THESE DE DOCTORAT

de

L'UNIVERSITE Pierre et Marie CURIE

Specialité:

Informatique

Présentée par:

Anthony Lester

Pour obtenir le grade de:

DOCTEUR de l'UNIVERSITE Pierre et Marie CURIE

Sujet de la thèse:

Abstraction Fonctionnelle des Circuits Numériques VLSI Avec une méthode formelle basée sur une extraction de réseau de portes

Soutenue le:

21 Décembre 1999

Devant le Jury composé de:

Mr. Pirouz Bazargan-Sabet	**Directeur de Thèse**
Mr. Alain Greiner	**Examinateur**
Mr. Alain Guyot	**Rapporteur**
Mr. Michel Minoux	**Examinateur**
Mr. Christian Piguet	**Rapporteur**
Mr. Sanjay Rajopadhye	**Examinateur**

UNIVERSITE PIERRE ET MARIE CURIE – 4, PLACE JUSSIEU – 75252 PARIS CEDEX 05

Résumé

Cette thèse s'inscrit dans le cadre de la vérification des circuits numériques de haute complexité. Elle porte sur le désassemblage et sur l'abstraction fonctionnelle. Le problème abordé concerne l'identification des portes et la génération automatique d'une description au niveau RTL à partir d'un réseau de transistors.

L'obtention de ce réseau de portes facilite, par la suite, les vérifications temporelles et l'analyse de la consommation. Surtout, la description RTL permet une vérification fonctionnelle d'un circuit VLSI quelle que soit la méthode de conception. Elle permet aussi de réaliser la migration technologique par le biais d'une resynthèse de cette description comportementale vers une nouvelle technologie cible.

Une méthode formelle (sans recourir à une bibliothèque de cellules prédéfinies) est proposée qui exploite les corrélations au sein d'un circuit pour extraire uniquement la fonctionnalité utile. Cela permet d'obtenir un réseau de portes avec une caractérisation fonctionnelle optimale.

Le problème de la reconnaissance et la modélisation des éléments séquentiels est adressé par l'élaboration d'un algorithme original. Cet algorithme s'appuie sur la dérivée partielle booléenne pour analyser la stabilité et la capacité de mémorisation des boucles. La méthode permet de générer automatiquement un modèle comportemental pour tous les latchs statiques dans un circuit.

Dans une dernière partie, la faisabilité d'une approche hybride est démontrée. Avec cette approche, une méthode par reconnaissance de formes hiérarchique vient compléter la méthode formelle. Cette approche apporte deux avantages principaux : premièrement il devient possible de traiter en entier des circuits contenant des parties analogiques, deuxièmement cette méthode permet un traitement plus efficace et une modélisation plus optimale des structures répétitives comme les RAMs.

Mots clés

Abstraction Fonctionnelle, Désassemblage, Vérification VLSI, Conception des Circuits Intégrés, Méthodes Formelles, Dérivée Partielle Booléenne, Reconnaissance de Formes.

Abstract

This doctoral thesis concerns the verification of highly complex digital integrated circuits. More precisely, circuit disassembly and functional abstraction. The problem considered is that of the automatic extraction of a gate netlist from a transistor netlist, together with the generation of a register-transfer level description.

Not only does this gate netlist simplify subsequent timing and power consumption analysis, but also, more importantly, the RTL description allows functional verification of a VLSI circuit regardless of the design flow. In addition, it provides a method for technology migration through the resynthesis of the behavioural description onto a modern target technology.

A formal method (not requiring a user-defined library) is proposed, which uses the correlation within a circuit to extract only the useful functionality. This results in the generation of a gate netlist that is optimum in terms of gate functional characterisation.

The problem of the identification and the modelling of sequential elements is handled by an original algorithm. This algorithm uses the Boolean partial derivative to help in the analysis of the stability and the capacity to memorise of any combinatorial loops. The result is the automatic generation of a behavioural description for any kind of static latch.

The final section demonstrates the feasibility of a hybrid method, in which a pattern recognition method complements the formal method. This kind of approach brings two advantages: firstly, it becomes possible to process circuits containing analog components, secondly it allows for a more efficient handling and more optimum modelling of repetitive circuit structures, such as RAMs.

Keywords

Functional Abstraction, Circuit Disassembly, VLSI Verification, Integrated Circuit Design, Formal Methods, Boolean Partial Derivative, Pattern Recognition.

TABLES DES MATIERES

TABLE DES ILLUSTRATIONS

Chapitre 1 : INTRODUCTION

Chapitre 2 : PROBLEMATIQUE

Chapitre 3 : L'ETAT DE L'ART

Chapitre 4 : METHODE DE DESASSEMBLAGE ET D'ABSTRACTION FONCTIONNELLE

Chapitre 5 : TRAITEMENT DES ELEMENTS SEQUENTIELS

Chapitre 6 : ETUDE D'UNE APPROCHE HYBRIDE

Chapitre 7 : MISE EN ŒUVRE LOGICIELLE

Chapitre 8 : VALIDATION ET RESULTATS

Chapitre 1

INTRODUCTION

Dans ce chapitre nous présentons un aperçu de la situation actuelle de la microélectronique et du contexte dans lequel se situe cette thèse. Ensuite, nous exposons les motivations qui se trouvent à l'origine de ce travail. La présentation du plan du manuscrit achève ce premier chapitre.

1.1 La loi de Moore

En 1965, Gordon Moore, un ingénieur de Intel, en préparant un discours sur l'évolution technologique, a fait une constatation intéressante. Lorsqu'il traçait la courbe de croissance de la densité d'intégration, il a remarqué que chaque nouvelle génération de circuits était approximativement deux fois plus complexe que la génération précédente. Le temps de développement de chaque génération restait constant et constituait dix-huit mois environ. Cette constatation, connue actuellement sous le nom de « la loi de Moore », est d'autant plus remarquable que (comme la Figure 1-1 l'indique), cette loi de croissance exponentielle continue à s'appliquer même avec les circuits qui atteignent aujourd'hui une complexité de plusieurs dizaines de millions de transistors.

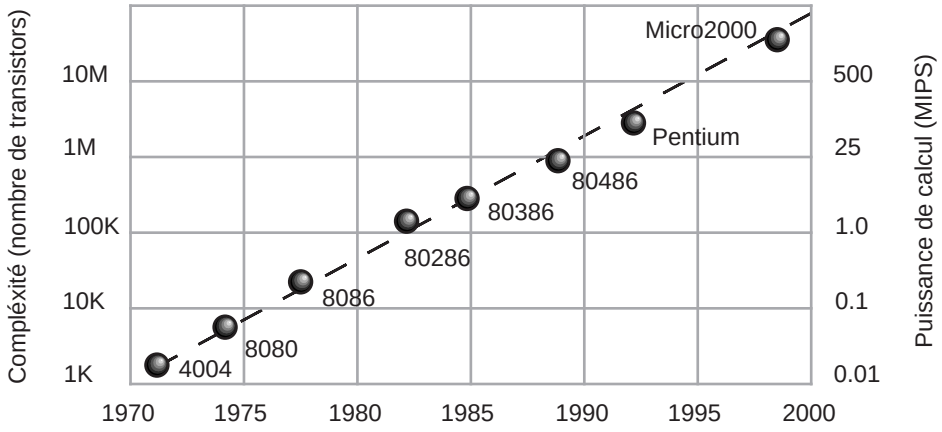


Figure 1-1 : La loi de Moore chez Intel (source : Intel Corp.)

Actuellement, la validation d'un circuit prend environ 50 % du temps total de conception. Ce temps est souvent difficile à compresser car, contrairement à la phase de la conception proprement dite, il dépend principalement du temps d'exécution des outils de vérification. En même temps, la pression afin de réduire ce temps de conception ne cesse de s'augmenter, ce qui entre nécessairement en contradiction avec le besoin pour une validation exhaustive.

Pour mieux comprendre les problèmes posés par la vérification, il suffit d'examiner l'exemple classique d'une simulation fonctionnelle au niveau portes : si on multiplie par dix le nombre de portes, le temps de simulation d'un vecteur est multiplié par dix. Mais, on a aussi besoin de dix fois plus de vecteurs pour valider le circuit. Le bilan démontre que le temps de vérification croît d'une manière au moins quadratique avec la complexité, et la complexité croît, elle-même, de manière exponentielle avec le temps.

1.2 Motivations et cadre de travail

Il est certain que l'évolution de la technologie serait plutôt néfaste si le temps nécessaire pour la conception et la validation des circuits rendait impossible, pour le concepteur, le respect des échéances imposées par le marché. Pour que la technologie disponible à un instant donné puisse être effectivement accessible, nous pouvons distinguer deux conditions primordiales.

Premièrement, il faut posséder une méthodologie permettant de maîtriser la conception sur une grande échelle, essentiellement pour pouvoir traiter des blocs de taille considérable et d'origine variée (bibliothèque de cellules, « full-custom », réutilisation IP, etc.). Par ailleurs, il faut des outils de vérification capable de gérer l'énorme volume de données que représentent ces blocs et qui puissent tirer profit des différents niveaux de modélisation (réseau de portes, description RTL, modèle C, etc.) pour conserver un temps de traitement raisonnable.

C'est sous cette optique que nous présentons, dans cette thèse, une approche de la validation fonctionnelle des circuits basés sur la technique de désassemblage. Cette approche vise la validation de la fonctionnalité d'un réseau de transistors par rapport aux spécifications initiales du comportement du circuit. Pour effectuer cette comparaison, nous sommes amenés à extraire une description au niveau RTL (Register Transfer Level) en passant par une phase de partitionnement du réseau de transistors pour obtenir un réseau de portes orienté.

Dans cette thèse, nous nous intéressons exclusivement à la technologie CMOS. Cette restriction se justifie d'une part par le constat que l'immense majorité des circuits actuels utilisent la technologie CMOS.

L'approche proposée permet de valider rapidement la fonctionnalité de blocs contenant plusieurs dizaines de milliers de transistors, quelle que soit leur méthode de conception. De plus, le partitionnement en portes rend la validation électrique ou l'analyse temporelle bien plus efficaces en introduisant l'orientation de flux de l'information (alors que cette orientation est totalement absente dans le réseau de transistors).

Fondamentalement deux techniques ont été utilisées : une première basée sur des techniques de reconnaissance de forme, et une deuxième basée sur des méthodes génériques de partitionnement du réseau de transistors. La première méthode exige une connaissance

préalable de toutes les structures utilisées dans un circuit. Cette limitation représente un désavantage indéniable.

Par ailleurs, parmi les outils qui mettent en œuvre la deuxième approche, nous avons identifié deux points faibles. Le premier est la difficulté de traiter des circuits utilisant des technique circuiterie non CMOS DUAL, tout en maintenant un temps d'exécution linéaire avec le nombre de transistors. Le deuxième porte sur le problème d'identification des éléments séquentiels. Ces composants, de fait de leur comportement mémorisant, posent des obstacles particuliers à l'élaboration d'une méthode d'abstraction générique. L'apport principal de notre travail est de proposer des solutions à ces limitations.

1.3 Plan du Manuscrit

Ce manuscrit comporte huit chapitres dont nous esquissons une brève description dans les lignes suivantes.

Le chapitre 2 expose la problématique de notre travail. Dans un premier temps, nous présentons succinctement une méthodologie de conception classique, afin de situer la place des outils de vérification et leurs fonctions. Cette présentation nous permettra, dans un deuxième temps, d'identifier deux domaines d'applications : le désassemblage et l'abstraction fonctionnelle.

Le chapitre 3 décrit l'état de l'art. Nous commençons par une description des premières tentatives. Ensuite, nous présentons l'évolution qui a abouti au développement d'outils basés sur les deux approches principales précitées : la reconnaissance de formes et les méthodes génériques.

La méthode proposée est présentée au chapitre 4. Ce chapitre décrit les principes de base de l'extraction d'un réseau de portes à partir d'un réseau de transistors. Il contient également la description du modèle général sur lequel se base la construction des portes. Nous y abordons aussi le problème primordial du traitement des corrélations qui est fondamental dans la reconstruction des portes non CMOS DUAL. Les analyses, locales et globales, appliquées sur les portes, sont exposées.

Le chapitre 5 porte sur le problème des éléments séquentiels. Nous proposons une méthode algébrique qui s'appuie sur la dérive partielle pour analyser la capacité de

mémorisation des boucles dans un circuit. La méthode permet d'identifier tous les types de latchs et de leur générer automatiquement un modèle comportemental.

Le chapitre 6 présente le développement d'une approche hybride qui permet de réunir les avantages d'une méthode générique avec ceux de la reconnaissance de formes. Cela permet de traiter les circuits mixtes analogique-numérique et donne aussi au concepteur la possibilité d'influencer la modélisation fonctionnelle.

Le chapitre 7 présente l'architecture d'un prototype de notre logiciel et explique la mise en œuvre des algorithmes. Dans notre présentation, nous mettons l'accent sur le module d'analyse fonctionnelle, qui constitue le noyau du désassemblage.

Le chapitre 8 regroupe les résultats obtenus avec notre prototype sur un ensemble de circuits réels. Trois études comparatives sont effectuées : une comparaison en performance avec les outils existant, et une comparaison des différentes approches pour l'identification des éléments séquentiels.

Le chapitre 9 fournit le bilan de cette études. En guise de conclusion, nous situons notre travail par rapport à l'idéal de l'abstraction fonctionnelle générique et complètement automatique.

Chapitre 2

PROBLEMATIQUE

La vérification des circuits de plusieurs millions de transistors exige des modifications majeures dans les méthodes de conception dans les conditions d'une croissance exponentielle des volume de données à traiter.

Le problème abordé dans notre thèse est celui de la définition et de l'expérimentation d'une approche de vérification, basée sur l'extraction automatique d'une description comportementale, à partir du réseau de transistors extrait du dessin des masques.

2.1 Méthodologie de conception

Dans cette thèse, nous nous intéressons à la vérification de circuits CMOS possédant un cœur numérique, mais pouvant contenir quelques fonctions analogiques. La conception de tels circuits s'effectue en deux phases principales. La première phase est celle de la synthèse descendante. Il s'agit de la génération d'une description de bas niveau, à partir d'une spécification ou d'une description de haut niveau. Cette spécification est en général fournie dans un langage de description de matériel, comme le langage VHDL [IEE88].

La deuxième phase est celle de la vérification. Dans les méthodes classiques, la vérification du résultat final se fait en s'appuyant principalement sur la (ou les) description(s) structurelles (schémas) générée(s) dans la phase descendante. La seule partie ascendante est une extraction des capacités et des résistances à partir de la description physique, permettant une retro-annotation des schémas. Ce type de méthode est représenté dans la Figure 2-1.

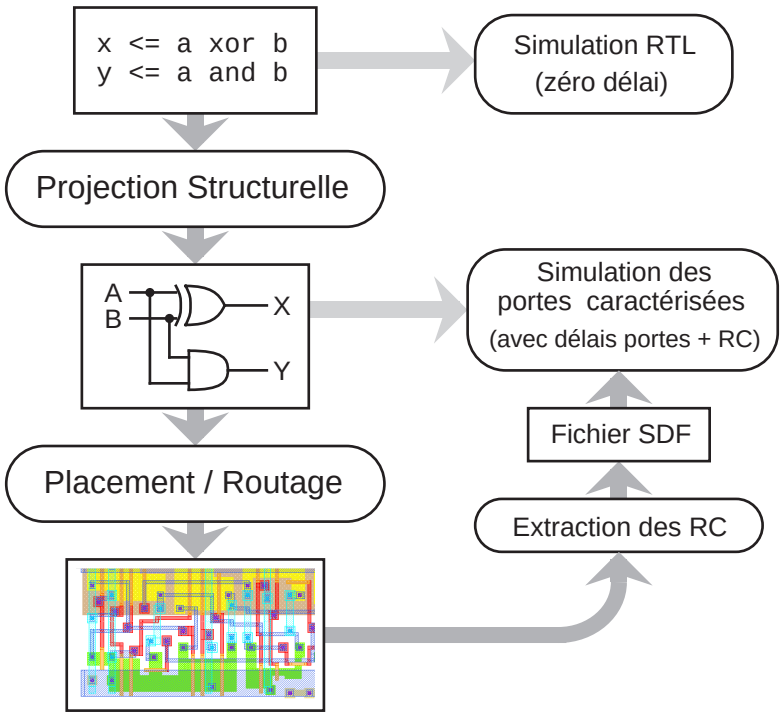


Figure 2-1 : *Une méthode de conception classique*

La vérification fonctionnelle se limite à une simulation du réseau de portes issu de la synthèse pour les blocs construits à partir de bibliothèque de cellules. L'analyse temporelle ou l'analyse de la consommation s'effectue, à partir du réseau de portes, après une phase de retro-annotation par les capacités et les résistances d'interconnexion extraites. En l'absence

d'une description en réseau de portes (par exemple pour les blocs « full-custom »), il n'existe pas d'autres alternatives que la simulation électrique du réseau de transistors obtenu par extraction.

Les limitations de cette approche sont évidentes. Dans un contexte de synthèse basée entièrement sur une bibliothèque de cellules, les résultats de la phase de vérification sont bons. Mais, dans un environnement de conception mixte, dans lequel le circuit se compose de blocs d'origines diverses, il devient plus compliqué d'obtenir des résultats fiables pour les vérifications électriques ou temporelles. Ceci est typiquement le cas des systèmes intégrés modernes contenant des parties composées de cellules précaractérisées, des parties « full-custom », et souvent des blocs IPs. Dans ce contexte, pour certaines parties du circuit, on ne dispose pas de descriptions au niveau porte permettant de faciliter les vérifications fonctionnelles, temporelles ou l'évaluation de la consommation.

Dans cette thèse, nous présentons une méthode de vérification basée sur le désassemblage. La vérification s'appuie sur une description structurelle au niveau porte logique obtenue à partir de la description au niveau transistor. Cette méthode permet d'appliquer une stratégie cohérente et homogène pour la vérification d'un circuit malgré les origines diverses de ses composants.

Cette approche n'est pas totalement originale. Comme nous le verrons dans le chapitre « Etat de l'art », des outils de vérification exploitant le désassemblage sont déjà utilisés dans l'industrie. Au sein de notre laboratoire, deux études significatives ont été menées au cours de ces dernières années. En s'appuyant sur une des deux méthodes principales existantes. La première est la thèse de Marc Laurentin qui à abouti à l'outil DESB [Lau94]. Cette réalisation constitue le point de départ de nos recherches. Son approche automatique permet en théorie le traitement d'une diversité importantes de styles de circuits, tout en limitant l'intervention de la part de l'utilisateur. La deuxième étude, la thèse de Jean-Bruce Guignet [Gui98] a permit le développement de l'outil GROG. Il s'agit d'une approche par reconnaissance. Malgré la contrainte liée à la nécessité de définir une bibliothèque de formes à reconnaître, l'approche demeure intéressante en raison de sa configurabilité.

Dans les deux sections suivantes, nous illustrons, à l'aide de quelques exemples, dans quelles mesures la technique de reconstruction d'un réseau de portes à partir d'un réseau de transistors facilite la vérification. Nous verrons également l'impact de cette technique sur la

méthode de conception. Nous avons identifié deux domaines d'applications à cette méthode : le désassemblage et l'abstraction fonctionnelle.

2.2 Le désassemblage

2.2.1 Définition

Le désassemblage se situe dans la phase ascendante, après l'étape d'extraction. Il s'agit de la conversion d'un réseau de transistors, non-orienté mais pourvu des informations physiques nécessaires aux outils de vérification, en un réseau de portes orienté.

Ces portes contiennent toutes les informations électriques contenues dans le réseau de transistors (dimensions des transistors, valeur des capacités, résistances, etc.). Mais en plus, cet ensemble de portes a été enrichi par des données supplémentaires : les interdépendances fonctionnelles entre les portes et l'orientation du flux d'information.

Dans le cas où le circuit provient d'une synthèse utilisant une bibliothèque de cellules, ce découpage permet de retrouver les cellules utilisées par l'outil de synthèse lors de l'étape de projection structurelle.

Par ailleurs, dans le cas où le circuit contient des blocs optimisés pour lesquels il n'existe pas de descriptions au niveau porte, cette approche permet de découper les blocs en un ensemble d'objets qui s'apparentent à des portes logiques.

2.2.2 Les niveaux de vérification

Dans les méthodes de conception actuelle, la vérification d'un circuit intégré se situe principalement à deux niveaux. Pour comprendre dans quelle mesure le désassemblage peut faciliter la tâche de vérification, ou la rendre tout simplement possible, nous commençons par expliquer les avantages et les difficultés rencontrés à chacun de ces niveaux.

2.2.2.1 Le niveau transistor

Une vérification au niveau transistor correspond à l'exploitation directe du réseau de transistors. La simulation électrique de type SPICE [Nag75] est considérée comme la référence en termes de précision. Dans cette approche, un circuit est modélisé par un système d'équations différentielles. Ce système d'équations est obtenu à partir des modèles de transistor, tels que le modèle Scichman-Hodges [Shi68]. Cette méthode permet d'effectuer

avec une haute fidélité de nombreuses analyses du comportement électrique du circuit : analyse statique ou transitoire, réponse en fréquence ou distorsion. Les simulateurs traditionnels utilisent des méthodes itératives pour résoudre le système d'équations pour chaque instant. Il est évident que ces méthodes sont forcément coûteuses en temps de calcul.

Il existe d'autres simulateurs moins précis. Afin de simuler des circuits numériques de haute complexité au niveau transistor, ces simulateurs modélisent le transistor MOS comme un simple interrupteur, piloté par le signal de sa grille. Si le transistor conduit, il permet de transférer une valeur de la source vers le drain ou inversement. Il faut noter que ces simulateurs n'utilisent aucune orientation des transistors, et c'est uniquement le contexte de la simulation qui définit la direction du transfert. Ces approches sont capables de tenir compte de certaines caractéristiques de la circuiterie MOS, telles que le partage de charge et la bidirectionnalité.

L'avantage de l'approche au niveau transistor repose dans un premier temps sur la précision offerte par la simulation électrique. Dans un deuxième temps, les méthodes de simulation électrique ouvre la possibilité d'un choix du modèle de transistor, permettant ainsi une maîtrise partielle du compromis entre la précision et la rapidité.

2.2.2.2 Le niveau porte

Une approche plus efficace consiste à voir un circuit numérique comme un réseau de portes orienté plutôt qu'un réseau de transistors non-orienté. Dans ce cas, l'ensemble des portes est considéré comme un système numérique où chaque signal possède un nombre réduit d'états possibles. Cette vision permet d'obtenir une modélisation simple du comportement du circuit. Par exemple, il est possible de modéliser chaque signal par deux états logiques '0' et '1' en se basant sur les expressions booléennes associées aux portes.

Dans cette approche, le réseau de portes peut être simulé grâce à des méthodes de simulation performantes, telles que la simulation à pilotage événementiel (Event Driven). Cette technique permet de simuler des circuits contenant plusieurs millions de transistors. En effet, dans cet algorithme, la sortie d'une porte n'est évaluée que si un changement se produit sur au moins une de ses entrées. Ainsi, à chaque pas de simulation, on effectue une réévaluation de l'état des signaux uniquement dans les zones actives du circuit.

Malheureusement, une analyse au niveau porte ne peut fournir des informations précises dans le domaine temporel que dans le cas où chaque composant correspond à un élément d'une bibliothèque de portes dont le comportement temporel a été caractérisé préalablement. Ainsi, pour pouvoir tirer avantage d'un traitement au niveau porte - indispensable pour la vérification des circuits numériques de grande taille - la conception d'un circuit doit se baser entièrement sur une bibliothèque de cellules précaractérisées. Des méthodes de conception qui incorporent plusieurs approches, comme l'utilisation de générateurs de blocs optimisés ou le « full-custom » échappent donc à cette approche.

2.2.3 Les domaines d'applications

Le désassemblage peut être vu comme une étape de préparation préalable pour les diverses tâches de vérification. A titre d'exemple nous examinons quelques outils de vérification qui exploitent le désassemblage pour générer une représentation plus adaptée du circuit. Cette nouvelle approche de vérification a été expérimentée dans le cadre de plusieurs thèses et les techniques mises en œuvre dans ces outils ont été validées ou sont en cours de validation sur des circuits complexes.

