CCK}	8.0		6.0		5.5		ns
C_{IN} input via F'/G' and H'	T_{CHCK}	10.0		7.5		7.3		ns
Hold Time after Clock K								
F/G inputs	T_{CKI}	0		0		0		ns
F/G inputs via H'	T_{CKIH}	0		0		0		ns
C inputs via H1	T_{CKHH}	0		0		0		ns
C inputs via DIN	T_{CKDI}	0		0		0		ns
C inputs via EC	T_{CKEC}	0		0		0		ns
C inputs via S/R, going Low (inactive)	T_{CKR}	0		0		0		ns
Clock								
Clock High time	T_{CH}	5.0		4.5		4.5		ns
Clock Low time	T_{CL}	5.0		4.5		4.5		ns
Set/Reset Direct								
Width (High)	T_{RPW}	5.0		4.0		4.0		ns
Delay from C inputs via S/R, going High to Q	T_{RIO}		9.0		8.0		7.0	ns
Master Set/Reset*								
Width (High or Low)	T_{MRW}	21.0		18.0		18.0		ns
Delay from Global Set/Reset net to Q	T_{MRQ}		33.0		31.0		28.0	ns

* Timing is based on the XC4005. For other devices see XACT timing calculator.