2.2.3.1 L'analyse temporelle

Dans le cas d'un circuit synchrone, l'objectif d'un outil d'analyse temporelle est de vérifier les spécifications temporelles, c'est-à-dire le respect des contraintes imposées par le temps de cycle. A cet effet, il est nécessaire de calculer les temps de propagation à travers les chemins fonctionnels du circuit afin de vérifier que les temps de prépositionnement et de maintien pour l'écriture dans les registres sont respectées. Les outils d'extraction fournissent, à partir du dessin des masques, un schéma de transistors ainsi que les résistances et les capacités parasites. Cette information constitue le point de départ de la vérification temporelle.

Afin d'apporter des informations temporelles utiles, une simulation électrique au niveau transistor doit s'effectuer avec un modèle de transistor relativement précis ; un simple modèle interrupteur ne suffit pas. Cette approche, dite analyse dynamique, devient alors rapidement prohibitive pour des circuits réels à cause du temps d'exécution et de la difficulté à générer un ensemble de stimuli caractéristiques du fonctionnement du circuit.

Dans une autre approche, dite statique, l'outil d'analyse est chargé d'identifier les chemins critiques qui peuvent mettre en échec les spécifications temporelles, sans exiger de

stimuli [Ous85]. On obtient, ainsi, une accélération considérable du temps d'exécution qui ne nécessite plus de recours à la simulation. De ce fait, il est possible de traiter des circuits de grande complexité. Cependant, cette accélération ne s'obtient qu'au prix d'une perte de précision. En effet, l'analyse statique s'appuie, pour la recherche des chemins, sur un réseau de portes orienté.

Le désassemblage permet de combiner les avantages de l'analyse dynamique et l'efficacité de l'approche statique. En construisant un réseau de portes à partir du réseau de transistors, il ouvre la possibilité d'une analyse statique et l'accélération qui en découle. De plus, puisque ce réseau de portes hérite des informations contenues dans le réseau de transistors, la technique de désassemblage permet d'accéder au même niveau de précision que l'approche dynamique avec la même flexibilité sur le choix des modèles. Enfin, et c'est sans doute le point le plus important, le désassemblage ouvre la possibilité du traitement des blocs « full-custom » par la même approche que celui des blocs contenant des cellules standard.

L'approche résultant de la combinaison du désassemblage et de l'analyse statique a été mise en œuvre et expérimentée dans un outil développé au laboratoire LIP6. Cet outil appelé TAS [Haj91] utilise un modèle de transistor dit modèle à canal court particulièrement bien adapté au circuits actuels. Les paramètres de ce modèle sont obtenus à travers des simulations électriques en fonctions des caractéristiques de la technologie de fabrication. Ce modèle est ensuite utilisé sur le réseau de portes issu du désassemblage du circuit.

2.2.3.2 La simulation logico-temporelle

L'évolution de la technologie permet aujourd'hui de réunir sur une même puce des parties numériques et des fonctions analogiques. La vérification fonctionnelle de tels circuits nécessite le recours à une technique de simulation mixte analogique-numérique. En général, ce type de simulateur se construit en combinant un simulateur numérique et un simulateur analogique. Or, pour pouvoir fournir au simulateur analogique des informations pertinentes, le simulateur numérique doit être capable de calculer avec précision l'instant de commutation des signaux et la forme des transitions à l'interface analogique-numérique.

Dans le contexte de la simulation logico-temporelle, l'avantage d'une méthode générale, pour remonter à une description en portes à partir du réseau de transistors, est nettement démontré. Toutefois, contrairement aux circuits conçus à partir d'une bibliothèque de cellules, la vérification des circuits contenant des blocs « full-custom » ne peut se baser sur

une précaractérisation des portes. Pour ces circuits, la technique de désassemblage, en plus d'accélérer la simulation, permet d'obtenir une grande précision dans le domaine temporel par une caractérisation « à la volée » des portes. En effet, le désassemblage permet de reproduire, dans le réseau de portes, toutes les informations nécessaires à un tel calcul, y compris la structure du réseau de transistors qui composent la porte et leurs caractéristiques.

Ces raisons ont motivé le développement d'un simulateur logico-temporelle au Laboratoire LIP6 dans le cadre d'une thèse de doctorat [Abd98]. Le simulateur exploite la technique de désassemblage que nous proposons dans cette thèse et que nous avons expérimenté. L'outil exploite le réseau de portes extrait. La caractérisation temporelle des portes se fait lors de la simulation et utilise le modèle à canal court de TAS. Basé sur un moteur de simulation évènementielle, cet outil a permis d'effectuer des simulations avec une erreur de quelques pour cent par rapport à SPICE. Pour les circuits de petite taille, le facteur d'accélération, par rapport à SPICE, est de trois ordres de grandeur. Mais surtout, grâce à cette méthode, il est possible de traiter des circuits bien plus complexes.

2.2.3.3 L'analyse de la consommation

Dans les méthodes de vérifications des circuits intégrés modernes, principalement deux raisons poussent à une analyse de la consommation. D'une part, l'expansion du marché de la téléphonie mobile et des ordinateurs portables a créé un besoin pour des produits dont l'autonomie est une caractéristique primordiale. De l'autre, la densité d'intégration des composants modernes et la fréquence de fonctionnement élevée entraînent des difficultés d'évacuation de la chaleur. Un taux d'activité très élevé dans une zone restreinte d'un composant peut amener à une surchauffe ponctuelle. Ces phénomènes peuvent réduire la durée de vie des composants parfois de manière drastique.

Une fois de plus, une simulation électrique de type SPICE est la méthode qui fournit la plus grande précision. Toutefois, la nécessité d'effectuer ces simulations avec un jeu de vecteurs de taille importante rend cette approche totalement inapplicable pour tous les circuits de taille réelle. Cependant, pour l'évaluation de la consommation, la précision est une exigence bien moins prépondérante que la rapidité et la possibilité de prise en compte de circuits complexes. Une approche au niveau transistor devient alors faisable, avec un modèle de transistor de type interrupteur comme dans l'outil *PowerMill* [Epi97].

Pour les circuits CMOS, la consommation est essentiellement due aux transitions entre deux états stables. On peut distinguer trois sources de consommation dynamique : la consommation due à la commutation de la capacité externe d'une porte, la consommation de court-circuit, et la consommation due aux commutations des capacités internes. Un outil d'évaluation de la consommation doit évaluer la fréquence et l'energie dissipée de chaque transition.

Les outils d'analyse de la consommation sont relativement récents. Pourtant, en ce qui concerne une analyse au niveau portes du circuit, on arrive à distinguer deux approches différentes. La première approche fait appel à des méthodes de simulation classique. Lors de la simulation, il suffit de comptabiliser les transitions de chaque signal pour obtenir une moyenne statistique de son activité [Kan86]. L'outil *Entice-Aspen* [Geo94] utilise cette première approche. La deuxième approche, plus originale, est basée sur une analyse probabiliste de l'activité du circuit [Naj91][Mar94].

Avec une méthode automatique d'extraction de portes, l'analyse de la consommation ne dépend plus d'une bibliothèque de cellules et le traitement de tous les types de circuits numériques devient possible. Il convient de mentionner ici les recherches effectuées au laboratoire LIP6 et qui combinent une identification automatique des portes avec une approche probabiliste.

2.3 L'abstraction fonctionnelle

2.3.1 Définition

De manière générale, l'objectif de l'abstraction fonctionnelle est d'arriver à établir une description fonctionnelle de haut niveau à partir d'une description de bas niveau. Dans cette thèse, nous considérons que le point de départ est le réseau de transistors, extrait à partir de la vue physique par un outil d'extraction. Le haut niveau correspond à une description RTL (register-transfer level) du circuit.

L'objectif de l'abstraction fonctionnelle est donc d'obtenir un ensemble d'expressions booléennes, qui définissent le comportement du circuit, à partir du réseau de transistors. Pour établir ces expressions booléennes, il faut identifier les points mémorisants du circuit et construire la fonction logique des parties combinatoires qui se situe entre ces points mémorisants. Ainsi, l'abstraction fonctionnelle comporte deux étapes : le désassemblage, qui

réalise un partitionnement du réseau de transistors non-orienté en un réseau de portes orienté, et la caractérisation fonctionnelle de chaque porte qui sert à en déduire un modèle comportemental du circuit.

Dans les pages qui suivent nous allons tenter de fournir quelques exemples d'applications de l'abstraction fonctionnelle dans la conception des circuits intégrés. Ces exemples permettent d'illustrer les motivations qui se trouvent à l'origine de ce domaine de recherche.

2.3.2 La vérification fonctionnelle

2.3.2.1 La méthode classique

Dans la section précédente, nous avons passé en revue différents types de vérification (temporelles, consommation, etc.) effectués lors de la phase ascendante de la conception et nous avons examiné le rôle que peut y jouer la technique de désassemblage. Nous allons abordé ici le problème de la vérification fonctionnelle.

Il existe diverses méthodes pour vérifier que le comportement d'un circuit reste conforme à sa spécification d'origine. Suivant la méthode utilisée pour la conception du circuit, cette vérification peut être plus ou moins ardue.

Dans les méthodes classiques de conception, la réalisation du circuit se base sur une bibliothèque de cellules précaractérisées. Lors de l'étape de synthèse, on transforme la description RTL du circuit en un réseau de portes en utilisant les cellules de la bibliothèque. A ce stade, il est facile de vérifier le travail de l'outil de synthèse puisque, pour chaque cellule de la bibliothèque, on dispose de son comportement.

A l'étape suivante, le réseau de portes est placé et routé. La vérification fonctionnelle doit passer, ici, par une phase d'extraction. Si l'extracteur est capable de séparer le réseau de routage des cellules, on retrouve un réseau de portes avec une expression booléenne associée à chaque porte. Dans le cas contraire, la description extraite représente un réseau de transistors. Cette description peut être vérifiée soit par une simulation au niveau transistor, soit par une comparaison avec le réseau obtenu après l'étape de synthèse.

Cette méthode de validation fonctionnelle est résumée à la Figure 2-2. On constate que la stratégie de validation, tout au long de la conception, dépend de l'existence d'une description au niveau porte.

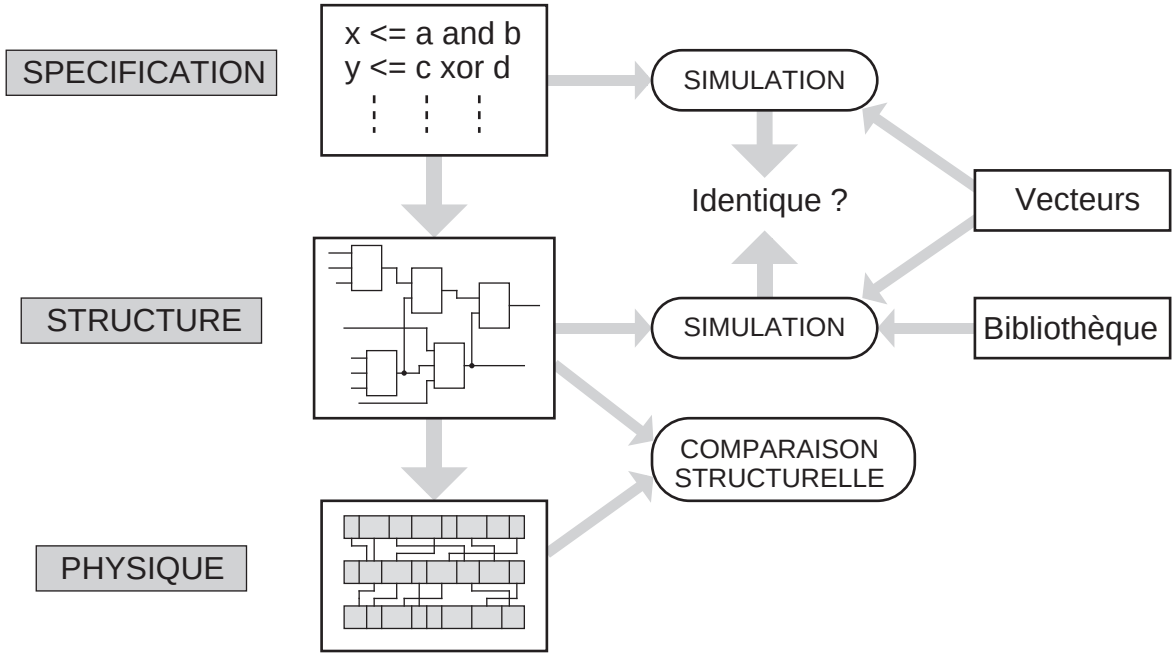


Figure 2-2 : *Méthode de validation classique*

2.3.2.2 La méthodologie utilisant l'abstraction fonctionnelle

Cependant, il existe des situations où la description en portes d'au moins une partie du circuit n'est pas disponible. Un concepteur introduit parfois des blocs « full-custom » dessinés à la main soit pour combler des lacunes de la bibliothèque, soit pour optimiser les performances ou la surface d'un bloc. La vérification des circuits contenant de tels blocs devient difficile d'autant plus que, contrairement aux cellules de la bibliothèque, ces blocs n'ont jamais été validés dans d'autres contextes.

La validation des circuits qui contiennent des blocs issus de générateurs optimisés pose pratiquement les mêmes difficultés. Dans les circuits actuels, l'utilisation de ces générateurs devient de plus en plus courante. Ils permettent de réunir les avantages de facilité de conception et de robustesse de l'approche par bibliothèque, avec les optimisations de performance et de surface apportées par l'approche « full-custom » [Gru97]. Les générateurs sont des modules logiciels qui, à partir d'un ensemble de paramètres fournis par l'utilisateur, créent directement les dessins de masques. Cette méthode de conception s'avère très efficace pour des blocs dont la structure varie d'une manière algorithmique en fonction des paramètres, tels que les opérateurs arithmétiques ou les bancs de registres.

Les générateurs s'appuient sur une bibliothèque de cellules physiques, qui sont assemblées selon des algorithmes propres à chaque générateur. Il est impossible de caractériser les cellules indépendamment, ni de leur associer un comportement. Par conséquent, les circuits issus de ces générateurs n'ont pas de représentation en portes. Toutefois, le générateur peut fournir un modèle comportemental simulable pour permettre la vérification fonctionnelle de l'ensemble du circuit. Mais il n'existe pas de moyen direct de vérifier la concordance de la description physique avec son modèle comportemental.

L'abstraction fonctionnelle est le seul moyen de vérification fonctionnelle des blocs générés. Le comportement est obtenu à partir du réseau de transistors extrait à son tour du dessin des masques. Notons que cette technique permet non seulement la validation des circuits contenant de tels blocs, mais aussi, elle permet de valider le générateur en comparant le modèle comportemental d'un bloc avec le comportement obtenu à partir du dessin des masques.

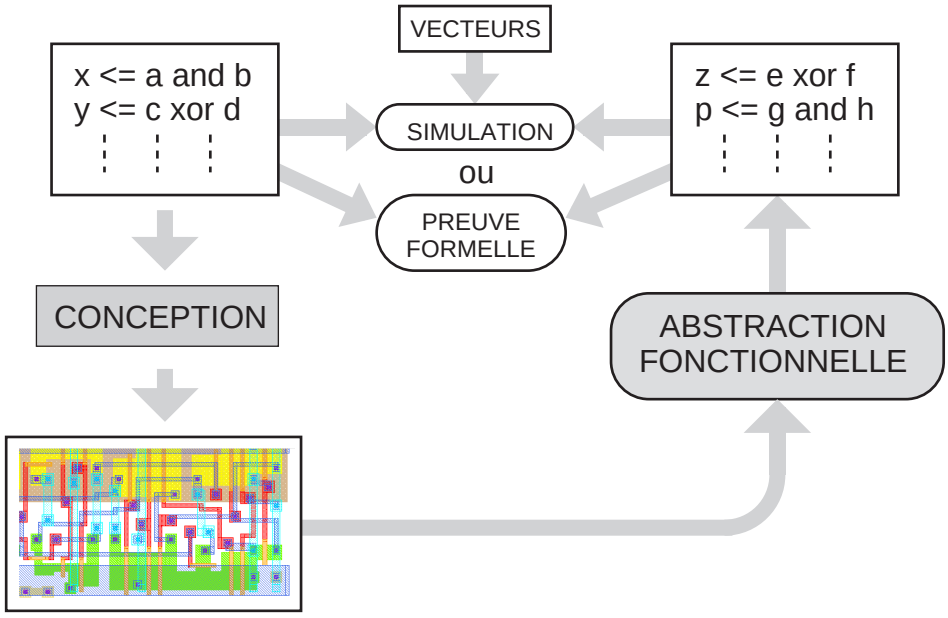


Figure 2-3 : *Validation par l'abstraction fonctionnelle*

La Figure 2-3 montre la méthode de vérification basée sur l'abstraction fonctionnelle. Nous constatons, notamment, qu'elle ne suppose pas l'existence d'un réseau de portes issu de la phase de conception. Remarquons aussi que les phases descendante et ascendante sont bien séparées, et la description physique est directement validée par rapport à la spécification initiale. Cependant, l'utilisation des réseaux de portes issus de la phase descendante, quand ils existent, contribue à la localisation des erreurs éventuelles.

2.3.3 La migration technologique

La rapidité de l'évolution du marché de la micro-électronique, impose aux fabricants de circuits de commercialiser des produits dans des délais de plus en plus courts, sous peine de se voir dépassé par leurs concurrents. Or, une partie non négligeable du temps de conception est mobilisée par la validation des spécifications du circuit. De ce fait, la réutilisation de blocs ou de circuits déjà conçus, et qui ont fait la preuve de leur fonctionnement correct dans des utilisations antérieures, peut représenter un gain important.

Malheureusement, la rapidité de l'évolution de la technologie rend la réutilisation en état de ces blocs inappropriée. En effet, l'existence même de ces blocs suggère que la technologie avec laquelle ils ont été fabriqués est obsolète. Aussi, pour pouvoir profiter de l'évolution de la technologie, les fabricants sont tentés de reproduire les mêmes blocs avec des technologies nouvelles. Cette opération est appelée la migration technologique.

Il existe plusieurs approches pour répondre au problème de la migration technologique. La plupart sont basées sur des techniques de transformations des masques [Mcl96]. Ces approches ont l'avantage de travailler directement sur les masques de fabrication sans aucune référence à la fonctionnalité des blocs. Par conséquent, elles peuvent, en théorie, être exécutées rapidement, surtout si les modifications restent localisées. Ces approches ont, néanmoins, plusieurs défauts. Premièrement, il est difficile de cibler plusieurs technologies d'une manière générique, car les règles de dessins, surtout dans la conception submicronique, imposent des contraintes de types différents d'une technologie à une autre. Deuxièmement, le passage d'une technologie à une autre ne peut pas se faire par simple homothétie et les règles de transformation des masques ne sont en général pas immédiates. Troisièmement, des erreurs fonctionnelles, qui proviennent des courts-circuits ou des circuits ouverts, introduits par ces transformations, sont difficilement repérables.

L'abstraction fonctionnelle permet une approche générique à la migration technologique. Cette approche est illustrée à la Figure 2-4. Il s'agit d'une conception à l'envers (reverse engineering). Elle comprend deux phases consécutives : d'abord, la phase ascendante qui permet de produire une description RTL à partir du réseau de transistors extrait, et, ensuite, la phase descendante de synthèse classique par bibliothèque, suivi par le placement et le routage. Bien que cette approche évite tous les problèmes associés à la transformation des masques, elle n'est pas sans défauts. Notamment, il est difficile de valider le travail de

l'abstraction fonctionnelle, car on ne dispose pas toujours des spécifications sous une forme adaptée pour effectuer une comparaison. De plus, il est difficile de tirer profit des techniques de conception performantes telles que le « full-custom ».

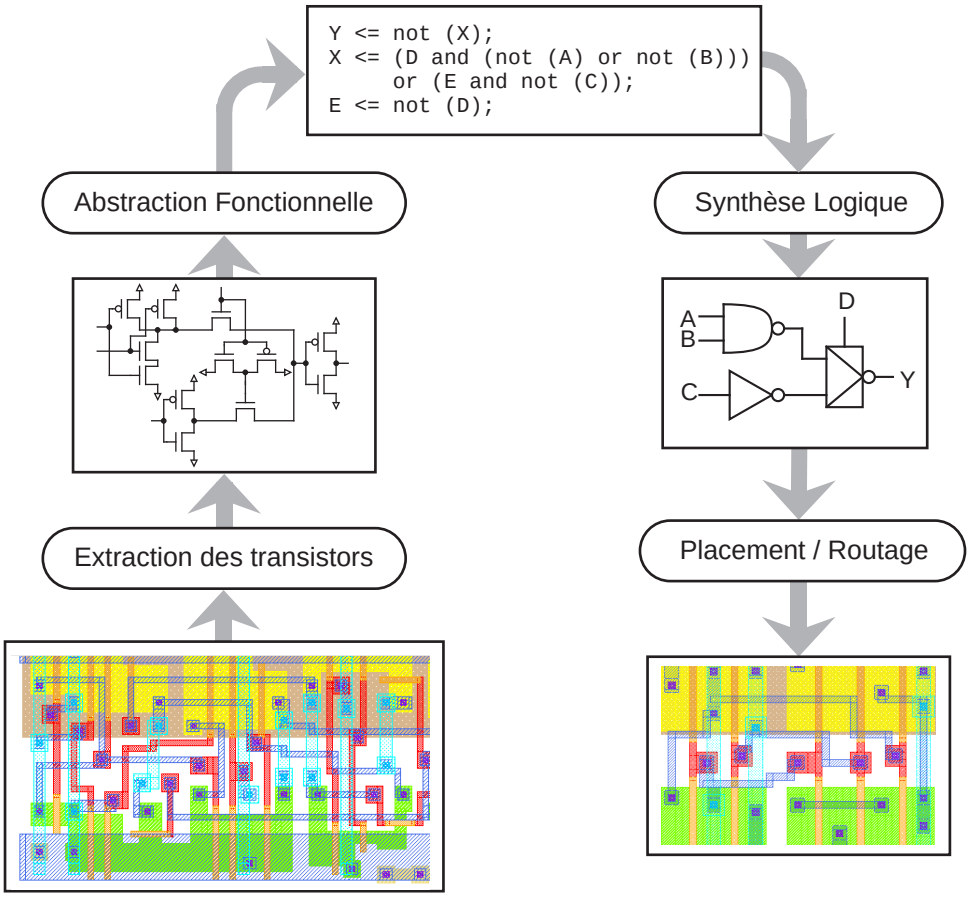


Figure 2-4 : *Migration technologique par l'abstraction fonctionnelle*

2.4 Les différentes approches envisageables

A la lumière de ces exemples, il apparaît clairement qu'une méthode de vérification basée sur l'extraction d'un réseau de portes à partir d'un réseau de transistors peut aboutir à une amélioration notable des performances des outils de vérification.

Comme nous le verrons en détail au chapitre 3 (« L'état de l'art »), deux techniques peuvent être utilisées pour réaliser ce travail : la reconnaissance de formes et le partitionnement générique.

2.4.1 La reconnaissance de formes

La reconnaissance de formes est la méthode la plus intuitive pour effectuer le désassemblage et/ou l'abstraction fonctionnelle. Il s'agit de rechercher explicitement, dans le

réseau de transistors, des formes particulières dont la structure et le modèle comportemental sont connus.

Chaque fois qu'une forme est reconnue, elle est remplacée par un composant de plus haut niveau, tel qu'une porte CMOS. Un modèle comportemental, défini pour ce composant, est alors inséré dans la description fonctionnelle du circuit. Cette procédure peut aussi s'effectuer d'une manière hiérarchique pour reconnaître des interconnexions de blocs précédemment abstraits.

2.4.2 Les approches génériques

Une approche générique réalise l'abstraction fonctionnelle par une analyse purement algorithmique du réseau de transistors. Aucune bibliothèque de formes n'est nécessaire. Ce type d'approches minimise donc la connaissance préalable des structures utilisées dans le circuit.

La mise en œuvre de cette approche comporte invariablement deux étapes. Une première étape de partitionnement où le réseau de transistors est découpé en un ensemble connexe de sous-réseaux. Ce partitionnement doit être complet, c'est-à-dire chaque transistor doit appartenir à au moins un des sous-réseaux. Toutefois, il ne s'agit pas forcément d'une partition au sens strict dans la mesure où un transistor peut appartenir à plusieurs sous-réseaux. La deuxième étape consiste à analyser individuellement chaque sous-réseau pour lui attribuer un modèle comportemental.

2.4.3 Les limitations de chaque approche

Chacune de ces deux techniques présente des avantages et des inconvénients dont nous discutons en détail à l'occasion du chapitre « L'état de l'art ». Dans cette section, afin de définir le plus clairement possible la problématique de notre recherche, nous nous contentons d'esquisser les limitations de chaque approche pour lesquelles nous avons tenté d'apporter une solution dans cette thèse.

2.4.3.1 Existence d'une bibliothèque

L'obligation de disposer d'une bibliothèque de structures à reconnaître présente certainement l'inconvénient majeur des approches par reconnaissance de formes. La méthode la plus simple consiste à définir une bibliothèque des portes utilisées dans le circuit et dont

chaque élément est un schéma en transistors d'une porte. Cette approche semble être bien adaptée aux circuits conçus à l'aide d'une bibliothèque de cellules connues, mais beaucoup moins à la migration technologique d'un circuit dont on ne possède pas le schéma.

2.4.3.2 Le problème des corrélations

Du point de vue du concepteur, les approches génériques sont plus simples à utiliser dans la mesure où elles ne nécessitent pas la définition préalable d'une bibliothèque de formes. Elles sont basés sur l'orientation des transistors dans le circuit.

La circuiterie MOS est caractérisée par le fait que les transistors n'ont pas d'orientation intrinsèque à cause de la symétrie totale de la source et du drain. Il existe de nombreux cas où l'orientation est simple. Par exemple, dans une porte CMOS DUAL classique, il n'y a aucune ambiguïté. Toutefois, étant donné cette symétrie, il est possible de concevoir des circuits avec des transistors bidirectionnels. Ce sont des cas rares. Néanmoins, il reste un grand nombre de situations pour lesquelles c'est uniquement le contexte qui détermine l'orientation des transistors. La Figure 2-5 fournit un exemple typique de ces cas. Il est évident que la propagation de l'information dans les transistors pilotés par les signaux C et D s'effectue selon les flèches. Ainsi, l'état du nœud F n'influence jamais celui du nœud B. Cependant, cette orientation n'est pas explicite dans le réseau de transistors. La seule manière de détecter le non passage de l'information entre F et B est d'exploiter la corrélation entre C et D.

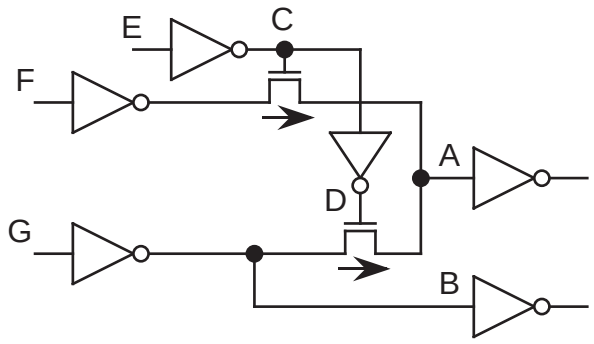


Figure 2-5 : *Orientation des transistors*

Dans l'exemple de la Figure 2-5, le problème ne paraît pas complexe, car la corrélation entre C et D est une simple inversion. Néanmoins, il est souvent bien plus difficile de mettre en évidence cette corrélation. De plus, la recherche et la prise en compte de ces corrélations a une très forte influence sur le temps de traitement du circuit. La définition d'une méthode de recherche de ces corrélations associée à une méthode d'orientation des transistors constitue le cœur de la problématique des approches génériques. La définition d'une méthode efficace est

la condition indispensable pour pouvoir traiter des circuits de taille réelle dans des temps raisonnables.

2.4.3.3 Le problème des éléments séquentiels

Le traitement des éléments séquentiels constitue une difficulté particulière pour les approches dites génériques. Un nœud mémorisant possède non seulement des émetteurs qui lui imposent un état, mais aussi un mécanisme de maintien d'un état dans l'absence de pilotes actifs. Leur détection est essentielle dans toute analyse fonctionnelle. De même, les registres constituent les extrémités des chemins temporels. Leur détection est nécessaire, si on souhaite effectuer une analyse temporelle sur le réseau de portes désassemblées.

Il y a ainsi deux enjeux. Premièrement, il faut détecter les nœuds mémorisant afin de délimiter les parties combinatoires d'un circuit. Deuxièmement, il faut distinguer les pilotes d'un nœud de type registre du mécanisme de sa mémorisation. La Figure 2-6 illustre les difficultés associées à cette distinction. Les deux latchs de la figure réalisent exactement la même fonction : échantillonnage sur l'état haut de l'horloge avec une mise à zéro asynchrone. Nous voudrions, alors, leurs affecter le même modèle comportemental. Cependant, les deux formes de latchs sont très différentes de point de vue électrique.

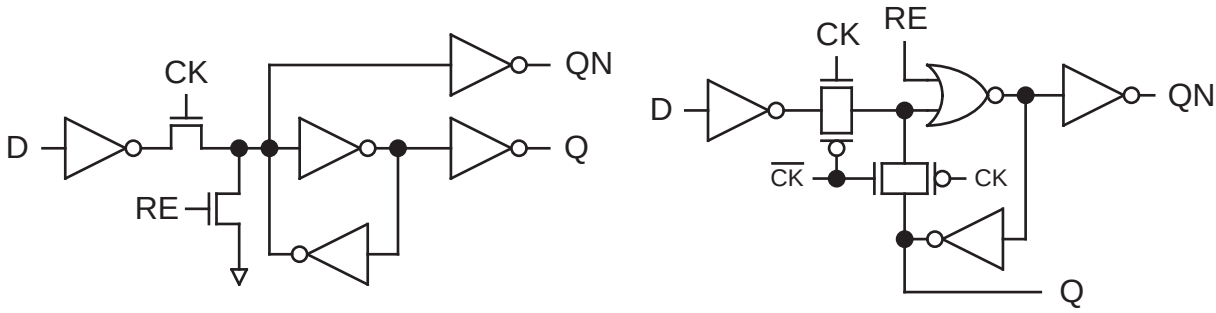
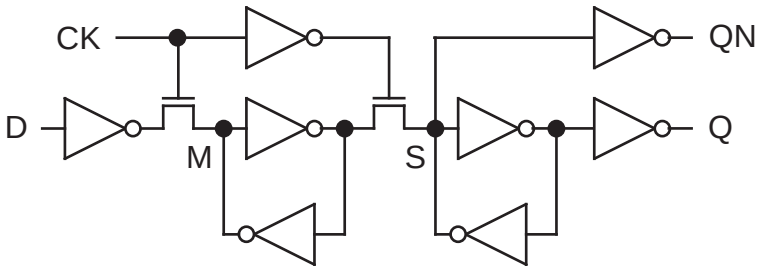


Figure 2-6 : *Deux schémas possibles d'un latch avec mise à zéro asynchrone*

La première forme est un latch dit à conflit : pendant la phase d'écriture, le pilote est en conflit avec la boucle de mémorisation. C'est la puissance relative du pilote par rapport à l'inverseur de rebouclage qui rend l'écriture possible. De la même façon, le transistor piloté par le signal de mise à zéro doit être encore plus puissant pour que la mise à zéro soit prioritaire. Ainsi, le dimensionnement physique des transistors dicte le comportement. En ce qui concerne la deuxième forme, il n'y a jamais de conflits sur le nœud mémorisant. Pendant la phase d'écriture, la boucle de mémorisation est coupée. La mise à zéro reste prioritaire et asynchrone, car elle conditionne la valeur lue sur la sortie.

La difficulté de fournir un modèle adéquat pour certaines structures est un autre problème. Par exemple, on réalise les bascules à échantillonnage sur front en mettant deux latchs en série avec une inversion de l'horloge entre les deux (Figure 2-7). Il est alors possible de décrire le composant soit en décrivant deux latchs, soit en utilisant un modèle qui exprime directement la dépendance sur le front de l'horloge. Le premier modèle est plus proche de la réalité, tandis que le deuxième reflète plutôt la vision du concepteur.

Dans cette optique, une méthode d'abstraction fonctionnelle doit permettre une souplesse dans la modélisation. Cela est vrai non seulement pour les descriptions des éléments séquentiels, mais aussi pour des composants analogiques. On rencontre de plus en plus souvent des circuits mixtes, c'est-à-dire des circuits principalement numériques, mais contenant un petit nombre des composants analogiques. Pour ces composants, la modélisation dépend directement des besoins de la vérification et la concordance entre le souhait du concepteur et le choix du modèle est primordial.



Modèle en une bascule :

```
process (CK)
begin
  if (CK'event and CK = '0') then
    S <= D;
  end if;
end process;
```

Modèle en deux latchs :

```
process (CK)
begin
  if (CK = '1') then
    M <= not D;
  end if;
end process;

process (CK)
begin
  if (CK = '0') then
    S <= not M;
  end if;
end process;
```

Figure 2-7 : *Deux modélisations possibles d'une bascule*

2.4.4 Une approche hybride

Comme nous venons de le voir, chacune des deux approches présente des avantages et des inconvénients. La méthode par reconnaissance nécessite une bibliothèque de formes. Le recours à une bibliothèque est une contrainte, puisque le concepteur est obligé de connaître et de décrire les structures contenues dans le circuit. Cependant, la bibliothèque permet d'utiliser cette méthode pour tous les types de technologies y compris les circuits mixtes analogiques-numériques. Elle permet aussi de contrôler le modèle généré.

La méthode générique, par contre, apporte l'avantage d'être une approche automatique qui offre la possibilité de traiter des circuits dont les structures sont complètement inconnues. Toutefois, l'impossibilité de traiter des composants analogiques, et l'absence de contrôle sur la modélisation présente les points faibles de cette approche.

Il semble, par conséquent, qu'une approche hybride pourrait profiter des avantages de chacune. Le concepteur fournira les descriptions structurelles et comportementales des blocs dont le traitement par une méthode générique est impossible. Par exemple les blocs analogiques, tels que les amplificateurs différentiels ou les convertisseurs analogiques-numériques que l'on trouve couramment dans les circuits mixtes.

En plus, l'utilisateur aura la possibilité d'optimiser l'efficacité du désassemblage ainsi que la qualité de la modélisation en fonction de ses connaissances du circuit. Un gain important est possible lorsque le circuit contient des blocs de mémoires dont la structure de la cellule élémentaire est connue. Il s'agit d'une seule structure répétée. La bibliothèque est alors simple à construire et le gain d'autant plus important que la répétition est élevée.

2.5 Conclusion et objectifs du travail

Pour traiter des circuits de plusieurs dizaines de millions de transistors, les outils de CAO se heurtent à la difficulté de gérer une telle quantité de données. En même temps, le marché impose ses conditions dans lesquelles la conception et la vérification doivent se faire dans des délais de plus en plus courts. Dans un tel contexte, la conception d'un système intégré fait appel à de nombreux macro-blocs provenant de diverses origines.

La validation de ces blocs s'effectue individuellement, chacun dans ses propres conditions. Il est nécessaire d'avoir une approche de la vérification qui prend en compte la diversité des méthodes de conception. Les méthodes de vérification classiques sont adaptées à la conception par bibliothèque de cellules précaractérisée. Lorsque la conception sort de ce schéma, un seul recours reste possible : le traitement au niveau transistor.

Le désassemblage peut être vu comme une base sur laquelle s'appuient de nombreuses étapes du processus de la vérification, notamment la validation fonctionnelle, mais aussi l'analyse temporelle et l'évaluation de la consommation.

La question à laquelle nous tentons d'apporter une réponse dans cette thèse est de rendre le désassemblage d'un réseau de transistors et sa caractérisation fonctionnelle aussi automatiques et aussi transparentes que possible pour le concepteur. Comme nous le verrons au chapitre 3, il existe actuellement plusieurs méthodes d'abstraction fonctionnelle. Certains utilisent la reconnaissance de formes, d'autres plus évolués font appel à une approche générique.

Dans la première partie de ce travail, nous étudierons l'évolution des approches automatiques selon deux aspects. Premièrement, nous proposons une méthode d'identification des corrélations qui permet d'obtenir un temps de traitement linéaire avec la complexité du circuit pour l'algorithme de désassemblage. Deuxièmement, nous proposons une approche pour l'identification des éléments mémorisants.

Un autre volet de notre travail concerne la mise au point d'une approche hybride – reconnaissance de formes et méthode générique. Grâce à cette combinaison nous allons tenter d'allier la souplesse de la reconnaissance de formes à la facilité d'utilisation de la méthode générique.

Chapitre 3

ETAT DE L'ART

Dans ce chapitre nous présentons l'état de l'art dans le domaine du désassemblage et de l'abstraction fonctionnelle. Nous décrivons les premières approches, ensuite, nous étudions les évolutions et les outils phares pour chacune des deux techniques fondamentales. Nous soulignons les points forts de chaque outil ainsi que les aspects qui nous semblent perfectibles.

3.1 Introduction

Dans ce chapitre, nous traçons l'historique des approches de la vérification fonctionnelle fondées sur les techniques d'abstraction de réseaux de transistors. Nous commençons par présenter les premières approches apparues dans les années 80. Même si ces approches ne sont plus d'actualité, cette présentation nous permet de montrer l'évolution de la technique qui a débouché sur les méthodes utilisées aujourd'hui.

Ces méthodes peuvent être classées en deux catégories : la reconnaissance de formes, et les méthodes génériques d'analyse de réseaux. Nous illustrons ces deux approches par les réalisations logicielles les plus représentatives.

Nous relevons les caractéristiques de chaque méthode pour justifier le choix du point de départ de nos recherches, et pour clarifier les aspects qui nous semblent perfectibles et pour lesquels nous proposons des solutions dans les chapitres suivants.

3.2 Les premières méthodes

Les premières recherches sur l'abstraction fonctionnelle proprement dite portent sur l'accélération de la simulation au niveau interrupteur. En absence de méthodes d'abstraction, le seul moyen réaliste de valider le comportement final d'un circuit est de simuler le réseau de transistors issu de l'extraction du dessin des masques. Plusieurs méthodes de simulation basées sur cette modélisation interrupteur des transistors ont été proposées [Bry84] [Ree87] [Bar88].

A ce niveau, l'abstraction ne cible pas la modélisation complète du réseau de transistors. Il s'agit tout simplement de remplacer certaines structures de transistors par des modèles équivalents afin d'accélérer la simulation. Ces premières approches peuvent être classés en deux catégories. La première couvre les approches où il s'agit essentiellement d'identifier certaines structures pour leur associer un modèle prédéfini ou calculable. La deuxième catégorie regroupe les méthodes de réduction topologique qui traite les structures NMOS ou CMOS classiques.

3.2.1 Classification

Sous cette appellation sont regroupées les méthodes qui permettent d'identifier un ensemble défini de structures au sein du réseau de transistors qui représente le circuit. La détection de ces structures se fait, soit automatiquement, soit par un étiquetage fourni par le concepteur. Un mécanisme de génération de modèle comportemental simulable doit également être décrit pour chacune des structures.

On trouve une très vieille référence sur l'abstraction fonctionnelle dans [Szy73] : il ne s'agit pas encore d'accélérer une simulation au niveau transistor, mais de remplacer les structures en portes par des modèles fonctionnels plus abstraits. Un langage de description structurelle de blocs à reconnaître est défini. A chaque structure définie, est associé un modèle de comportement.

Ce n'est que bien plus tard, que l'on parle d'abstraction fonctionnelle à partir d'un réseau de transistors. Dans [Kaw82] une approche développée par une équipe de Toshiba est présentée. Il s'agit d'une méthode de partitionnement automatique. Le choix de nœuds et l'identification du sous-réseau de transistors qui définit la valeur émise sur le nœud se fait selon des critères simple et bien définis (Figure 3-1). Chaque partition est comparée avec un ensemble prédéfini de structures élémentaires, telles que des portes NAND ou NOR. En outre, une méthode de description de portes complexes à partir de ces éléments de base est également présentée dans cet article. Cette méthode permet par exemple de décrire des latchs et des bascules.

a) Sortie d'une porte CMOS b) Sortie d'une porte NMOS

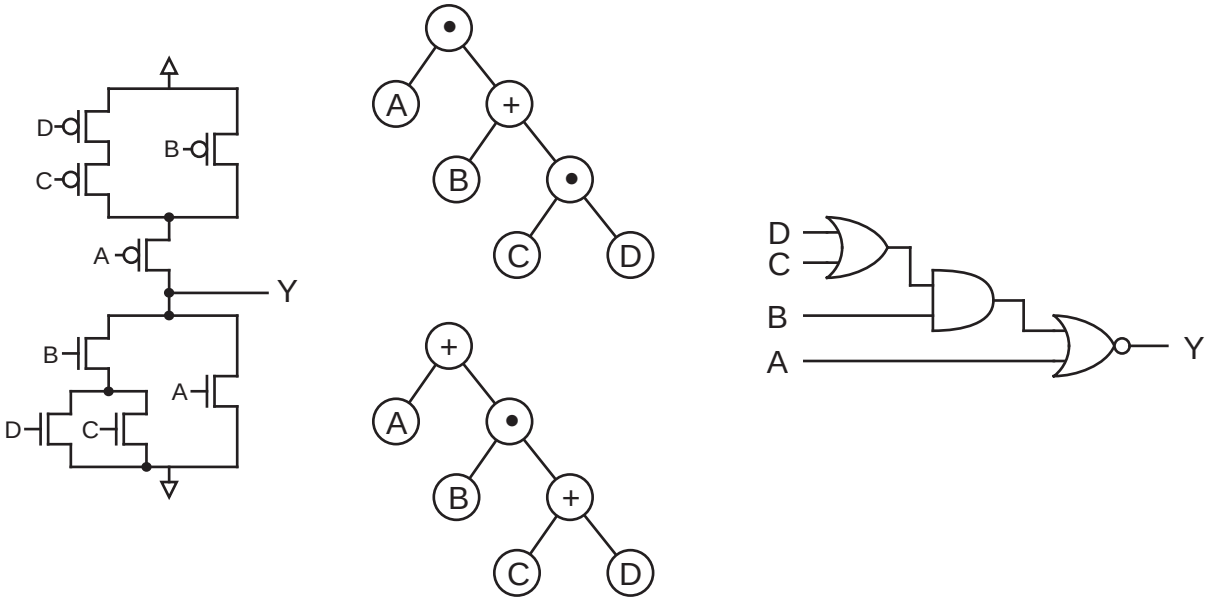
Figure 3-1 : *Critère de partitionnement*

L'intérêt de cette approche réside dans la technique de partitionnement automatique et dans l'introduction d'une méthode de description d'une bibliothèque de portes complexes. Cependant, l'ensemble prédéfini de portes élémentaires impose une limitation incontournable sur le nombre de structures reconnues. De plus, les éléments de la bibliothèque sont définis en fonction de ces portes élémentaires, ce qui limite les structures que l'on est capable de représenter.

3.2.2 Réduction

La réduction topologique, ou la réduction en série/parallèle, figure dans de nombreux articles. Au départ, elle a été élaborée pour les circuits NMOS [Apt82] [Saa82]. Par la suite, elle a été étendue aux circuits CMOS ; citons notamment le logiciel CTL [Tak82] de Toshiba et REDUCE [Bur83] de Hewlett Packard. Pour la clarté de l'exposé nous nous limitons ici au cas des circuits CMOS.

La première étape de la méthode consiste à identifier, dans le réseau de transistors, les nœuds pour lesquels on souhaite extraire un réseau de portes équivalentes. Dans le cas de la technologie CMOS, il s'agit de nœuds rattachés à la source d'un transistor de type 'N' et au drain d'un transistor de type 'P'.



a) Sous-réseau pilotant le nœud Y

b) Transformation en graphe

c) Transformation en portes

Figure 3-2 : *La réduction topologique en série/parallèle*

L'étape suivante consiste à isoler, pour chacun de ces nœuds, le sous-réseau de transistors qui pilote l'état du nœud. Dans la technologie CMOS, ce réseau se divise en deux sous-réseaux. Le premier sous-réseau est un ensemble de transistors qui se trouve sur un chemin de courant entre V_{dd} et le nœud, et le deuxième est un ensemble de transistors qui se trouve sur un chemin entre V_{ss} et le nœud. La Figure 3-2a donne l'exemple d'un circuit CMOS montrant clairement ce découpage.

A partir de ces deux réseaux, on construit deux graphes série/parallèle (Figure 3-2b). Les nœuds terminaux de ces graphes correspondent aux transistors du réseau. Ils sont étiquetés par le signal qui pilote la grille du transistor. Les nœuds non-terminaux de ce graphe représentent des opérateurs. On distingue deux types d'opérateurs : '•' correspond à une connexion en série de deux ensembles de transistors et '+' à une connexion en parallèle. Les arcs du graphe représentent alors les interconnexions de sous-réseaux de transistors. Si la structure constitue une porte CMOS duale, les deux graphes ainsi générés doivent être isomorphes et les étiquettes des nœuds doivent être identiques après l'inversion des '•' et '+' dans l'un des deux graphes. Dans ce cas, on procède à la phase de réduction.

La réduction s'effectue sur le graphe de transistors N par un simple parcours du graphe, en profondeur d'abord. Chaque nœud de type '•' est remplacé par une porte 'ET' et chaque nœud '+' par une porte OU. Les nœuds terminaux qui correspondent aux transistors N sont remplacés par le signal indiqué par l'étiquette du nœud. Enfin, un inverseur est ajouté sur le signal de sortie du réseau de portes ainsi généré (voir Figure 3.1c). D'une manière similaire, il est possible de générer une expression booléenne à partir du graphe.

Cette méthode est intéressante, car il s'agit d'une analyse automatique du réseau. La détection des structures sur laquelle la réduction s'applique, et leur modélisation, s'effectue sans informations supplémentaires de la part du concepteur. Cependant, l'approche se limite aux portes NMOS ou CMOS duales classiques. Il n'y a pas, a priori, de possibilité de traiter des structures plus complexes utilisant des transistors de passage. Une tentative de prendre en compte ce type de circuits est décrite dans [Kis83]. Mais la solution proposée ne permet de traiter que le cas le plus simple d'un transistor de passage à la sortie d'un réseau dual. Aussi, cette approche est encore loin de fournir une solution pour traiter convenablement un circuit réel dans son ensemble.

3.2.3 Conclusion

Dans ces approches préfigurent déjà les principales étapes de l'abstraction fonctionnelle : sélection des nœuds, partitionnement, modélisation. On y voit apparaître les difficultés de chaque étape et les premières réflexions sur les techniques de partitionnement et de modélisation automatique. Pourtant, l'ambition de ces approches s'est trouvée limitée par l'application visée. Elles ne cherchaient pas à effectuer un partitionnement exhaustif et

complètement automatique du circuit, mais se contentaient d'identifier des structures typiques.

En revanches, ces premières approches contenaient déjà deux notions bien distinctes : celle de la reconnaissance de structures définies, et celle de l'analyse automatique. Ces deux notions ont conduit à la réalisation de deux ensembles d'outils. Contrairement aux premières approches, ces outils visent la modélisation complète d'un circuit, c'est-à-dire, l'extraction d'une structure plus abstraite (un réseau de portes) à partir du réseau de transistors.

3.3 Reconnaissance de formes

3.3.1 Introduction

Dans cette section, nous exposons les méthodes basées sur la reconnaissance de formes. Nous commençons par la description de deux méthodes de base pour l'identification de sous-réseaux de transistors au sein du réseau qui représente le circuit entier. Ces méthodes nous intéressent dans la mesure où elles sont exploitables comme base pour une approche globale à la reconnaissance.

3.3.2 Identification des blocs fonctionnels

L'identification des blocs au sein d'un réseau de transistors ne correspond pas à un désassemblage proprement dit. Ici l'objectif n'est pas d'effectuer un partitionnement complet du circuit, mais d'isoler un composant ou un ensemble réduit de transistors.

Ces méthodes reflètent l'évolution des algorithmes de comparaison directe des réseaux de transistors. Elles tentent de faciliter cette comparaison en faisant une extraction (au moins partielle) d'une description en réseaux de portes.

3.3.2.1 BLEX

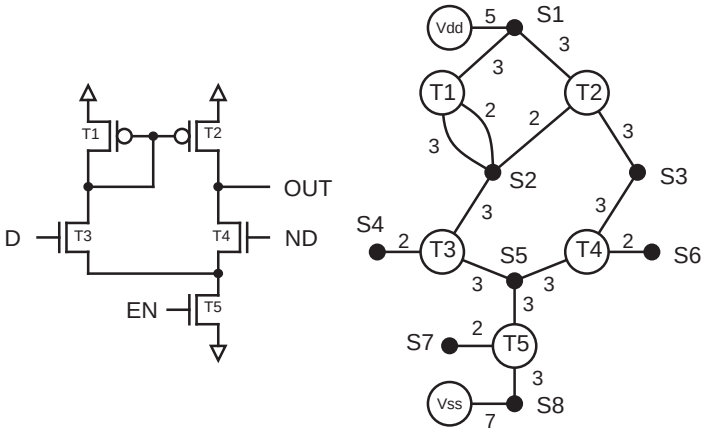
BLEX [Lue84] a été développé à l'université de Hanovre, dans le cadre d'un système de vérification physique. Ce système vise un partitionnement complet du circuit en blocs afin de permettre la vérification physique de celui-ci. Cette vérification passe par la comparaison de réseaux de portes ; il n'y a pas de modélisation haut niveau des blocs reconnus.

Un réseau de transistors est représenté par un graphe bipartite. Ce graphe comprend deux types de nœuds :

1) Les nœuds correspondant aux composants du circuit, appelés simplement nœuds dans la suite de cette section.

2) Les nœuds correspondant aux équipotentiels appelés araignées.

Un poids est attribué à chaque arc qui relie une araignée à un nœud. La valeur de ce poids dépend du type de connecteur (drain, grille, …) du nœud auquel est relié l'équipotentiel. Des nombres premiers sont choisis pour les poids. Ce graphe est, ensuite, transformé en une matrice d'incidence (Figure 3-3), où les lignes sont les nœuds et les colonnes les araignées. Chaque case de cette matrice contient le produit du poids des arcs qui relient le nœud et l'araignée correspondants. Puisque les poids des arcs sont des nombres premiers, la valeur de chaque case de la matrice a une factorisation unique qui caractérise le type de connexion.



a) Schéma d'un amplificateur différentiel

b) Le graphe bipartite

Araignées

Noeuds	S1	S2	S3	S4	S5	S6	S7	S8
Vdd	5	-	-	-	-	-	-	-
T1	3	6	-	-	-	-	-	-
T2	3	2	3	-	-	-	-	-
T3	-	3	-	2	3	-	-	-
T4	-	-	3	-	3	2	-	-
T5	-	-	-	-	3	-	2	3
Vss	-	-	-	-	-	-	-	7

c) La matrice d'incidence

Figure 3-3 : *Le graphe bipartite de BLEX avec sa matrice d'incidence*

Dans BLEX, une matrice est construite pour le circuit et une autre pour le bloc à reconnaître. L'identification du bloc est alors ramenée à la détection de la matrice d'incidence du bloc au sein de la matrice d'incidence du circuit. Il est possible d'isoler des blocs de matrice identiques à la matrice du bloc à reconnaître en modifiant l'ordre des lignes ou des colonnes dans la matrice du circuit. Bien évidemment, le nombre de permutations possibles rend cette approche directe inadéquate pour des circuits de taille réele. Ainsi, BLEX met en œuvre une optimisation de l'approche directe. Il effectue un partitionnement et un ordonnancement des matrices d'incidence, avant d'appliquer une méthode de comparaison progressive.

Cette méthode nous intéresse à cause de sa généralité. Aucune contrainte n'est imposée sur les structures à reconnaître. A priori, cette généralité lui permet même de fonctionner à

des niveaux hiérarchiques supérieurs au niveau transistor. Cependant, cette éventualité n'a pas été explorée. Les résultats obtenus après les optimisations sont bons, mais le temps d'exécution dépend directement de la taille du circuit et non de celle du bloc à reconnaître.

Par ailleurs, la technique d'étiquetage d'un arc selon un seul critère, à savoir le type de connecteur, est très efficace pour la reconnaissance des réseaux de composants asymétriques comme les transistors bipolars. Or, les circuits actuels sont principalement conçus autour des transistors MOS. Ce composant pose un problème particulier pour l'identification d'un sous-réseau : celui de la symétrie de la source et du drain. Cela implique que l'information sur le voisinage immédiat est insuffisante pour effectuer une identification rapide.

3.3.2.2 SubGemini

SubGemini tente de palier à ce problème en calculant à chaque point du réseau une information qui dépend de son voisinage plus ou moins éloigné. L'algorithme de SubGemini [Ohl93] formalise la tâche de recherche d'un sous-circuit dans un circuit comme un problème d'isomorphisme de sous-graphes. Il s'agit de l'extension d'une méthode de validation par comparaison directe de réseaux de transistors [Ebe88]. Cette première méthode avait fourni une solution pour vérifier l'isomorphisme de deux graphes, une tâche extrêmement complexe dans le cas général [Rea77]. Cependant, il existe de nombreux algorithmes de comparaison des graphes faisant partie des classes restreintes, tels que les graphes bipartites ou les graphes orientés et acycliques [Cor80].

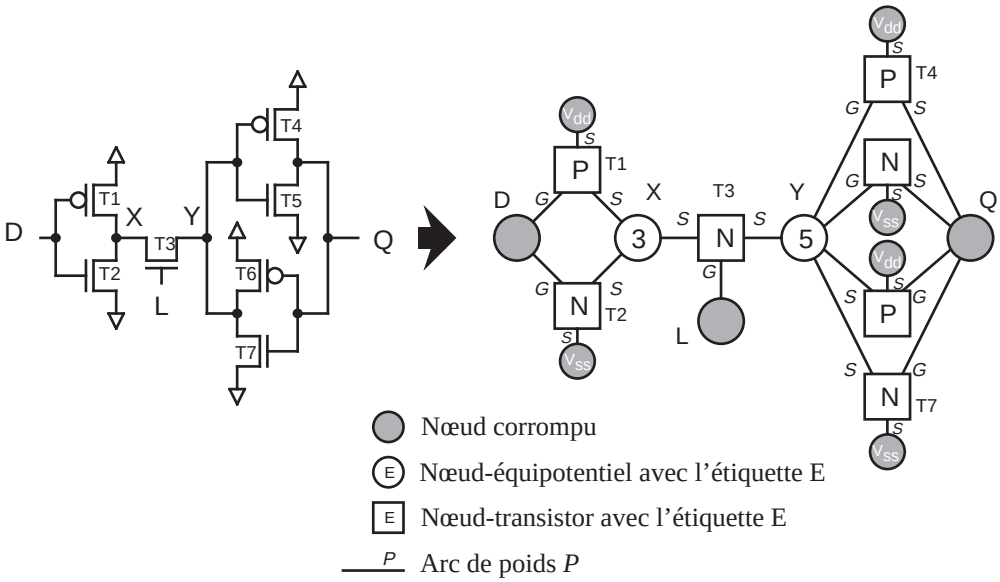


Figure 3-4 : *Initialisation du graphe bipartite dans SubGemini*

Dans SubGemini, le circuit est représenté par un graphe, et le réseau de transistors à reconnaître par un autre graphe. Il s'agit alors d'identifier dans le graphe du circuit un sous-graphe isomorphe au graphe du bloc à reconnaître. Comme dans BLEX, les deux graphes sont des graphes bipartites. Il existe deux types de nœuds : des nœuds correspondant aux transistors, et des nœuds correspondant aux équipotentiels. Les équipotentiels du réseau de transistors qui représentent les entrées et les sorties du réseau (les connecteurs du réseau) sont appelés des nœuds externes.

Comme dans BLEX, un poids est attribué à chaque arc qui relie un nœud-transistor à un nœud-équipotentiel. Ce poids dépend du type du connecteur de nœud-transistor auquel l'arc est relié. Chaque nœud porte une étiquette. Le principe de la méthode est celui d'un re-étiquetage itératif des nœuds simultanément dans les deux graphes.

La méthode mise en œuvre comporte deux phases. La première phase réalise deux opérations en parallèle. Premièrement, elle identifie, dans le graphe à reconnaître, un nœud dit nœud-clé qui porte une étiquette représentative du graphe. Intuitivement, un nœud-clé est un nœud choisi parmi les nœuds les plus internes (ou les plus éloignés de la périphérie) du graphe. Deuxièmement, elle isole, dans le graphe du circuit, des nœuds candidats susceptibles de correspondre au nœud-clé du graphe à reconnaître.

L'opération s'effectue de manière itérative en calculant à chaque itération une nouvelle étiquette pour chaque nœud. Les étiquettes des nœuds-transistors sont initialisées avec une valeur qui indique le type du transistor (N ou P). Les nœuds-équipotentiels sont initialisés avec le nombre d'arcs auxquels ils sont reliés (voir Figure 3-4). Les nœuds externes du graphe à reconnaître sont étiquetés « corrompues », car on ne connaît pas l'environnement du ceux-ci dans le graphe du circuit. A chaque itération, un re-étiquetage des nœuds est effectué dans le circuit et dans le réseau de transistors.

Si on note $E_i(n)$ l'étiquette de l'itération i d'un nœud n, l'étiquette de l'itération i+1 est calculé par :

$$\boxed{E_{i+1}(n) = E_i(n) + \sum_j P(n, v_j) \cdot E_i(v_j)}$$

où v_j représente un nœud voisin de n et $P(n,v_j)$ le poids de l'arc qui relie n à v_j. Cette opération est illustrée dans la Figure 3-5.

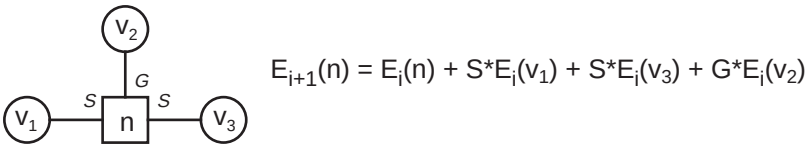


Figure 3-5 : *Re-étiquetage dans SubGemini phase 1*

Dans le cas où l'étiquette d'un des nœuds voisins indique que celui-ci est corrompu, alors $E_{i+1}(n)$ devient corrompue. Ainsi, à chaque itération, un nœud corrompu corrompt son voisinage.

La boucle d'itérations se termine juste avant qu'une nouvelle itération ait pour effet la corruption de tous les nœuds du graphe.

Le ré-étiquetage s'effectue en même temps dans le circuit et dans le réseau de transistors à reconnaître. De ce fait, à la fin de la boucle d'itérations, si un nœud du circuit porte la même étiquette qu'un nœud du réseau de transistors, il y a de grandes chances que le sous-réseau autour de ce nœud soit isomorphe au réseau de transistors. Ainsi, afin de limiter le champ de recherche, parmi les nœuds non-corrompus du réseau de transistors, le nœud dont l'étiquette est la moins fréquente dans le circuit est retenu comme nœud-clé.

Dans la deuxième phase, on tente de prouver l'isomorphisme du réseau de transistors avec un sous-réseau du circuit en partant du nœud-clé et d'un nœud du circuit, dit nœud-candidat, qui porte la même étiquette. On commence par attribuer la même unique étiquette au nœud-clé et au nœud-candidat. Cette étiquette restera la même tout au long des itérations. L'étiquette des autres nœuds-équipotentiel sont initialisés à zéro et celles des nœuds-transistors à une valeur qui indique leur type. Ensuite SubGemini effectue implicitement un parcours en largeur, parallèlement dans le circuit et le réseau de transistors. Ce parcours correspond à un re-étiquetage itératif, à partir du nœud-clé et du nœud-candidat dans le sens inverse de la première phase (Figure 3-6).

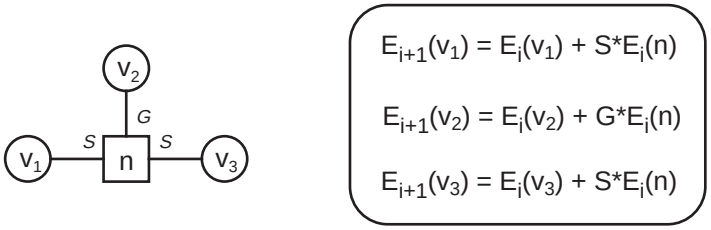


Figure 3-6 *: Re-étiquetage dans SubGemini phase 2*

Les étiquettes des voisins du nœud-clé et du nœud-candidat sont recalculées suivant le même algorithme que précédemment. Les nœuds ainsi re-étiquetés font recalculer les étiquettes de leurs voisins.

Afin de garantir qu'à la fin de ces itérations un nœud dans le réseau de transistors possède la même étiquette que son correspondant dans le circuit, il faut éviter l'utilisation des étiquettes des nœuds du circuit qui sont en-dehors de l'instance du réseau à reconnaître. A cet effet, SubGemini associe un niveau de confiance à chaque étiquette (sûr, non-sûr. C'est uniquement les nœuds qui possèdent des étiquettes « sûres » qui provoquent le re-étiquetage de leurs voisins. Une étiquette est considérée « sure » s'il existe autant de nœuds qui portent cette étiquette dans le circuit que dans le réseau à reconnaître.

L'ensemble des nœuds qui portent la même étiquette constitue une classe d'équivalence. Si, à la fin d'une phase de re-étiquetage, on obtient, dans le circuit, une classe d'équivalence contenant un seul nœud dont l'étiquette est « sûre », alors on considère que ce nœud correspond à l'unique nœud qui porte la même étiquette dans le réseau à reconnaître. On attribue alors une nouvelle étiquette unique et fixe à ces deux nœuds. La procédure de re-étiquetage s'arrête quand tous les nœuds du réseau à reconnaître ont des correspondants dans le circuit ou si aucun progrès n'est possible. Dans le premier cas, le sous-réseau autour du nœud-candidat et le réseau à reconnaître sont isomorphes. Dans le deuxième cas, la reconnaissance a échoué.

Toutefois, cet échec n'implique pas qu'il n'y a pas d'isomorphismes. Si le progrès du re-étiquetage n'est plus possible, SubGemini doit choisir de faire correspondre à un des nœuds du circuit un nœud dans le réseau de transistors qui porte la même étiquette avant de continuer à partir de celui-ci. Alors, si la reconnaissance échoue, il doit revenir et essayer une autre correspondance jusqu'à épuisement.

L'efficacité de cette approche provient du fait que le calcul de l'étiquette maximalise l'information du contexte d'un nœud. De plus, la complexité est plutôt en fonction du sous-graphe que du graphe. La première phase dépend de la taille du graphe, mais elle converge très vite, et le calcul de l'étiquette est peu coûteux. Le temps d'exécution de la deuxième phase est fonction de la taille du graphe, dans la mesure où le nombre de candidats est susceptible d'être plus élevé. Cependant, dans le cas d'un graphe fortement symétrique, la deuxième phase peut devenir très coûteuse.

3.3.2.3 Conclusion

Les deux méthodes présentées ont l'avantage principal de la généralité. Elles ne font pas de partitionnement préalable du circuit qui pourrait imposer des contraintes sur les structures à reconnaître. La description de ces structures étant faite avec des réseaux de transistors, elle autorise toute souplesse. Cependant, ces méthodes se heurtant au problème intrinsèquement complexe de prouver l'isomorphisme d'un sous-graphe et d'un graphe. De ce fait, elles se trouvent limitées à la reconnaissance d' objets simples.

Toutefois, leur souplesse leur permet une utilisation au sein de méthodes d'abstraction plus complètes. Les exemples de reconnaissance de formes, que nous allons maintenant étudier, s'appuie sur des méthodes similaires pour identifier les structures élémentaires d'un circuit. Ensuite ces structures sont utilisées pour aider à l'identification de structures plus complexes.

3.3.3 VERA

VERA [Kos91] est un outil qui vise un partitionnement complet d'un circuit. C'est un logiciel développé chez Philips pour la vérification structurelle. L'objectif est de reconstruire l'architecture structurelle à partir d'une base de données de structures élémentaires et de règles hiérarchiques.

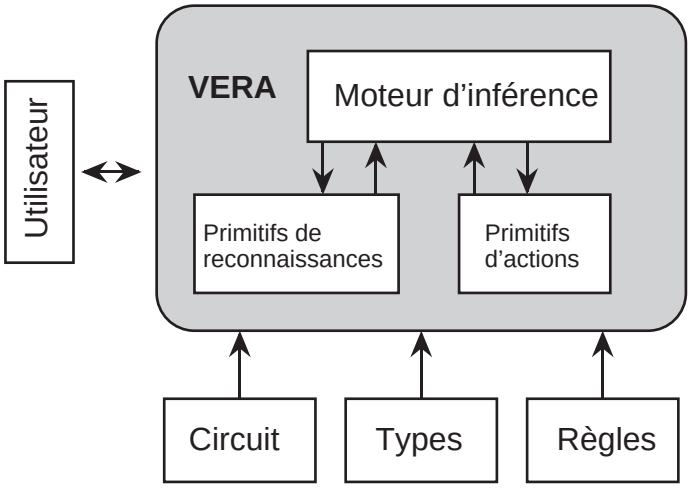


Figure 3-7 : *Architecture de VERA*

VERA est uniquement adapté à des méthodes de conception bien précises. Il a été utilisé pour la vérification des circuits conçus par des compilateurs de silicium, tels que PIRAMID [Hui88] ou CATHEDRAL [Gin86]. Ces compilateurs réalisent une synthèse des descriptions de haut niveau ou encore une synthèse d'architecture. Il est alors possible

d'élaborer des règles pour la reconstruction de l'architecture à partir du réseau synthétisé. Dans ce cas, la vérification effectuée par VERA est complète, dans la mesure où les règles de reconstruction sont correctes.

L'architecture logicielle de VERA est illustrée à la Figure 3-7. Le moteur d'inférence applique une méthode de la reconnaissance de formes. Il repère d'abord les structures élémentaires dans le circuit représenté par un réseau de transistors. Par la suite, des applications successives permettent la reconstruction hiérarchique de structures plus complexes. Pour chacune de ces structures, il doit exister un contrôleur qui définit sa connectivité sans ambiguïté. Une erreur de connectivité impliquerait nécessairement un échec de la reconstruction.

Les contrôleurs sont générés à partir de règles de reconstruction élaborées par l'utilisateur. Ces règles sont établies grâce à la connaissance précise de toutes les structures du circuit. Elles se composent toutes de deux clauses : une condition et une action, définies par un jeu de primitifs.

VERA distingue quatre classes de structures à reconstruire :

1) ***Structures non paramétrées***. Ce sont les structures pour lesquelles la description ne contient pas de variables, par exemple les éléments d'une bibliothèque de cellules (inverseur, NAND, NOR, etc.). Elles peuvent être décrites, soit en termes de transistors, soit en termes d'autres structures non paramétrées. Une règle pour la reconnaissance de telles structures utilise le primitif RECOGNIZE qui déclenche la recherche, et le primitif ABSTRACT qui remplace les structures identifiées par un élément abstrait.

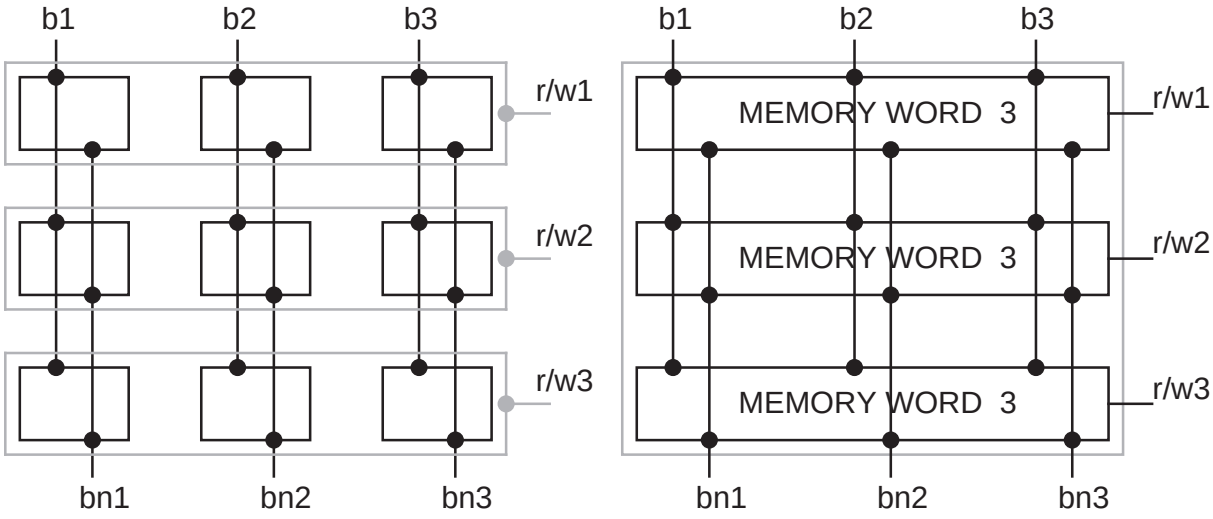
2) ***Structures à paramètre unique***. Ce sont des structures qui se composent d'une structure non paramétrée, répétée en série ou en parallèle, par exemple une colonne de cellules mémorisantes. Les primitifs CHAIN et FORK sont introduits pour la recherche de la répétition respectivement en série et en parallèle.

3) ***Structures à multiples paramètres***. C'est typiquement le cas d'un bloc de RAM dont la structure est définie à partir de deux paramètres : le nombre de bits et le nombre de mots. Cette classe est reconstruite par l'application hiérarchique des règles pour les

structures à paramètre unique. La Figure 3-8 présente l'exemple de la RAM et ses règles de reconstruction.

4) ***Structures paramétrées à fonctionnalité variable***. Ce sont des modules dont la structure ne se décompose pas en des répétitions mono-dimensionelles. Des primitifs dédiés à des ROMs et des PLAs sont définis ; ils reconstruisent une table de vérité pour la structure. L'éventualité d'autres structures dans cette classe n'a pas été prévue.

Le principal avantage de VERA consiste à sa bibliothèque de structures ouvertes et à son langage de règles hiérarchiques. Cela permet de reconstruire une architecture structurelle de très haut niveau à partir de réseaux de transistors utilisant des technologies diverses (NMOS, CMOS, BiCMOS, etc.).



a) regroupements des bits

b) regroupements des mots

```
RULE FORK-AND-ABSTRACT (abelt, n)(model, common)
  IF FORK (set, n, model, common)
    FIND-NAME (abelt, concat(model, n, '-))
  THEN ABSTRACT-FORK (abelt, set, model, common)
END
```

```
RULE FIND-MEM-CORE  (memcore, nbits, nwords)()
  IF FORK-AND-ABSTRACT  (word, nbits)('memorycell, 'r/w)
    FORK-AND-ABSTRACT  (memcore, nwords)
                        ( concat('forkmemorycell-, nbits),
                          r-l-a-i('(in out), nbits) )
  THEN MESSAGE  (memory core abstracted)
END
```

c) les règles de la reconstruction

Figure 3-8 : *La reconstruction d'un bloc de RAM avec VERA*

On reproche à VERA principalement son incapacité à traiter des circuits dont les structures internes sont, a priori, inconnues. La fonctionnalité de ce logiciel se limite à une comparaison structurelle entre le résultat obtenu avec un compilateur de silicium avec la description architecturelle d'origine fournie au compilateur. L'obstacle principal à une

utilisation plus générale de VERA est le fait que beaucoup de structures que l'on trouve couramment dans des circuits rentrent dans la quatrième classe, pour lesquelles on ne sait pas définir des primitives. Il est aussi regrettable que VERA ne permette pas une modélisation comportementale des structures reconnues, ce qui aurait rendu possible une vérification fonctionnelle.

3.3.4 GROG

L'abstraction fonctionnelle chez BULL à une longue histoire. La première version de l'outil d'abstraction, DESA, savait extraire, comme VERA, une représentation sous forme d'un réseau de blocs fonctionnels à partir du réseau de transistors. Cependant, son objectif était de générer une représentation logique formellement équivalente à la spécification du circuit.

Chaque élément à reconnaître était décrit par une règle qui exprimait la structure de l'élément en fonction d'une interconnexion de transistors MOS et d'éléments décrits par d'autres règles. Le moteur de reconnaissance distinguait trois types de structures : les portes CMOS duales, les blocs fonctionnels prédéfinis, et les structures à paramètre. Ainsi, DESA effectuait une abstraction hiérarchique, mais il fallait présenter les règles dans l'ordre de leur dépendance.

DESA avait été développé en PROLOG, et les règles de la bibliothèque étaient exprimées dans le logiciel comme des prédicats. Cette approche constituait l'un des inconvénients majeurs de DESA ; il était impossible de l'utiliser en dehors de BULL, car seul le concepteur de l'outil avait la possibilité d'adapter les règles à un nouvel ensemble de structures.

GROG [Gui98] est le successeur de DESA chez BULL. Son principal apport est de proposer une solution à la limitation majeure de DESA - celle de la bibliothèque fermée. Il intègre également la fonctionnalité de FAON et peut générer une description fonctionnelle. Il définit un langage général pour la description des règles. Ce langage permet à l'utilisateur d'adapter la reconnaissance à tous les styles de conception.

La méthode d'abstraction est plus généralisée, et ne distingue pas de classes particulières. Les structures à reconnaître sont des interconnexions de composants élémentaires (les primitives) et des structures déjà reconnues (les unités abstraites). Ainsi GROG retient les mêmes notions de règles hiérarchiques que DESA. La liste des primitives est prédéfinie, et

contient : les transistors MOS, les transistors bipolaires, les diodes, et les résistances. Cette liste est suffisamment complète pour couvrir toutes les technologies envisageables.

Toutes les règles ont la forme :

```
-> "nom de la règle" IF <modèle> THEN BUILD <unité abstraite>
```

Le modèle est une fonction des prédicats, où un prédicat est une expression de la connectivité d'un composant. Chaque unité abstraite, générée par une règle, peut devenir un prédicat dans une règle appliquée ultérieurement. Le système de règles permet ainsi une abstraction hiérarchique. Il est possible de décrire des structures paramétrables, telles que des multiplexeurs ayant un nombre d'entrées quelconque, par des règles récursives.

La fonction qui exprime le modèle à reconnaître est représentée par un arbre binaire, dont les nœuds sont les opérateurs {*et*, *ou*} et les terminaux sont les prédicats. L'opérateur *et* entre deux prédicats implique que les deux prédicats doivent être présents, avec une certaine connectivite, pour que le modèle soit reconnu. L'opérateur *ou* offre la possibilité de préciser des structures alternatives pour la même unité abstraite.

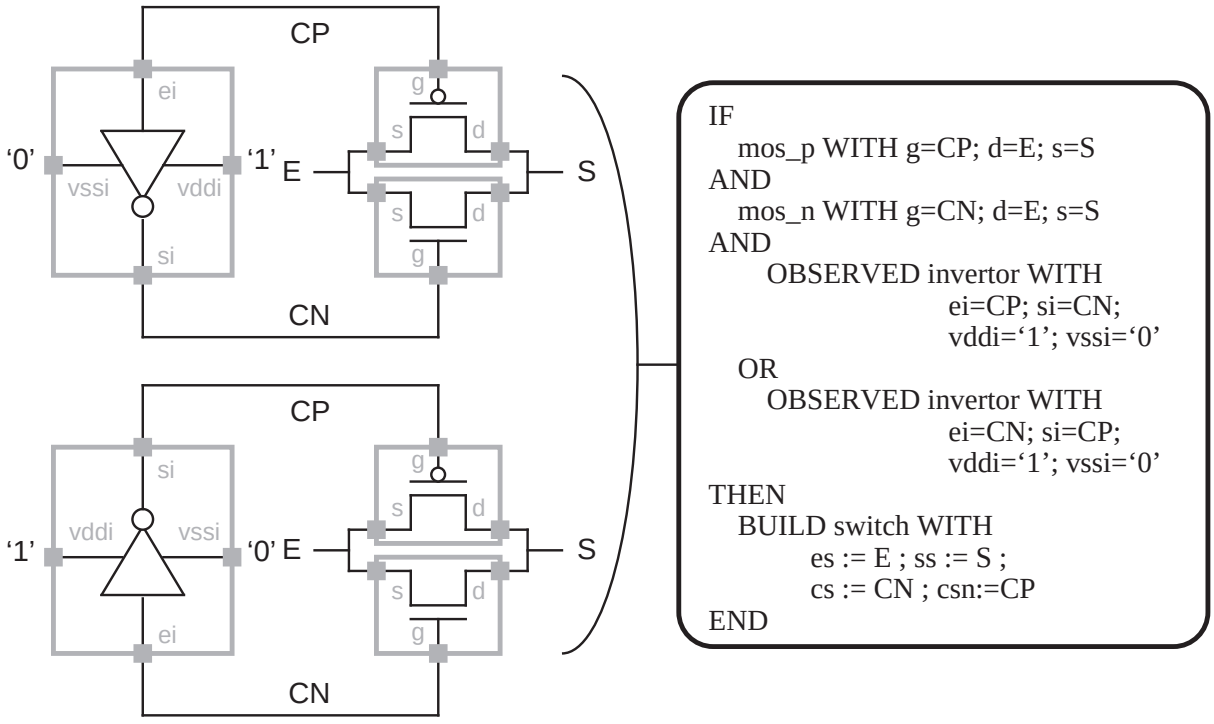


Figure 3-9 : *L'utilisation des opérateurs dans GROG*

Le partitionnement est strict, comme celui effectué par DESA. Cependant, il est possible de conditionner la reconnaissance d'un modèle par la présence ou l'absence de structures voisines. Cela s'exprime par un ensemble d'opérateurs unaires qui sont les suivants :

NOT : Vrai, si le composant précisé comme opérande est absent.

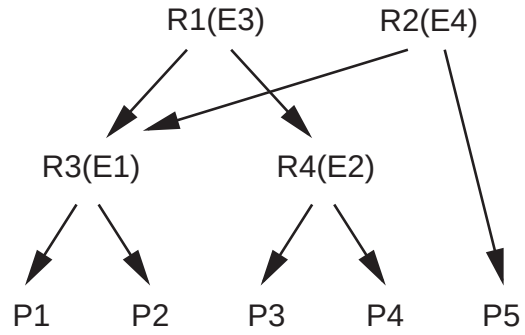
REDUCED : Vrai, si le nœud précisé comme opérande n'est connecté à rien d'autre qu'à des composants cités dans la règle. C'est un nœud strictement interne.

OBSERVED : Vrai, si le composant précisé comme opérande est présent. Cependant, le composant ne sera pas incorporé dans l'unité abstraite.

La Figure 3-9 montre un exemple d'utilisation des opérateurs binaires pour décrire deux structures possibles de la même unité abstraite. Il utilise également l'opérateur unaire *OBSERVED* pour permettre le partage de l'inverseur entre plusieurs « switches ».

Dans GROG, contrairement à DESA, les règles ne sont pas appliquées dans l'ordre de leur définition dans la bibliothèque. Etant donné que chaque unité abstraite correspond à une règle unique, GROG est capable d'identifier explicitement les dépendances entre les régles. Ces dépendances peuvent être exprimées par un graphe de dépendances. Un parcours « en profondeur d'abord » de cet arbre permet d'obtenir un ordonnancement des règles. La Figure 3-10 donne l'exemple d'un système de règles, et son ordonnancement à l'aide du graphe de dépendances.

```
"règle R1" if E1 and E2 then build E3
"règle R2" if E1 and P5 then build E4
"règle R3" if P1 and P2 then build E1
"règle R4" if P3 and P4 then build E2
```



Ri(Ej) signifie que la règle Ri construit l'unité abstraite Ej

Figure 3-10 : *Ordonnancement des règles dans GROG*

Dans GROG, il est possible d'associer à chaque règle un modèle fonctionnel en VHDL ou en Verilog. Le modèle sera associé à l'unité abstraite générée. Lors de la phase d'abstraction fonctionnelle, GROG rassemble les modèles associés de toutes les unités abstraites et, après substitution des noms des signaux, construit la description comportementale du circuit. Il est à noter que lorsqu'une règle *A* fait appel à une autre règle *B*, et si *A* a été reconnu dans le circuit, seul le modèle associé à *A* apparaîtra dans la description comportementale. Il n'est pas nécessaire d'ordonner cette description, car aussi bien VHDL que Verilog supportent les assignations concurrentes.

L'apport principal de GROG réside dans son langage de description de règles (GRL). Grâce à ce langage puissant, GROG a su répondre à la critique principale de DESA, celle de sa bibliothèque fermée. A l'aide de ce langage, le concepteur a la possibilité de définir ses propres règles d'abstraction. Cette ouverture a permis une utilisation de GROG dans de nombreux contextes industriels. De plus, la flexibilité du langage, alliée à une liste très large de primitifs élémentaires lui apporte l'indépendance technologique. Par exemple, il est possible de traiter des circuits numériques contenant des blocs analogiques, comme c'est le cas dans la technologie BiCMOS. La modélisation fonctionnelle de tous les types de composants est possible, car le texte de la description d'une unité abstraite est recopié directement, sans analyse sémantique.

Toutefois GROG souffre de l'utilisation de la méthode d'abstraction par reconnaissance de formes. Malgré la souplesse de GRL, l'élaboration et la validation d'une bibliothèque de règles est une tâche longue et complexe. Elle demande surtout une connaissance précise de toutes les structures utilisées. Il est même souhaitable d'avoir une connaissance du circuit, car les prédicats d'un modèle ne sont pas ordonnés. Les composants d'un modèle sont recherchés dans l'ordre dans lequel ils apparaissent dans la règle. L'ordre n'influence pas le résultat final, cependant il peut avoir un impact significatif sur la performance de la reconnaissance.

3.4 Les méthodes génériques

Les méthodes d'abstraction fonctionnelle génériques comportent deux étapes : une étape de partitionnement automatique du réseau de transistors, et une étape d'analyse des sous-réseaux ainsi obtenus. Les méthodes de réduction en série/parallèle représentent les premières tentatives d'analyse de sous-réseaux. Elles se limitaient au traitement des sous-réseaux duaux. Le principal obstacle à l'extension de ces méthodes à un sous-réseau quelconque est l'orientation des transistors.

Le problème de l'orientation avait déjà été abordé dans le contexte de la vérification temporelle [Ous85] et [Jou87]. Dans ces articles, les auteurs proposent un ensemble d'heuristiques permettant une orientation automatique d'un grand nombre de transistors. Dans cette section, nous étudions trois outils plus modernes qui abordent le problème de l'orientation dans le cadre de l'abstraction fonctionnelle.

3.4.1 LOGEX

LOGEX [Boe88] a été développé chez Siemens. Son objectif est d'obtenir un partitionnement complet du réseau de transistors sans l'aide d'une bibliothèque de cellules. Il a été conçu pour extraire un réseau de portes directement exploitable par un simulateur logique, y compris pour des circuits contenant des structures avec des transistors de passage.

LOGEX effectue un partitionnement en branches. Une branche est un chemin reliant deux équipotentielles, composée d'un ensemble de transistors connectés entre eux par leur source et leur drain. Une branche possède donc deux équipotentielles terminales. A l'initialisation, chaque branche contient un seul transistor, et chaque transistor appartient à une branche, et une seule. Par la suite, LOGEX effectue une réduction en série/parallèle en regroupant les branches d'après les deux règles suivantes :

1) Les branches qui partagent la même équipotentielle sur un de leurs terminaux sont combinées en série si cette équipotentielle n'est connectée à aucun autre transistor.

2) Les branches qui partagent le même nœud sur leurs deux terminaux sont inconditionnellement combinées en parallèle.

Ce partitionnement est illustre dans la Figure 3-11. Notons que le partitionnement est exhaustif, car dans le pire des cas, une branche contient un seul transistor.

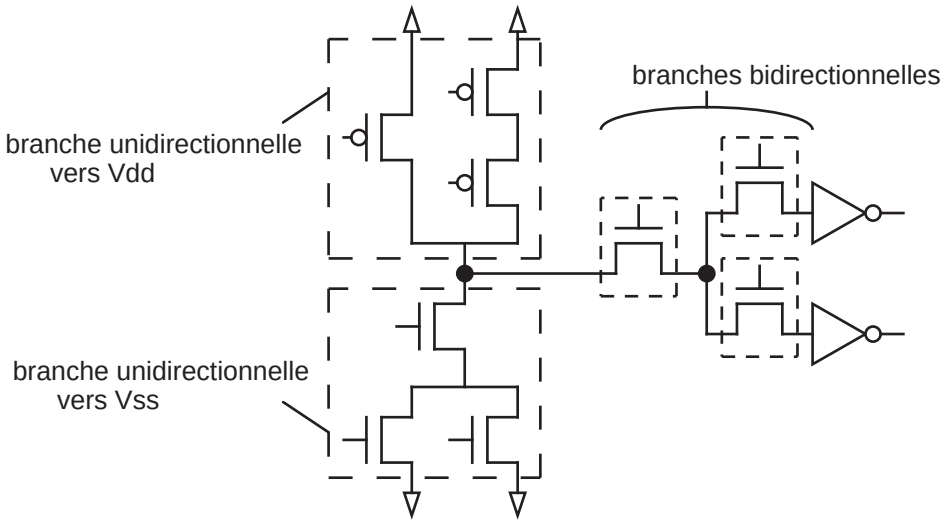


Figure 3-11 : *Partition et classification des branches dans LOGEX*

Après la réduction, les branches ainsi obtenues sont classées selon trois types :

Type 1 : Les branches unidirectionnelles vers V_{dd}.

Type 2 : Les branches unidirectionnelles vers V_{ss}.

Type 3 : Les branches bidirectionnelles entre deux nœuds internes.

A la suite de cette étape de partitionnement, une caractérisation logique est effectuée pour chaque branche. Cette caractérisation est une étape intermédiaire pour obtenir l'expression logique des nœuds. Elle consiste à établir une représentation équivalant au graphe série/parallèle. Cette représentation caractérise uniquement la structure du graphe, sans identifier les équipotentielles qui le conditionnent.

Après la caractérisation de chaque branche, les nœuds correspondant à la sortie des portes CMOS duales sont repérés. LOGEX recherche les nœuds auxquels sont connectés une branche V_{dd} (type 1) et une branche V_{ss} (type 2). Si les caractérisations sont complémentaires et les grilles de chacune de ces branches sont connectées aux mêmes nœuds, la branche V_{dd} est supprimée, et la branche V_{ss} reçoit un nouveau type (porte CMOS) avec la même caractérisation fonctionnelle.

Il reste l'étape critique de l'orientation des branches de type 3. LOGEX met en œuvre deux règles universelles :

Règle 1 : Les terminaux des branches de type 3 connectes aux sorties des portes CMOS sont obligatoirement des entrées de ces branches (Figure 3-12a).

Règle 2 : Un terminal d'une branche de type 3, connecté à au moins une entrée d'une porte CMOS ou à la grille d'un transistor d'une autre branche, est obligatoirement l'entrée de cette branche, si il n'y a pas d'autres branches bidirectionnelle connectées aux mêmes nœuds (Figure 3-12b).

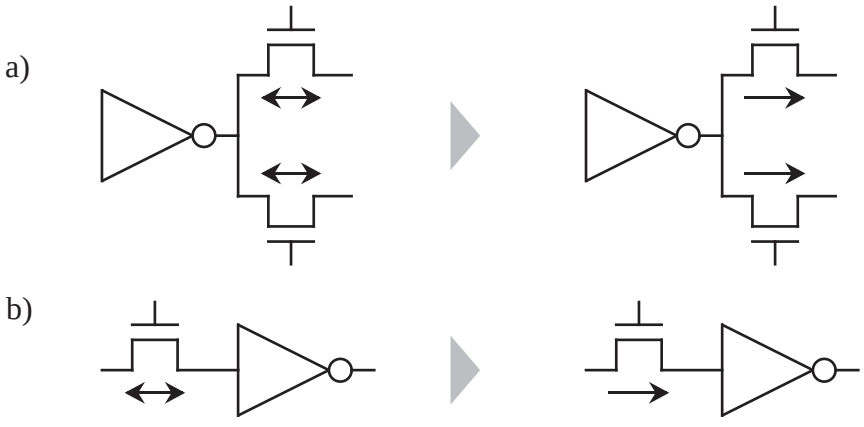


Figure 3-12 : *Les règles universelles de LOGEX*

Cependant, même après l'application de ces deux règles universelles, il peut rester de nombreuses branches non-orientées. Dans ces cas, des règles supplémentaires peuvent être activées. Ces règles sont basées sur des heuristiques de classement de puissance des terminaux, selon le type et l'orientation des branches.

L'apport significatif de LOGEX repose sur son intégration de l'approche de réduction en série/parallèle avec des techniques d'orientation des transistors. Cette combinaison lui permet d'effectuer un partitionnement complet avec une caractérisation fonctionnelle adéquate dans un bon nombre de cas.

Cependant, on constate qu'une méthode d'orientation par des heuristiques ne sera jamais exhaustive. Ces heuristiques ne sont pas établies à partir des règles formelles, mais représentent plutôt des styles de conception. De ce fait, des structures complexes avec des transistors de passage ou des structures comportant des transistors bidirectionnels ne sont pas correctement caractérisées. En outre, les heuristiques font l'hypothèse de la validité du circuit en cours de vérification.

3.4.2 ANAMOS

ANAMOS [Bry87] a été développé à l'Université Carnegie Mellon dans le cadre des recherches sur la simulation des circuits MOS. Il a été utilisé comme un pré-processeur pour le simulateur COSMOS (Compiled Simulator for MOS Circuits). A partir du réseau de transistors, ANAMOS fabrique une représentation booléenne. Cette représentation est ensuite traduite en un ensemble de fonctions d'évaluations, qui sont compilées avec un noyau de simulation pour générer un simulateur exécutable. ANAMOS réalise donc deux tâches. D'abord, il effectue un partitionnement du circuit. Puis il fournit une représentation Booléenne pour chaque partie.

ANAMOS partitionne le circuit en CCC (channel connected components). Ce partitionnement est effectué à partir de la règle suivante :

Règle : Deux transistors sont dans le même CCC s'ils sont connectés ensemble par leur source ou leur drain.

Un CCC est donc constitué de transistors. Il contient également les équipotentielles, reliées aux sources et aux drains de ceux-ci. Les équipotentielles peuvent être de type « entrée » ou « mémorisant ». Les équipotentielles reliées à des connecteurs externes sont de type « entrée ». Toutes les autres équipotentielles sont de type « mémorisant ».

La tâche de modélisation est basée sur l'identification des chemins qui influencent l'état d'un nœud et sur la comparaison de la « force relative » de chaque chemin. Chaque nœud du circuit est caractérisé par un poids appelé taille (*size*) qui modélise de manière simplifiée la

capacité de mémorisation de ce nœud. L'ensemble des valeurs qui peut prendre ce poids est un ensemble de valeurs discrètes, prédéfinies et normalisées {*1, …, k, w*}. *w* est le poids attribué aux nœuds externes et sa valeur est telle que $w > k$. A l'aide de ces poids, il est possible de modéliser les phénomènes de partage de charge et de déterminer la valeur d'un nœud sans avoir recours à la définition d'une orientation pour les transistors.

Par ailleurs, un poids appelé force (*strength*) est attribué à chaque transistor. L'ensemble des valeurs que peut ce poids est un ensemble de valeurs discrètes, prédéfinies et normalisées { *k+1*, …, *w-1*}. Ce poids modélise de manière simplifiée la conductivité du transistor. Ainsi, pour chaque chemin *c* qui influence l'état d'un nœud, il est possible de déterminer un poids *p* caractéristique du nœud source de ce chemin et de la résistance des transistors qui constituent le chemin. Le poids *p* d'un chemin *c* est défini par la formule :

$$p = \min\{taille(source(c)), \min_{t \in c}\{force(t)\}\}$$

où *source*(*c*) est le nœud origine du chemin *c* et *t* un transistor appartenant au chemin.

Cette caractérisation des chemins permet de les classer et donc d'établir l'expression booléenne de chaque nœud en définissant l'ordre de priorité de chaque chemin.

L'extraction d'un modèle comportemental est basée sur une méthode d'analyse symbolique du réseau de transistors, en s'appuyant sur un modèle interrupteur. Dans ce modèle, un nœud est une variable à trois états $N \in \{0, 1, X\}$, ou X correspond à une valeur logique indéterminée sur un nœud non initialisé ou conflictuel. Pour permettre des opérations booléennes, l'état d'un nœud est codé par deux variables booléennes selon Figure 3-13.

N	N^1	N^0
0	1	0
1	0	1
X	1	1

Figure 3-13 : *Encodage des variables à trois états*

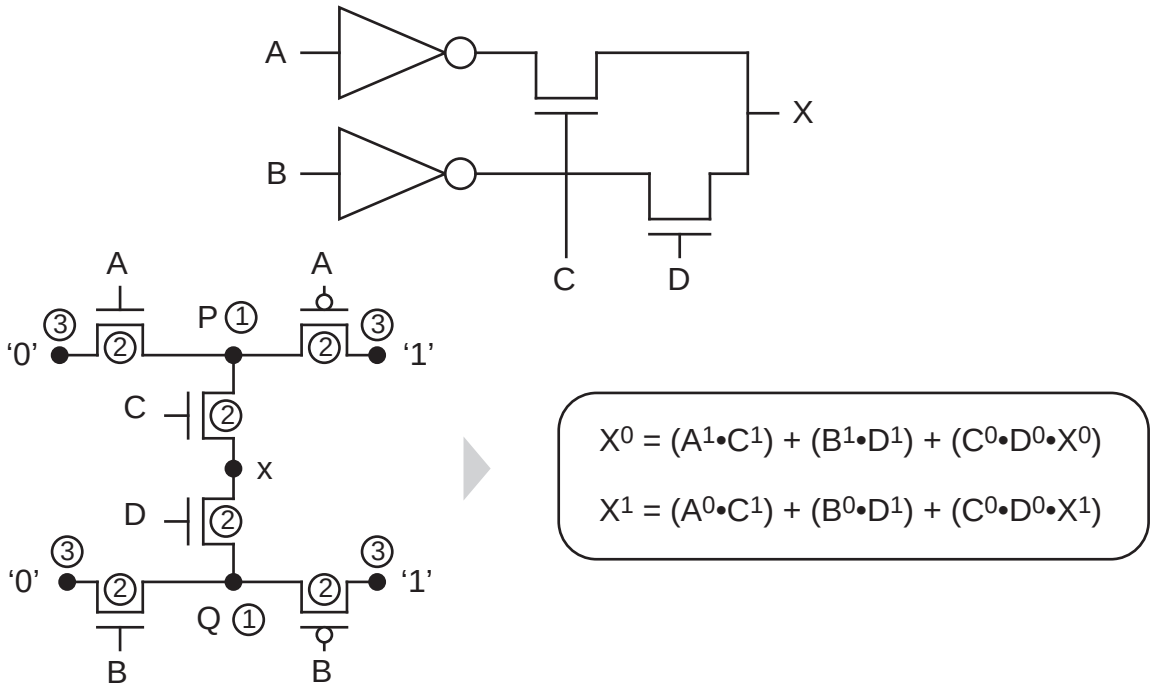
Pour chaque poids *s* et nœud *n*, des expressions N^1_s et N^0_s sont calculées. Ces expressions représentent la réponse statique du nœud *n* aux chemins dont le poids est supérieur ou égal à *s*. Elles sont obtenues par la solution de deux systèmes d'équations booléennes. Ces systèmes sont initialisés avec N^1_{s+1} et N^0_{s+1}, les chemins de poids *s*, et les conditions pour

lesquelles il n'y a pas de chemins actifs de poids supérieur à *s*. Le dernier terme est obtenu pour chaque nœud par la solution d'un troisième système d'équations. C'est alors un algorithme itératif, à partir du poids *s* le plus élevé, pour obtenir le N^1_I et le N^0_I de chaque nœud *n*. Ces deux termes sont équivalents à la réponse statique due à tous les chemins (N^1 et N^0). La Figure 3-14 montre le résultat de l'analyse par ANAMOS sur le nœud de sortie d'un multiplexeur à deux entrées.

Les réponses statiques des nœuds internes sont aussi calculées par ANAMOS, cependant ces nœuds sont supprimés dans une phase d'optimisation par la règle suivante :

Règle : Un nœud est supprimé s'il n'est pas directement observables à l'extérieur du CCC, et leur charge n'influence pas les autres nœuds.

L'exemple de la Figure 3-14 illustre deux particularités de la méthode. Premièrement, l'analyse de chaque CCC se fait de manière isolée et en dehors de son environnement. En particulier, les corrélations entre les entrées d'un CCC sont ignorées. Ici, c'est l'inversion des entrées *c* et *d* qui n'est pas prise en compte. Cela se traduit par le fait que les expressions booléennes codent la possibilité que le nœud soit conflictuel.



ⓝ signifie un nœud de force n ou un transistor de taille n

Figure 3-14 : *Analyse d'un multiplexeur par ANAMOS*

La deuxième particularité est la notion de mémorisation des nœuds. En absence de chemins actifs, on considère qu'un nœud retient sa valeur initiale. Dans l'exemple, on voit

que les deux expressions du nœud *X* possèdent un terme qui correspond à l'état initial du nœud. Cette approche a l'avantage de permettre le traitement des mémoires dynamiques. Cependant, elle rend plus difficile la détection d'une erreur due à l'absence d'emetteurs pour un nœud (par exemple si *C* et *D* ont l'état '0' dans la Figure 3-14). Dans ce cas, la haute impédance n'est pas signalée au cours d'une simulation, est une valeur erronée peut être propagée.

ANAMOS présente une approche puissante, car le partitionnement en CCC est complet, et la modélisation au niveau interrupteur est parfaitement généralisée. Toutefois, la règle de partitionnement construit dans certains cas des CCC contenant plusieurs milliers de transistors. Ce fait, lié avec l'ignorance des corrélations peut conduire à des systèmes d'équations très coûteux à résoudre. De plus, les expressions booléennes obtenues sans prendre en compte ces corrélations sont souvent sous-optimales. Ces problèmes se manifestent en particulier sur les décaleurs construit à partir d'un réseau de transistors de passage (Figure 3-15).

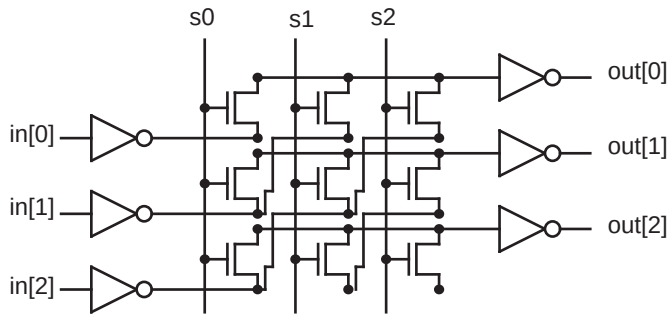


Figure 3-15 : *La structure du décaleur*

Enfin, les points mémorisants statiques ne sont pas explicitement repérés. En fait, ils sont correctement modélisés, mais c'est seulement au cours de la simulation que le comportement mémorisant d'un nœud se manifeste.

3.4.3 DESB

Le logiciel DESB [Lau94] a été développé au Laboratoire LIP6. Il met en œuvre une méthode de pseudo partitionnement et de caractérisation automatique pour les circuits CMOS.

Dans l'étape de partitionnement, le circuit est découpé en un ensemble d'objets que l'on appelle des cônes. Un cône distinct est fabriqué pour chaque nœud du circuit qui attaque une ou plusieurs grilles de transistor. Un cône se compose d'un ensemble de branches. Une branche est un chemin vers des sources de tension constante (des alimentations) ou vers des

connecteurs externes, à travers des canaux des transistors. De ce fait, ils ressemblent aux CCC de ANAMOS dans la mesure où l'objet représente les chemins de courrant qui influencent l'état d'un nœud. Cependant, à la différence de ANAMOS, DESB n'effectue pas un partitionnement strict et un transistor peut faire partie de plusieurs cônes.

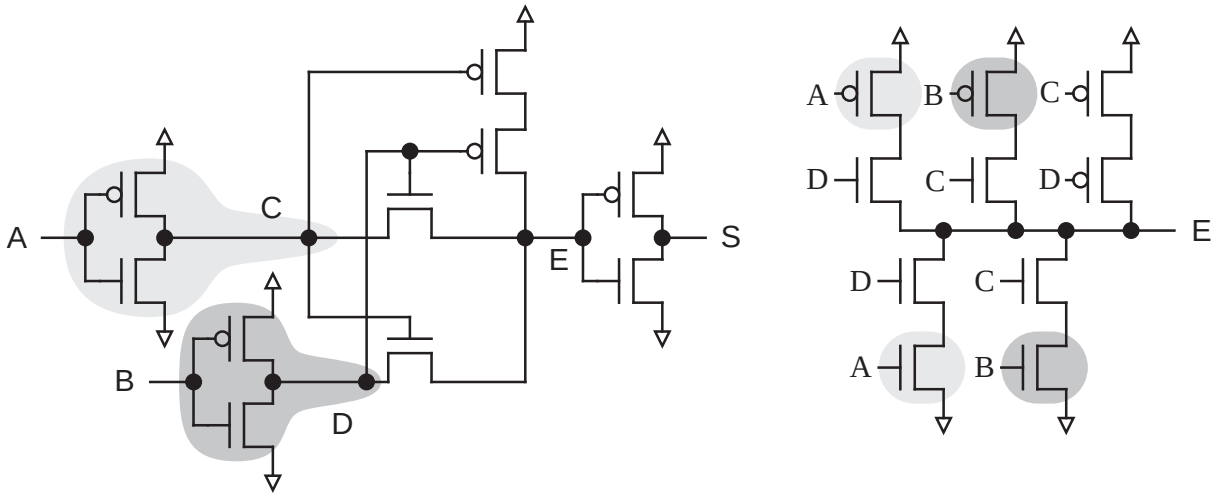
La Figure 3-16 montre bien ce découpage recouvrant sur une porte ou-exclusive. Quatre cônes sont construits pour ce réseau de transistors. Les transistors des cônes C et D sont aussi dans le cône E. Pour éviter la construction des branches dont les contributions fonctionnelles ne sont pas significatives, trois principes sont respectés.

Principe 1 : Une branche ne passe jamais deux fois par le même signal.

Principe 2 : Une branche ne contient jamais deux transistors de types différents pilotés par le même signal.

Principe 3 : En parcourant une branche dans un sens quelconque, il ne doit pas être possible de rencontrer sur un drain (ou une source) un signal qui attaque la grille d'un transistor préalablement parcouru.

Le troisième principe est mis en œuvre dans l'exemple de la Figure 3-16. Elle a pour effet d'éviter que les deux cônes C et D ne contiennent tous les transistors du cône E.



a) Le réseau de transistors

b) Le cône principal avec transistors partagés

Figure 3-16 : *Le découpage rcouvrant de DESB*

A la suite de la fabrication des cônes, deux expressions de la fonctionnalité de chaque cône sont générées. Ces deux expressions représentent les conditions de mise à un et de mises à zéro du cône. Pour chaque paire d'expressions, les propriétés d'orthogonalité et de complétude sont vérifiées, afin d'établir si un cône correspond à une porte CMOS DUAL

Toutefois, le respect des trois principes cités plus haut ne garantit pas l'élimination de toutes les branches non fonctionnelles. Il est possible qu'un cône, qui aurait dû être reconnu comme une porte CMOS DUAL, soit marquée conflictuel. Cela est dû à la construction de fausses branches. Il s'agit de branches qui satisfont les trois principes, mais qui ne sont jamais passantes à cause des corrélations entre les signaux qui contrôlent les transistors de la branche.

La Figure 3-17 donne un exemple de construction de ces fausses branches. Le cône *E* devrait être un simple inverseur contrôlé par *B*, mais DESB construit deux branches supplémentaires. On constante que la suppression des fausses branches est un problème identique au problème de l'orientation des transistors.

La solution adoptée par DESB est une phase d'analyse fonctionnelle globale après la fabrication de tous les cônes. C'est un algorithme itératif qui examine chaque cône, en élimine les fausses branches détectables, avant de passer au cône suivant. Les itérations se poursuivent jusqu'à ce qu'aucune nouvelle fausse branche ne soit détectée sur aucun cône. Il est nécessaire de réexaminer les cônes de manière itérative, car l'élimination de fausses branches dans un cône peut modifier son statut (devenant non-conflictuel). Cette modification peut alors avoir des répercussions sur les autres cônes. L'analyse est effectuée pour chaque cône dont les expressions ne vérifient pas les deux propriétés d'orthogonalité et de complétude.

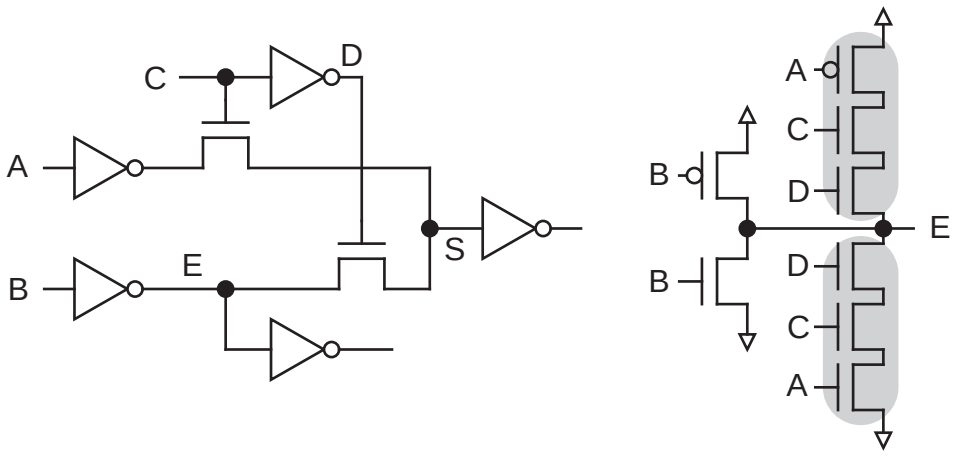


Figure 3-17 : *Des fausses branches dans DESB*

En résumé, l'analyse des fausses branches passe par l'établissement de la condition de conductance des branches. Pour établir cette condition, il est nécessaire de tenir compte des corrélations entre les signaux qui pilotent la grille des transistors qui composent une branche.

Pour ce faire, cette condition de conductance est exprimée en fonction d'un ensemble de variables dites primaires qui doit posséder les deux propriétés suivantes :

1) Il doit être possible de re-exprimer la fonctionnalité du cône uniquement en termes de ces variables.

2) Les variables primaires doivent être indépendantes (non-corrélées)

DESB est un outil puissant qui a fait ses preuves sur de nombreux circuits industriels. Le partitionnement en cônes est particulièrement puissant car, comme pour les CCC, il n'y a pas de transfert de courant entre les cônes. Cette caractéristique permet de générer, pour chaque cône, un modèle indépendamment des autres cônes. Mais, contrairement aux CCC, la taille des cônes, après l'élimination des fausses branches, reste bien limitée. Un autre avantage de DESB est de modéliser un nombre réduit de nœuds. A la différence de ANAMOS, dans DESB, un cône est construit pour chaque nœud dont l'état conditionne l'état d'un autre nœud. On ne cherche pas à établir des expressions pour des nœuds internes à un cône.

Par ailleurs, la méthode utilisée par DESB pour déterminer les variables primaires n'est pas satisfaisante et ne garantit pas l'obtention d'un nombre minimal de variables indépendantes.

De plus, la séparation des phases de construction des cônes et de l'analyse fonctionnelle peut créer temporairement des cônes très complexes à cause des fausses branches. Dans certains cas, la construction de ces fausses branches peut aboutir à une forte consommation de mémoire. De plus, l'analyse de ces cônes peut s'avérer par la suite très coûteuse ou même prohibitive.

Enfin, la reconnaissance et la modélisation des points mémorisants est le principal point faible de DESB. En effet, seul un nombre limité de formes peuvent être traitées. Chaque boucle entre deux cônes est comparée avec un ensemble de structures. Ces structures sont prédéfinies dans le logiciel et non-modifiables. L'approche est efficace mais peu générale. Enfin, DESB est strictement réservé aux circuits CMOS numérique. Il n'est pas capable de modéliser correctement les circuits contenants des composants analogiques.

3.5 Conclusion

Nous avons étudié en détail plusieurs implémentations des deux principales approches à l'abstraction fonctionnelle : la reconnaissance de formes et les méthodes génériques. Une approche entièrement par reconnaissance n'est pas envisageable pour plusieurs raisons.

Premièrement, la nécessité de décrire toutes les structures utilisées dans un circuit oblige une connaissance approfondie du circuit, qui n'est pas toujours disponible. Cela est en contradiction avec une des motivations importantes de l'abstraction fonctionnelle : la réutilisation des anciens circuits par la migration technologique. Deuxièmement, l'élaboration et la validation d'une bibliothèque de règles sont des taches complexes. En effet, le temps et l'expertise nécessaires ne sont souvent pas disponibles.

C'est pour ces raisons que nous ayons choisi de développer une approche basée sur les méthodes génériques d'analyse de réseau. Comme nous avons vu, l'outil DESB représente l'état de l'art actuel de cette approche. Cela est du a deux raisons : premièrement le concept original de partitionnement du réseau de transistors en objets recouvrants, deuxièmement la reconnaissance que le problème de l'orientation des transistors ne peut pas être résolu en ignorant la corrélation des signaux.

Notre travail consiste à adresser les deux problèmes principaux que nous avons identifiés comme étant mal traités par DESB. Le premier problème est celui de la séparation de l'analyse fonctionnelle et la phase de construction des cônes, alors que cette analyse demeure essentielle pour un partitionnement efficace. Le deuxième problème est le manque de généralité dans le traitement des éléments mémorisants.

Cependant, nous retenons certains aspects intéressants de la reconnaissance de formes. La possibilité de configurer les outils, permet leur adaptation à diverse technologies. Malgré la prépondérance de CMOS, il devient de plus en plus courant de fabriquer des circuits de type mixte numérique/analogique. Cela ne devrait pas être bloquant pour un outil d'abstraction fonctionnelle. Enfin, cette approche donne la possibilité à l'utilisateur d'intervenir dans la phase de modélisation.

Chapitre 4

METHODE DE DESASSEMBLAGE ET D'ABSTRACTION FONCTIONNELLE

Dans ce chapitre, nous présentons le cœur de notre approche de l'abstraction fonctionnelle. Nous montrons le modèle de partitionnement sur lequel nous nous reposons, ainsi que les problèmes posés par le partitionnement. Ensuite, nous nous consacrons à détailler les solutions que nous avons adoptées pour la résolution de ces difficultés.

4.1 Introduction

Comme nous l'avons vu au chapitre précédent, les techniques génériques présentent l'avantage de ne pas nécessiter de bibliothèque de formes prédéfinies pour réaliser le désassemblage d'un circuit.

Parmi les outils qui mettent en œuvre ce type d'approches, DESB, développé au laboratoire LIP6, utilise une méthode de pseudo partitionnement qui autorise le partage des transistors entre plusieurs cônes. De ce fait, il aboutit à des structures beaucoup plus petites que ANAMOS.

C'est pour cette raison que nous avons décidé de bâtir notre approche en partant des concepts et des méthodes proposés par DESB.

Dans ce chapitre, nous commençons donc par rappeler le modèle de cônes de DESB, puis nous présentons les améliorations que nous avons amené pour limiter la fabrication des branches non-fonctionnelles.

4.2 Modèle de cônes

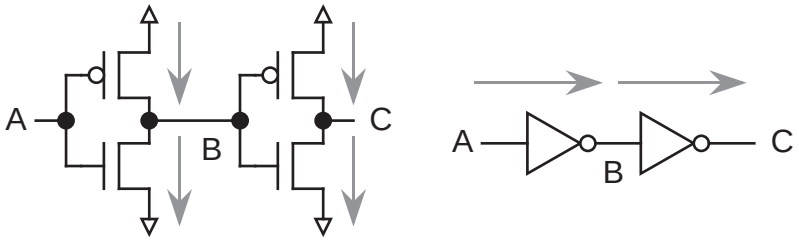
Le cône constitue l'objet de base de la méthode adoptée. Le désassemblage se résume à la conversion d'un réseau de transistors non-orientés en un réseau de cônes orientés. Dans cette section, nous détaillons les caractéristiques de cet objet et nous exposons la méthode de partitionnement.

4.2.1 Règles de partitionnement

Le point de départ de toute stratégie de partitionnement est d'identifier les nœuds pour lesquels on souhaite extraire un sous-réseau. Puisque DESB a été conçu pour les circuits utilisant la technologie CMOS, cette contrainte se traduit par le fait que l'interface entre deux cônes est un signal qui contrôle la grille d'un transistor. Ainsi, les nœuds pour lesquels un sous-réseau est extrait, lors du partitionnement, sont les nœuds qui attaquent au moins une grille de transistor (Figure 4-1).

Une autre implication de cette contrainte est que pour un tel nœud tous les chemins de conduction doivent faire partie de son cône. Autrement dit un cône est un objet DC-connexe :

il contient tous les chemins qui relient le nœud à une source de tension à travers les canaux (connexion source-drain) des transistors. On considère comme source de tension, les connecteurs d'alimentation du circuit.



a) Le courant ne traverse pas les grilles b) L'information traverse les grilles

Figure 4-1 : *Flot de courant et flot de l'information*

Ainsi chaque cône possède une sortie unique et un certain nombre d'entrées qui sont les signaux qui contrôlent les grilles des transistors qui composent le cône. Dans certains cas (lorsqu'un chemin aboutit à un connecteur externe) l'entrée d'un cône peut être un connecteur externe relié à un drain de transistor.

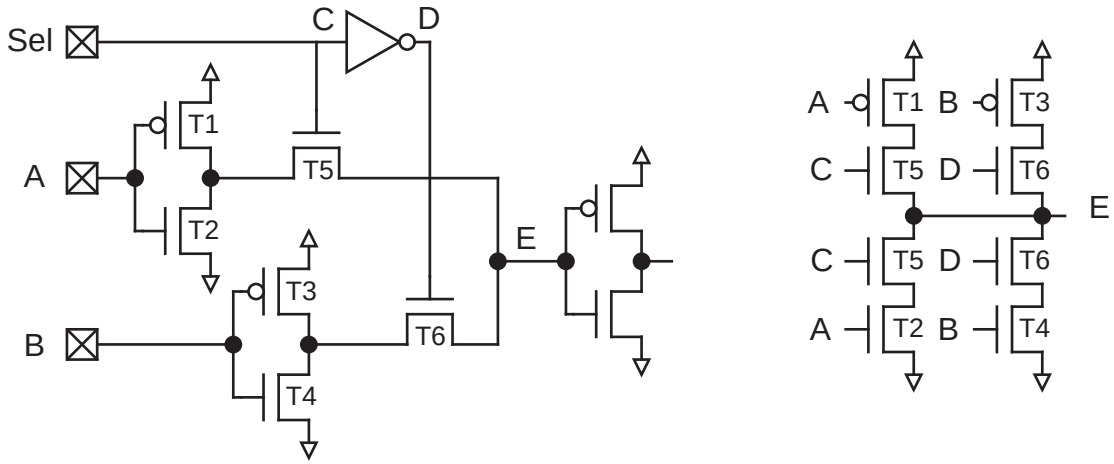
4.2.2 Fabrication des branches

La construction d'un cône sur un nœud *S* consiste à identifier tous chemins de courant entre le nœud *S* et une source de tension. On appelle branche un chemin qui relie le nœud *S à* une source de tension.

Dans la plupart des cas, un cône *S* contient deux types de branches. Les branches V_{dd} expriment les possibilités d'imposer l'état haut sur le nœud *S*, et les branches V_{ss} expriment les possibilités d'imposer l'état bas. Cependant, il existe aussi un troisième type, les branches externes, pour les chemins qui aboutissent à un connecteur relié au drain d'un transistor.

- On appelle ***entrées directes*** du cône *S* l'ensemble des signaux qui attaquent au moins une grille d'un transistor d'une branche du cône *S*.

La Figure 4-2 montre la structure en branches des cônes fabriqués pour un multiplexeur conçu avec des transistors de passage. Une branche se compose de maillons, un pour chaque transistor parcouru. Nous remarquons qu'une branche complète est fabriquée pour chaque chemin distinct, cela implique que le même transistor peut appartenir à plusieurs branches à la fois.



a) Un multiplexeur en transistors de passage b) Le cône associé à E

Figure 4-2 : *Les branches d'un cône*

4.2.3 Caractérisation fonctionnelle

La caractérisation fonctionnelle d'un cône est effectuée à partir de la structure en branches. Une expression booléenne est générée pour l'ensemble des branches V_{dd}. Elle indique la condition de « mise à un » du nœud. Une autre expression est construite pour l'ensemble des branches V_{ss}. Elle indique la condition de « mise à zéro ».

Chaque branche est considérée comme une simple chaîne d'interrupteurs. Pour un transistor de type *N*, l'interrupteur est fermé lorsque le signal de grille est à l'état haut et, inversement, pour un transistor de type *P*, l'interrupteur est fermé à l'état bas. A la différence de ANAMOS, on ne considère pas que les nœuds internes sont capables d'imposer leur état. Autrement dit, les signaux n'ont pas de capacité de mémorisation.

- La fonction de « mise à un » : $S_{UP} = \sum_{VddPaths} \prod_{TN_i} x_i \prod_{TP_j} \overline{x}_j$
- La fonction de « mise à zéro » : $S_{DN} = \sum_{VssPaths} \prod_{TN_i} x_i \prod_{TP_j} \overline{x}_j$

où x_j est une entrée qui contrôle la grille d'un transistor *P* est x_i une entrée qui contrôle la grille d'un transistor *N*.

Avec ces deux expressions, il est possible d'analyser localement la fonctionnalité du cône. Il faut vérifier l'orthogonalité et la complétude de ce couple d'expressions.

- Si les deux expressions sont orthogonales, il n'existe aucune combinaison d'entrées pour laquelle une branche V_{dd} soit active simultanément avec une branche V_{ss}. Autrement dit :

  Si $S_{UP} \cdot S_{DN} = 0$, le cône est « non-conflictuel ».

- Si les deux expressions sont complètes, il n'existe aucune combinaison d'entrées pour laquelle aucune branche ne soit active. Autrement dit :

  Si $S_{UP} + S_{DN} = 1$, le cône est « non haute impédance ».

Un cône qui respecte les deux conditions d'orthogonalité et de complétude est dit CMOS DUAL. On peut alors associer au nœud *S* son expression booléenne en fonction des entrées directes X_i :

$$S \equiv \sum_{VddPaths} \prod_{TN_i} x_i \prod_{TP_j} \overline{x}_j$$

En revanche, un cône qui respecte les deux conditions d'orthogonalité n'est pas nécessairement conflictuel : il peut exister des corrélations entre les entrées X_i du cône qui garantissent que les deux expressions S_{UP} et S_{DN} ne sont jamais vraies simultanément.

4.3 Le problème des fausses branches

4.3.1 Introduction

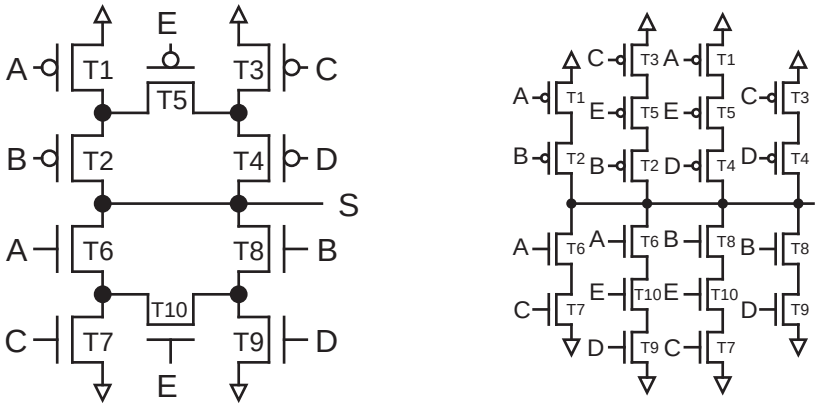
Comme nous avons vu dans les chapitres précédents, la difficulté principale dans le désassemblage des circuits CMOS provient de l'ambiguïté dans l'orientation des transistors MOS. L'étape de construction des cônes passe par l'identification de tous les chemins de courant. Nous allons voir qu'en raison des corrélations existant entre les entrées primaires, certains chemins de courant ne sont jamais passants : ce sont les fausses branches.

4.3.2 Les fausses branches

La nature symétrique des transistors MOS a un impact significatif sur la construction des branches. Sans connaître l'orientation des transistors, il faut construire tous les chemins vers des sources de tension.

La plupart de transistors MOS sont effectivement orientés par leur contexte d'utilisation : c'est à dire qu'on peut identifier sans ambiguïté le drain et la source. Mais, il arrive que des transistors soient bidirectionnels, comme c'est le cas des transistors T_5 et T_{10}

dans la Figure 4-3. Toutefois, il est intéressant de remarquer que, dans cet exemple, ce problème est résolu par le découpage en cônes, car chaque transistor bidirectionnel apparaît deux fois dans le cône : une fois pour chaque orientation.



a) Une porte avec 2 transistors bidirectionnelle b) Le cône correspondant

Figure 4-3 : *Exemple de bidirectionnalité*

Comme nous pouvons le constater dans la Figure 4-4, c'est le contexte logique - c'est-à-dire les corrélations entre différentes entrées directes d'un cône - qui permet d'établir l'orientation de manière rigoureuse. Ce contexte logique permet alors d'éliminer les fausses branches. Ainsi, une approche qui empêche la fabrication des fausses branches résout également le problème d'orientation des transistors.

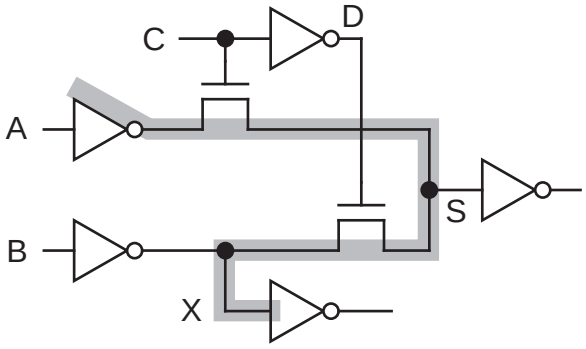


Figure 4-4 : *Fausse branche du cône X*

4.3.3 Elimination des fausses branches

Dans sa thèse [Lau94], Marc Laurentin reconnaît le problème des fausses branches. Cependant, dans DESB, après la construction des cônes, les fausses branches sont supprimées par un post traitement. Ce traitement différé peut se traduire par la construction de cônes de taille importante. Par conséquent, la phase de post-traitement peut devenir très coûteuse, en faisant intervenir un grand nombre de variables. La Figure 4-5 montre le cas extrême d'un

décaleur. Cet exemple est volontairement limité à 3 bits. On remarque déjà que, pour les trois cônes de sorties, une simple recherche en profondeur donne un grand nombre de branches possédant un très grand nombre de transistors. La plupart de ces branches sont des fausses branches.

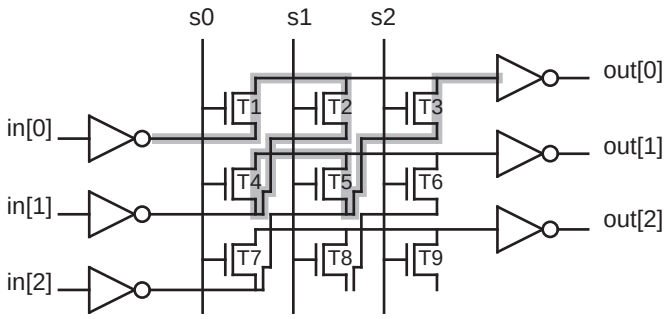


Figure 4-5 : *Une fausse branche dans le décaleur*

Nous proposons donc une autre approche, plus prudente, pour la construction des branches. Pour le décaleur, nous constatons que des fausses branches peuvent être mises en évidence dès le deuxième maillon. Tous les signaux qui commandent les transistors de la matrice sont corrélés : seul un parmi eux peut être actif à la fois. En règle générale, pour des raisons électriques, il est rare, dans les circuits CMOS, de trouver des branches dont la longueur dépasse quatre ou cinq transistors. Cette constatation justifie notre approche qui consiste à construire les branches progressivement, maillon par maillon (Figure 4-6) en s'assurant au fur et à mesure qu'il ne s'agit pas d'une fausse branche.

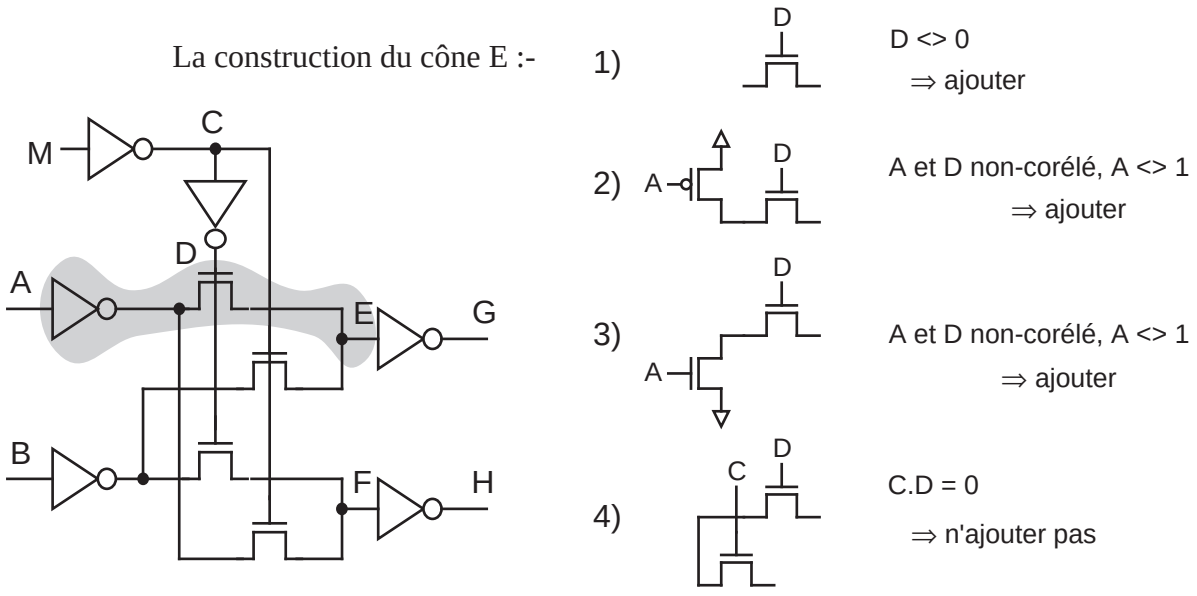


Figure 4-6 : *La construction maillon par maillon*

A l'ajout de chaque maillon, on vérifie si le signal, qui commande la grille du transistor correspondant, est corrélé avec les signaux qui pilotent les transistors des maillons précédents de la branche. S'ils sont corrélés, il faut vérifier que l'ajout du nouveau maillon ne donne pas une branche qui ne soit jamais passante.

4.3.4 Conclusion

L'outil DESB reconnaît le problème des fausses branches, mais, en essayant de « faire le ménage » tout à la fin du désassemblage, se heurte au problème d'explosion du nombre des branches.

Le partitionnement effectué par DESB, sans suppression des fausses branches, peut conduire à des objets aussi complexes que les CCC. Pire encore, ils sont en plus grand nombre. La matrice du décaleur correspond à un seul CCC complexe. DESB construit un cône pour chaque sortie de cette matrice, mais, sans suppression des fausses branches, et chacun de ces cônes contient tous les transistors de la matrice.

Nous proposons alors une solution à ce problème qui consiste à s'appuyer sur le contexte logique d'un cône (c'est-à-dire sur la prise en compte des corrélations entre les entrées) pour éviter la construction des fausses branches. Dans les prochaines sections, nous allons décrire cette solution à partir d'une définition précise de ce contexte logique.

4.4 L'identification des corrélations

4.4.1 Introduction

Pour effectuer l'analyse fonctionnelle, il est nécessaire d'identifier la corrélation dans le circuit, afin de pouvoir l'exploiter dans la phase de fabrication des cônes. La corrélation représente les relations fonctionnelles entre les signaux. Un ensemble de signaux non-corrélés correspond à des signaux statistiquement indépendants.

Les signaux sont souvent fortement corrélés, et le calcul complet de toutes les corrélations est hors d'atteinte [Dun99]. Heureusement, il n'est pas nécessaire d'exploiter *toutes* les corrélations existantes pour résoudre le problème des fausses branches.

4.4.2 Notions de la théorie de graphes

Afin de clarifier la méthode utilisée, nous allons nous appuyer sur une modélisation d'un réseau booléen par un graphe orienté. Nous introduisons dans cette section quelques notions de la théorie des graphes. Ces définitions sont en partie extraites de [Gib85][Gon79].

> Un *graphe G* = [*S*, *A*] est determiné par :
>
> Un ensemble *S* dont les éléments sont les ***sommets*** (ou ***nœuds***) du graphe
>
> Un ensemble *A* dont les éléments sont les ***arcs*** du graphe

Nous spécifions chaque arc par les nœuds qui sont ses *extrémités*. En ce qui nous concerne, il est nécessaire d'associer une direction à chaque arc. Un graphe dont tous les arcs ont une direction est un *graphe orienté*. Ainsi, pour le graphe de la Figure 4-7 nous avons :

$$S = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8\}$$

$$A = \{(s_2, s_1), (s_3, s_1), (s_4, s_3), (s_5, s_3), (s_6, s_5), (s_7, s_5), (s_8, s_6)\}$$

Notons que nous spécifions chaque arc par : (extrémité initiale, extrémité finale)

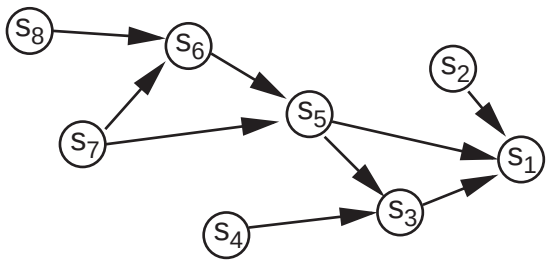


Figure 4-7 : *Exemple d'un graphe orienté*

Dans le contexte d'un graphe orienté, nous définissons :

- Un ***chemin*** entre deux sommets s_1 et s_i est une séquence ($C = s_1, a_1, s_2, a_2, \ldots, a_{i-1}, s_i$) de sommets et d'arcs en alternance, telle que pour $1 \leq j < i$, s_j est l'extrémité initiale de a_j et s_{j+1} est l'extrémité finale de a_j. Nous appelons s_1 et s_i respectivement l'extrémité initiale et l'extrémité finale du *chemin*.

- La ***longueur*** d'un chemin est le nombre d'arcs faisant partie de la séquence de ce chemin.

- Un ***cycle*** est un chemin dont les extrémités coïncident.

- Un ***prédécesseur*** d'un sommet s_i est un sommet s_j telle que s_j est l'extrémité initiale d'un *arc* dont s_i est l'extrémité finale.

- Un ***ancêtre*** d'un sommet s_i est un sommet s_j telle que s_j est l'extrémité initiale d'un *chemin* dont s_i est l'extrémité finale.

- On dit qu'un nœud est un ***nœud terminal*** s'il ne possède aucun prédécesseur.

- On dit que deux nœuds s_i et s_j sont ***indépendants*** si et seulement si :

  $\{\text{ancêtre de } s_i\} \cap \{\text{ancêtre de } s_j\} = \varnothing$

Un ***graphe orienté acyclique*** (« Directed Acyclic Graph » ou DAG) est un graphe orienté qui ne possède aucun cycle. La Figure 4-7 est un exemple de cette classe de graphe.

- Un graphe orienté acyclique contient au moins *un* nœud terminal.

Par ailleurs, on peut définir la notion de connectivité dans un graphe. Intuitivement, deux ensembles de nœuds et d'arcs complètement disjoint correspondent à deux composants connexes (voir la Figure 4-8). Formellement :

- La relation de connectivité $\mathfrak{C}$ ci-dessous définit une partition de l'ensemble des nœuds du graphe en classe d'équivalence.

  Soient s_i et s_j deux nœuds du graphe alors :

  $s_i \mathfrak{C} s_j$ est vrai si et seulement si :

  il existe au moins un chemin dans le graphe dont une extrémité est s_i et l'autre s_j

  ou

  si $\exists\ s_k \in S$ tel que $s_i \mathfrak{C} s_k$ et $s_k \mathfrak{C} s_j$

  On dit alors que s_i et s_j sont des nœuds connexes.

- Deux sommets connexes appartiennent à la même classe d'équivalence. Chaque classe d'équivalence est appelée un ***composant connexe***.

- Un graphe est dit ***connexe*** s'il contient un seul composant connexe.

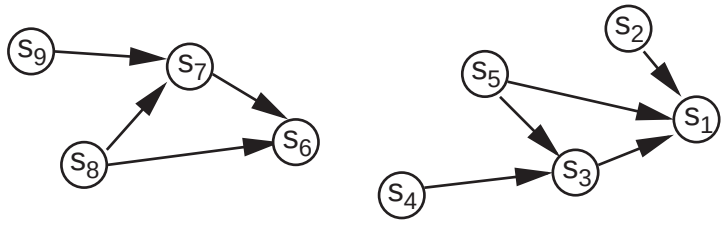


Figure 4-8 : *Un graphe non-connexe contenant deux composants connexes*

- On dit qu'un sommet r d'un graphe orienté G est une ***racine*** de ce graphe si et seulement si :

  Il n'existe aucun chemin ayant pour extrémité initiale r.

La Figure 4-7 donne l'exemple d'un graphe ayant comme racine le nœud s_1.

- Un graphe orienté acyclique possède au moins une racine.

- Soit G un graphe connexe, orienté et acyclique. Si G possède une seule racine r alors :

  $\forall s \in S \mid s \neq r$, il existe dans le graphe un chemin dont l'extrémité initiale est s et l'extrémité finale est r

Soit G un graphe orienté acyclique ayant une seule racine r :

- On définit une ***frontière*** F du graphe G comme un ensemble de nœuds tel que :

  $\forall t \in \{\text{nœud terminaux}\}$, $\forall$ chemin c ayant pour extrémité initiale t et pour extrémité finale r, ce chemin passe par au moins un nœud de F.

- Une frontière F est dite ***indépendante*** si et seulement si $\forall\ s_i, s_j \in F$, s_i et s_j sont indépendants.

- L'ensemble des nœuds terminaux est une frontière indépendante de G.

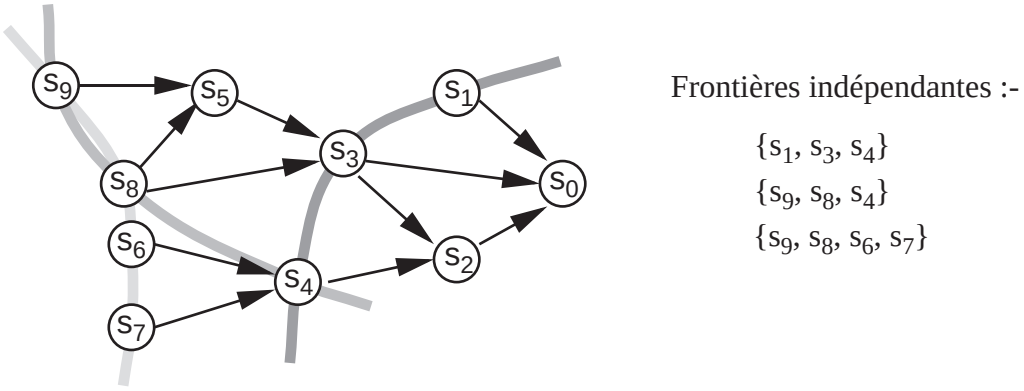


Figure 4-9 : *Les frontières indépendantes dans un graphe*

Définition :	On appelle P ensemble de ***nœuds primaires*** une frontière indépendante de G telle que $\forall s_i \in P$, il n'existe pas de chemin c ayant pour extrémité initiale s_i et pour extrémité finale r passant par un nœud appartenant à une autre frontière indépendante.

- Il existe un ensemble de nœuds primaires pour tout graphe orienté acyclique ayant une seule racine.

Au pire cas, lorsque le graphe présente une seule frontière indépendante (l'ensemble des nœuds terminaux) celle-ci constitue l'ensemble des nœuds primaires.

4.4.3 Nœuds primaires et corrélation

Nous considérons maintenant un graphe orienté et acyclique G. Ce graphe permet de modéliser un réseau booléen. Chaque nœud dans ce graphe correspond à une variable booléenne. Une expression booléenne peut être associée à ce nœud qui exprime l'expression de cette variable en fonction des variables qui correspondent aux nœuds prédécesseurs.

On dit que deux variables sont non-corrélées s'il n'existe aucune dépendance fonctionnelle entre elles. Elles sont alors statistiquement indépendantes.

Formellement :

Deux variables booléennes $\{A, B\}$ sont indépendantes si et seulement si :

$$\text{Prob}(A=1) = \text{Prob}(A=1|B=1) = \text{Prob}(A=1|B=0)$$
$$\text{Prob}(B=1) = \text{Prob}(B=1|A=1) = \text{Prob}(B=1|A=0)$$

Si le graphe possède une seule racine et si on fait l'hypothèse que les nœuds terminaux du graphe sont statistiquement indépendants, il est possible de démontrer que les nœuds primaires de ce graphe représentent un ensemble de variables statistiquement indépendantes en fonction desquelles on peut exprimer l'expression du nœud racine. De plus, cet ensemble représente le plus petit ensemble de variables possédant ces deux propriétés.

4.4.4 Le graphe de dépendance locale

Si l'on considère un circuit numérique synchrone correctement conçu, le réseau de cônes issu du désassemblage et les interconnexions des équipotentielles entre les cônes, représentent un graphe orienté. Ce graphe contient, dans la plupart des cas, des cycles. Mais, si le circuit est correctement construit, ces cycles se limitent aux cycles contenant des éléments mémorisants. Ainsi, en ouvrant le graphe à partir des éléments mémorisants (en considérant que les éléments mémorisants sont à la fois des nœuds terminaux et des nœuds racines), on obtient un graphe orienté acyclique. Ce graphe représente la partie combinatoire du circuit.

On appelle ***Graphe de Dépendance Locale*** d'un cône C, le sous-graphe de ce graphe ayant pour unique racine le cône C et compris entre les nœuds terminaux et le cône C.

Une définition formelle du graphe de dépendance locale serait la suivante :

Définition : Soit $G = [S, A]$ un graphe orienté, le graphe de dépendance locale d'un nœud s_i de S est un sous-graphe G' de G défini par :

$G' = [S', A']$ tel que : $S' = \{\text{ancêtre de } s_i\} \cup \{s_i\}$

$A' = \{\text{arc } a_i \text{ de } A \mid \text{Les deux sommets de } a_i \text{ appartienne à } S'\}$

On peut remarquer que le graphe de dépendance locale d'un cône C est un graphe connexe, orienté et acyclique (si l'on suppose que le graphe du circuit est acyclique) ayant pour unique racine le cône C. La Figure 4-10 montre l'exemple du graphe de dépendance locale pour un multiplexeur.

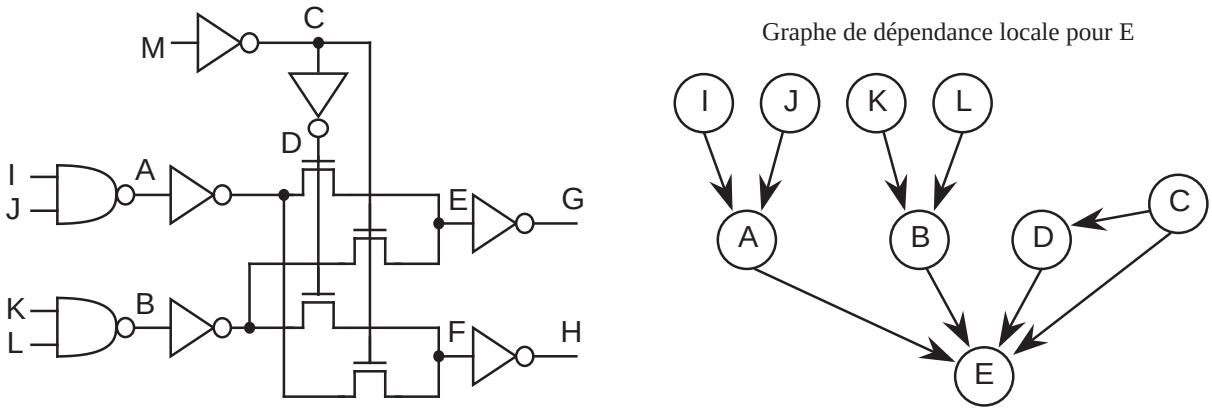


Figure 4-10 : *Exemple d'un graphe de dépendance locale*

Le graphe de dépendance locale regroupe, pour un cône C donné, toutes les dépendances en amont de ce cône. Ainsi, si l'on fait l'hypothèse que les éléments mémorisants et les entrées du circuit représentent des variables booléennes statistiquement indépendantes, alors l'ensemble de nœuds primaires du graphe de dépendance locale constitue le plus petit ensemble de variables booléennes indépendantes qui permet d'exprimer la fonction du cône C.

Ainsi, la recherche des variables primaires pour un cône donné peut être limitée au graphe de dépendance locale.

4.4.5 Recherche des variables primaires

L'algorithme de recherche des variables primaires s'appuie directement sur la définition des nœuds primaires de §4.4.2. Il s'agit, d'abord, pour chaque nœud du graphe de dépendance locale (GDL), de vérifier s'il appartient à une frontière indépendante. Par la suite, on effectue

un parcours en profondeur d'abord, du GDL, pour identifier les nœuds de la frontière indépendante la plus proche de la racine.

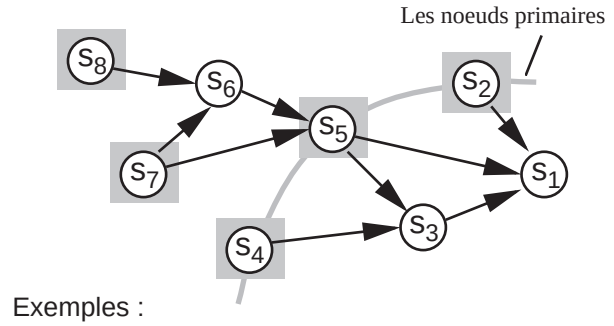
Afin de vérifier l'appartenance d'un nœud à une frontière indépendante, on effectue un parcours du GDL, en profondeur d'abord. Le choix de l'algorithme de parcours est arbitraire ; il est simplement nécessaire que le parcours soit exhaustif.

L'algorithme consiste à parcourir l'ensemble des nœuds du graphe en amont du nœud *X* analysé. En effet, on doit vérifier s'il est possible d'atteindre la racine, à partir d'un nœud en amont, sans passer par le nœud *X*. Un autre parcours, en profondeur d'abord, est effectué en maintenant deux compteurs globaux.

Fils_total : chaque fois qu'un nœud est vue pour la première fois, cette variable est augmentée du nombre de fils de ce nœud.

Fils_visités : incrémenté à chaque visite d'un nœud.

Si après avoir parcouru l'ensemble du graphe en amont d'un nœud, ces deux compteurs indiquent la même valeur, alors le nœud appartient à une frontière indépendante. Cette méthode de vérification est résumée dans la Figure 4-11.



Exemples :

Pour S_3 : Fils_total = 7, Fils_visité = 6 ⇒ NON

Pour S_5 : Fils_total = 4, Fils_visité = 4 ⇒ OUI

```
POUR chaque nœud N
  parcour à partir de N
  POUR chaque nœud S visité
    incrémentez Fils_visité
    SI première visite de S
      accumulez Fils_total
    FIN SI
  FIN POUR
  SI Fils_total == Fils_visité
    marquez nœud N
  FIN SI
FIN POUR
```

Figure 4-11 : *Recherche des variables primaires*

L'algorithme paraît complexe, à cause de la double récursivité. Cependant, l'identification des nœuds qui appartient à une frontière indépendante est extrêmement rapide car l'opération élémentaire, la mise à jour des deux variables, est peu coûteuse.

Une fois que l'ensemble des nœuds appartenant aux frontières indépendantes ont été identifiés, on procède à l'identification des nœuds primaires. Cette étape est relativement simple car il s'agit de trouver les nœuds appartenant à une frontière indépendante qui, sur leur

chemin vers la racine, ne rencontrent pas d'autres nœuds appartenant à une frontière indépendante.

4.4.6 La limitation de la profondeur

En pratique, il n'est pas toujours possible de construire entièrement le graphe de dépendance locale. Par exemple, lors de la construction du graphe on peut rencontrer des cônes conflictuels dont la fonctionnalité ne peut pas être exprimée par une fonction booléenne. Dans ce cas, nous considérons ces nœuds comme des nœuds terminaux.

De plus, il est souhaitable de limiter la construction du graphe de dépendance locale afin de ne pas rechercher des corrélations trop éloignées. Intuitivement, plus une corrélation est éloignée, moins elle a de chance d'apporter des informations pertinentes pour l'élimination des fausses branches. Ainsi, nous appliquons une heuristique de distance pour limiter la construction du graphe de dépendance locale aux corrélations les plus proches.

Nous introduisons formellement la notion de profondeur dans le graphe de dépendance locale :

Définition : La profondeur d'un nœud est égale à la longueur du plus long chemin entre ce nœud et le nœud racine.

Ainsi, lors de la construction du graphe de dépendance locale nous limitons la profondeur des nœuds à une valeur maximale fixée à l'avance. Le choix de la valeur de ce paramètre est laissé au concepteur, car il est impossible de connaître la profondeur idéale avant d'avoir effectué l'analyse.

4.4.7 Conclusion

Dans cette section, nous avons présenté deux idées importantes. La première est la notion du contexte logique d'un cône. Ce contexte est défini par la fabrication du graphe de dépendance locale. Ce graphe inclut les dépendances entre les cônes, en amont d'un cône sous analyse et qui influencent son état. Nous avons également introduit le paramètre de profondeur qui définit la taille du voisinage pris en considération.

La deuxième notion est celle de l'exploitation du graphe, pour la recherche des corrélations qui peut avoir une influence sur la construction des branches. Il est possible de prendre en compte cette corrélation en exprimant les entrées directes des cônes en fonction des

variables primaires. Nous avons vu que les nœuds correspondant à ces variables appartiennent à la frontière indépendante la plus proche du cône.

4.5 Fabrication des cônes et analyse fonctionnelle

4.5.1 Introduction

Après avoir défini la méthode d'identification des variables primaires d'un cône, nous pouvons maintenant étudier l'exploitation de ces informations dans la phase de fabrication des branches. Notre objectif est de maintenir une approche prudente pour éviter de construire des fausses branches pour lesquelles les expressions booléennes deviennent trop complexes, en temps de calcul ou en consommation mémoire. Nous proposons une approche qui permet de réduire au minimum les redondances dans les calculs.

Comme nos travaux concernent les circuits dont la majeure partie se compose de logique numérique (voir §1.2), une approche en deux phases semble être adaptée au problème. La première phase tente d'extraire rapidement toutes les portes CMOS duales simples. Cette extraction est effectuée sous des contraintes sévères, pour maximiser l'efficacité. La deuxième phase s'appuie sur la logique extraite dans la phase initiale. Cette phase correspond à la construction, maillon par maillon des branches pour les cônes qui restent à construire. L'efficacité globale de la méthode est maintenue par le fait qu'elle met en œuvre un algorithme complexe uniquement lorsque la structure du circuit le nécessite.

4.5.2 Extraction des portes duales

On commence par identifier les nœuds qui satisfont les critères de fabrication d'un cône. Le critère est l'existence d'au moins un connecteur de grille de transistor. La première phase consiste à identifier les nœuds sur lesquels il est possible de construire un ensemble de branches équivalent à une porte CMOS duale. Pour qu'une porte soit identifiée comme une porte CMOS duale, cet ensemble de branches doit respecter des contraintes sévères :

1) Le nœud de sortie doit correspondre à la connexion entre une source ou un drain d'un transistor de type *N* et une source ou un drain d'un transistor de type *P*.

2) Tous les transistors faisant partie du sous-ensemble de branches de type V_{dd} doivent être de type *P*. Ils doivent être de type *N* pour des branches V_{ss}.

3) Le cône ne doit pas comporter de branches externes.

4) Aucun des nœuds reliant les sources et les drains des transistors sur une branche ne doit être relié à une grille de transistor.

5) Les deux expressions booléennes S_{UP} et S_{DN}, qui expriment respectivement la fonctionnalité de l'ensemble des branches V_{dd}, et de l'ensemble des branches V_{ss}, doivent satisfaire les critères de complétude et d'orthogonalité de la §4.2.3.

L'application de ces critères sévères, résumé dans la Figure 4-12, permet d'extraire rapidement toutes les portes CMOS duales.

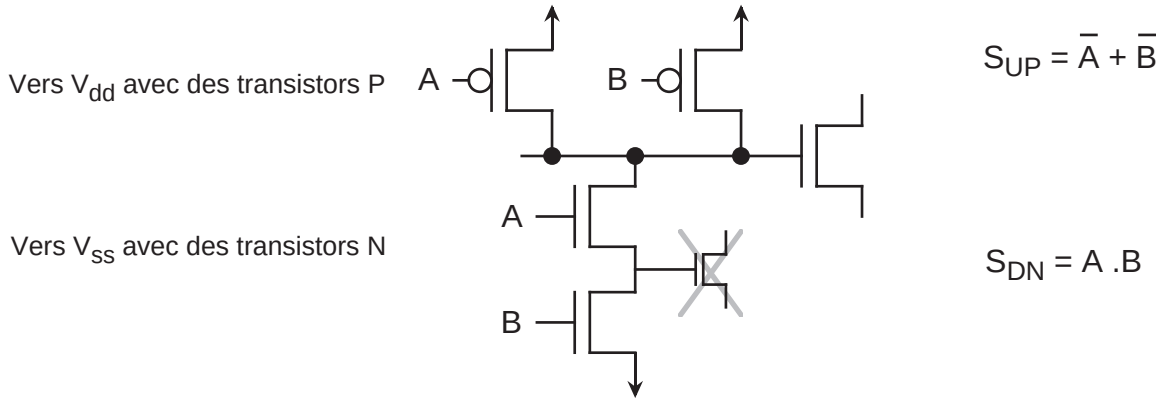


Figure 4-12 : *Critères d'extraction d'une porte CMOS*

4.5.3 Fabrication des cônes non-duaux

Selon la structure des circuits, une partie plus ou moins importante du circuit reste non-traitée après la phase d'extraction des portes CMOS duales. Certains nœuds, qui satisfont les critères de fabrication d'un cône, n'ont aucune branche construite. D'autres ne contiennent que quelques branches.

Pour compléter ces cônes, nous exploitons le contexte logique de chacun de ces cônes. Ce contexte est défini par le graphe de dépendance locale (voir §4.4.4). La manipulation des expressions booléennes peut, dans certaines situations, devenir très coûteuse. Le décaleur de la Figure 4-5 est l'exemple par excellence. Pour chacune des sorties, il faut démontrer que seulement une des lignes de contrôle (une pour chaque bit du décaleur) est active à la fois.

4.5.3.1 Le graphe de dépendance globale

L'extraction complète d'un cône qui ne correspond pas à une porte CMOS duale nécessite l'extraction préalable de son contexte logique. Pour minimiser les parcours redondants, il est utile d'établir un ordre dans le traitement des cônes, afin de ne traiter un cône qu'après l'extraction de tous les cônes dont il dépend.

Cet ordre peut être obtenu à partir du graphe de dépendance globale (GDG) qui représente les connexions entre tous les cônes du circuit. Dans ce graphe, chaque cône est relié à un autre si la sortie de ce dernier est une entrée directe du premier.

Après la phase d'extraction des portes CMOS duales. Pour certains cônes, toutes les entrées directes sont déjà connues. Pour d'autres, la liste des entrées directes est partiellement ou totalement inconnue. Pour pouvoir construire le graphe de dépendance globale, il est donc nécessaire d'établir pour ces cônes une liste approximative des entrées. Cette liste correspond à l'ensemble des signaux qui contrôlent les grilles des transistors susceptibles d'appartenir à une ou plusieurs branches du cône.

Il est important de noter que la qualité de cette approximation détermine en partie la performance de l'extraction. En effet, si le graphe de dépendance globale est correctement construit, l'ordre dans lequel les cônes vont être construits est correct. Dans ce cas, lors de la construction d'un cône, toutes les informations de corrélation en amont de ce cône sont déjà disponibles et peuvent être exploitées pour la construction du graphe de dépendance locale.

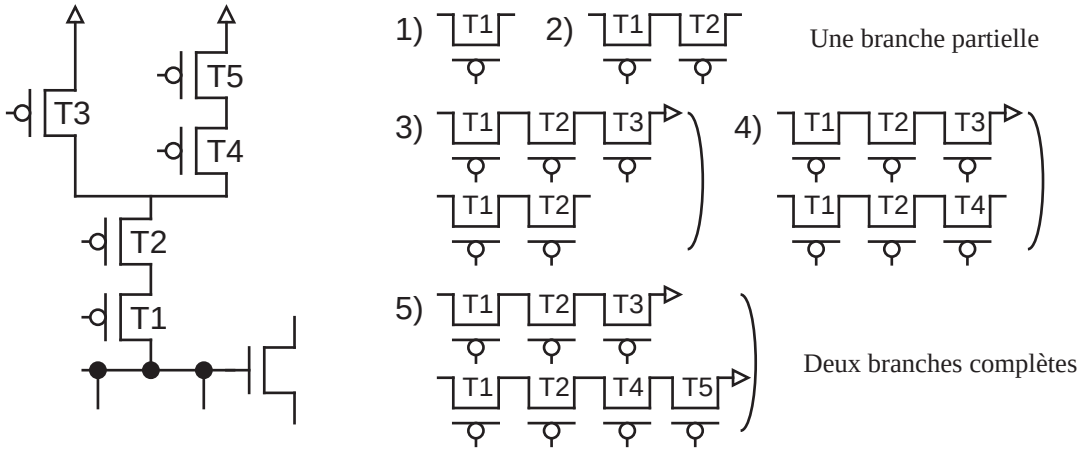
Dans le pire des cas, le graphe de dépendance globale peut contenir des cycles qui ne se trouvent pas réellement dans le circuit, mais qui sont due aux fausses branches. Ces boucles peuvent empêcher la résolution de certains cônes. Ainsi il arrive qu'on cherche à construire un cône pour lequel on ne dispose pas encore de suffisamment de cônes en amont. On peut être amené à effectuer plusieurs parcours du graphe de dépendance globale, car la construction d'un cône peut influencer la construction du graphe de dépendance locale d'un autre cône.

4.5.3.2 La construction des branches

Le but de la deuxième phase de la construction de branches est l'identification de tous les chemins DC connexe entre le nœud de sortie d'un cône et une source de tension en évitant la construction des fausses branches.

L'idée principale est la construction maillon par maillon des branches, à partir du nœud de sortie du cône. La définition d'un cône (§4.2) impose la construction d'une branche distincte pour chaque chemin DC. Ainsi, pendant l'agrandissement d'une branche, si on atteint un nœud à partir duquel il existe plusieurs chemins vers des sources de courant, la branche est dupliquée pour chaque chemin possible. Chacune de ces nouvelles branches est

ensuite agrandie récursivement. Cette procédure de duplication et de construction récursive est illustrée dans la Figure 4-13.



a) Un réseau de transistors b) La construction des branches équivalentes

Figure 4-13 : *Agrandissement récursive des branches*

Lors de l'addition d'un maillon à une branche, il faut vérifier que la condition de conduction de la branche, n'est pas identiquement nulle.

Autrement dit que : $\prod_{i \in T_N} x_i \cdot \prod_{j \in T_P} \overline{x}_j \neq 0$

x_i représente un signal qui pilote la grille d'un transistor de type 'N' et x_j un signal qui pilote un transistor 'P'.

Pour vérifier si l'expression de conduction d'une branche est identiquement nulle, il faut expliciter les corrélations entre les variables x_i et x_j Le graphe de dépendance locale et les variables primaires remplissent cette fonction. A cet effet, il suffit d'exprimer, dans la condition de conduction, les x_i et x_j en fonction des variables primaires (qui sont par définition indépendantes les unes des autres). Autrement dit, il faut remplacer les x_i et x_j par leur expression booléenne en fonction des variables primaires. Si l'expression obtenue n'est pas identiquement nulle, alors il existe au moins une configuration des nœuds terminaux qui rend la branche passante et il ne s'agit donc pas d'une fausse branche.

On peut remarquer que la condition de conduction doit être re-examinée au moment de l'ajout d'un maillon à la branche. Or, le signal de grille de ce maillon n'est pas forcément corrélé avec tous les signaux de grille des autres transistors de la branche. Ainsi, pour vérifier la condition de conduction, il suffit de développer uniquement les signaux de grille corrélés

avec le signal de grille du transistor que l'on désire ajouter. Il s'agit des signaux qui, dans le graphe de dépendance locale, dépendent d'au moins une variable primaire dont dépend le signal de grille du transistor à ajouter. (voir la Figure 4-14)

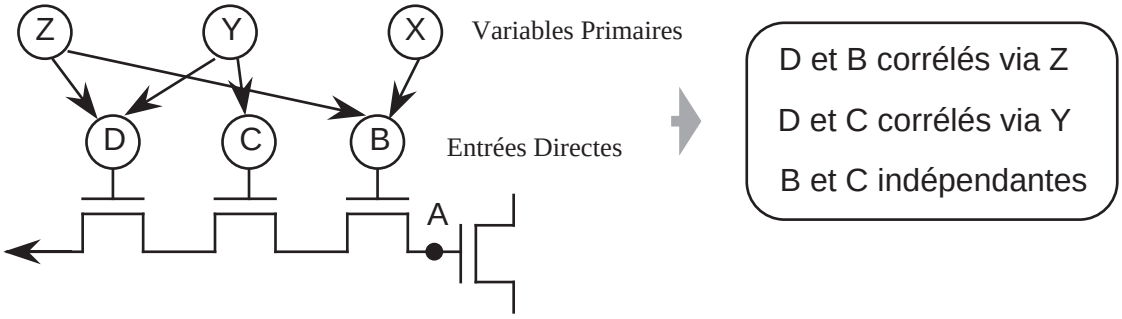


Figure 4-14 : *Les corrélations au sein d'une branche*

Cette procédure de construction récursive des branches, maillon par maillon, est effectuée jusqu'à ce que toutes les branches d'un cône aient été construites ou éliminées. Le cône est alors considéré complet ; cet état permet l'utilisation de sa fonctionnalité pour la fabrication d'autres cônes. Cependant, il faut d'abord établir les caractéristiques fonctionnelles de ce cône.

4.5.4 Phase de caractérisation globale

La caractérisation globale d'un cône ressemble à la caractérisation locale (§4.2.3) dans la mesure où nous voulons vérifier s'il est possible que le cône soit dans un état conflictuel ou haute impédance. Cependant, à la différence de cette analyse locale, les propriétés de conflictualité et de haute impédance doivent être vérifiées par rapport aux variables primaires du cône plutôt qu'à ses entrées directes.

Dans un premier temps, les conditions de « mise à un » et de « mise à zéro » du cône sont établies en fonction des entrées directes du cône. Une vérification locale de la complétude et de l'orthogonalité permet d'identifier immédiatement les portes duales.

S'il n'est pas possible d'affirmer l'orthogonalité ou la complétude localement, alors les deux expressions sont re-exprimées en fonction des variables primaires du cône. Cette opération s'effectue par une substitution de chaque variable par son expression en fonction des variables primaires.

Si les conditions d'orthogonalité et de complétude sont vérifiées, on peut considérer le cône comme pseudo-dual et lui associer une expression booléenne $S = S_{UP}$ comme pour un cône dual, qui peut alors être exploitée pour la fabrication des cônes qui en dépendent.

4.5.5 Conclusion

Dans cette section, nous avons montré comment une méthode à deux étapes permet de traiter une grande diversité de styles de circuit tout en minimisant la complexité des calculs à effectuer.

La première étape d'extraction de portes CMOS duales se déroule très rapidement grâce aux contraintes sévères qui se traduisent par des conditions simples à vérifier localement. L'expérience montre que cette première phase permet de traiter une partie très importante du circuit.

La deuxième phase, bien plus lourde, s'appuie sur les résultats obtenus dans la pr e-mière phase. Cette phase exploite le contexte logique d'un cône, défini par le graphe de dépendance locale, pour contrôler la fabrication de ce cône. A l'aide de cette approche nous avons pu éviter les calculs redondants et coûteux d'un post-traitement des cônes pour la suppression des fausses branches.

4.6 Le problème des boucles

Dans la section précédente, nous avons fait l'hypothèse que le graphe de dépendance locale est un graphe acyclique. Cette hypothèse est nécessaire pour la définition de la frontière et la détermination des variables primaires. Heureusement, la méthode de construction du graphe garantit le respect de cette contrainte.

Toutefois, il est possible que le graphe de dépendance globale, qui représente les dépendances entre l'ensemble des cônes du circuit, puisse contenir des boucles. Certaines de ces boucles existent uniquement dans l'approximation initiale, et vont disparaître lors de la fabrication des cônes. D'autres ont une existence réelle dans le réseau de cônes. Elles peuvent, dans certains cas, exiger des traitements particuliers.

En principe, un circuit synchrone correctement conçu, ne doit jamais contenir de vraie boucles fonctionnelles ne doit jamais exister dans les parties combinatoires. Cependant, il est possible que l'approximation initiale du graphe de dépendance globale fasse apparaître des

boucles dans la partie combinatoire. L'effet de ces boucles est de rendre sous-optimal le parcours du graphe pour l'ordonnancement des cônes dans la phase de fabrication. Cet effet est indésirable, mais n'empêche pas le bon fonctionnement de la méthode.

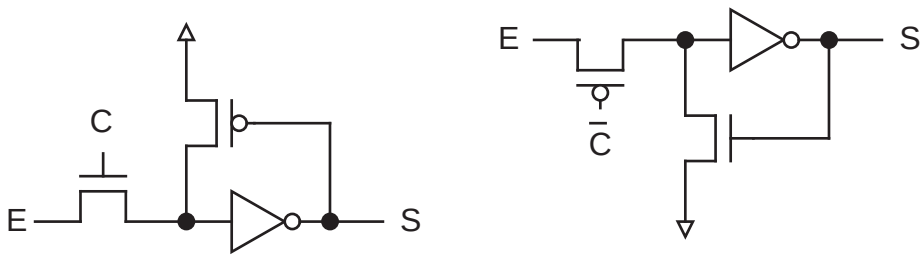
Toutefois, dans un circuit synchrone correctement conçu, en dehors des parties combinatoires, il existe des boucles. Ces boucles sont de trois types :

1) Les éléments mémorisants.
2) Les boucles séquentielles contenant un élément mémorisant.
3) Les bleeders.

Ainsi, si on écarte le cas des bleeders, dont le traitementest simple, l'identifications des éléments mémorisants permet de résoudre la difficulté posé par les boucles dans la caractérisation fonctionnelle.

4.6.1 Les bleeders

Les « bleeders » sont utilisés pour rétablir le potentiel électrique d'un signal dont le niveau est dégradé. Un exemple typique de ce montage est donné dans la Figure 4-15. Cette structure pose un problème particulier, car elle représente une boucle entre deux cônes dans une partie combinatoire (voir Figure 4-16). La construction des graphes de dépendance locales peut alors se heurter à l'existence de telles boucles dans le circuit et l'exploitation des corrélations peut être appauvrie. De plus, la présence du bleeder est uniquement liée à des problèmes électriques et n'influence pas la fonction logique du cône.

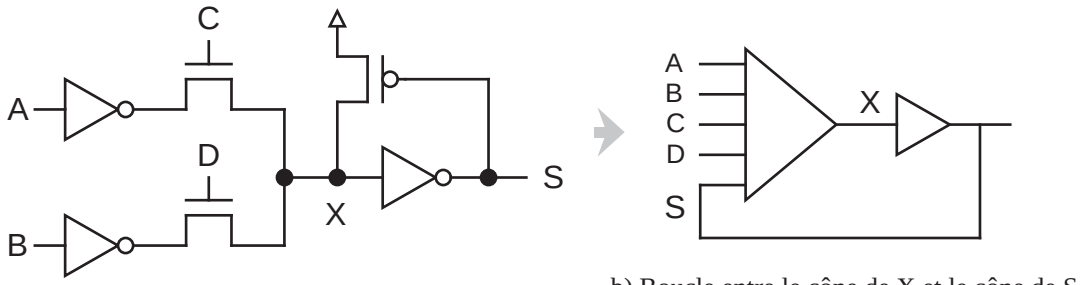


a) Remise à niveau d'un état haut dégradé b) Remise à niveau d'un état bas dégradé

Figure 4-15 : *Les structures de bleeders*

Nous avons donc besoin d'identifier les bleeders avant la phase principale de fabrication des cônes, afin qu'ils ne soient pas considérés comme des boucles pendant la fabrication des graphes de dépendances locales. Toutefois, la forme d'un bleeder est très simple et il n'y a pas beaucoup de variations possibles. Il s'agit toujours d'une branche avec un seul

transistor, dont la grille est pilotée par l'inverse du signal de sortie du cône auquel il appartient. Ces structures sont donc très simples à repérer. En pratique, l'identification et l'élimination des bleeders sont effectuées à la suite de la phase d'extraction des portes CMOS duales.



a) Un multiplexeur avec bleeder

b) Boucle entre le cône de X et le cône de S si le bleeder n'est pas éliminé

Figure 4-16 : *Les bleeders et les fausses boucles*

4.6.2 Les éléments séquentiels

A la fin de la phase de fabrication des cônes, la seule indication de l'existence potentielle des éléments séquentiels est l'occurrence des boucles qui ne sont pas des bleeders. Dans tous les circuits que nous avons rencontrés, ce sont des boucles faisant intervenir deux cônes dont chacun est une entrée de l'autre.

Une première méthode pour vérifier si une boucle correspond à un élément séquentiel pourrait être de comparer la structure des cônes dont la boucle est composée avec des modèles connus. Cette approche qui a été adoptée par le logiciel DESB[Lau94].

Cependant, cette méthode souffre de son manque de généralité, et également de la difficulté de l'adapter pour reconnaître des éléments séquentiels inconnus. C'est pour cette raison que nous présentons, dans le chapitre suivant, une méthode purement analytique pour l'identification et la modélisation des éléments séquentiels.

4.7 Conclusion

Dans ce chapitre, nous avons détaillé le modèle de cônes, inspiré de DESB. Ce modèle est la base de notre méthode de partitionnement du réseau de transistors. Nous avons montré comment, à partir de la structure d'un cône, il est possible de générer automatiquement une caractérisation fonctionnelle. Etant donné que le partitionnement d'un circuit en cônes couvre

entièrement le circuit, la caractérisation de tous les cônes permet de générer un modèle comportemental global du circuit.

Nous avons vu que la difficulté principale dans la fabrication des cônes est celle des fausses branches. Ces fausses branches représentent des chemins entre la sortie d'un cône et une alimentation qui ne peuvent pas être activés à cause de la corrélation entre les signaux qui attaquent les grilles des transistors de la branche.

En nous appuyant sur une formalisation qui fait appel à la théorie des graphes, nous avons montré qu'il est possible d'obtenir, pour chaque cône, un ensemble de nœuds du circuit qui représente des variables indépendantes. Les entrées directes d'un cône peuvent être re-exprimées en fonction de ces variables indépendantes. Par la suite, nous avons expliqué les étapes de notre algorithme qui, en s'appuyant sur ces variables dites primaires, permet d'identifier une fausse branche pendant sa propre fabrication.

Finalement nous avons vu que l'existence des boucles, dans le réseau de cônes, pouvait avoir une influence néfaste sur l'identification des fausses branches. En effet, les corrélations qui se trouvent en amont de ces boucles ne peuvent pas être exploitées. Nous avons vu que dans certains cas, ces boucles pouvaient être éliminées simplement car elles n'avaient pas de rôle fonctionnel. Le chapitre suivant est dédié au problème de traitement des boucles associées aux éléments séquentiels.

Chapitre 5

TRAITEMENT DES ELEMENTS SEQUENTIELS

Dans ce chapitre nous présentons la méthode de traitement des éléments séquentiels. Après avoir discuté des objectifs, nous exposons en détail une méthode analytique pour l'identification des points mémorisants d'un circuit. Dans la dernière partie, nous montrons comment générer automatiquement un modèle comportemental à partir des expressions obtenues pendant la phase d'identification.

5.1 Introduction

Le traitement des éléments mémorisants, tels que les latchs et les bascules, présente une des difficultés majeures pour un outil d'abstraction fonctionnelle. Les éléments mémorisants les plus couramment utilisés sont des bascules à échantillonnage sur front. En pratique, une bascule à échantillonnage sur front est le plus souvent réalisée par la mise en série de deux latchs à mémorisation sur état (Figure 5-1). Une bascule à échantillonage sur front est donc un objet complexe construit à partir d'objets plus simples que sont les latchs. C'est pourquoi nous nous intéressons en priorité, dans ce chapitre, à l'identification et à la caractérisation des latchs.

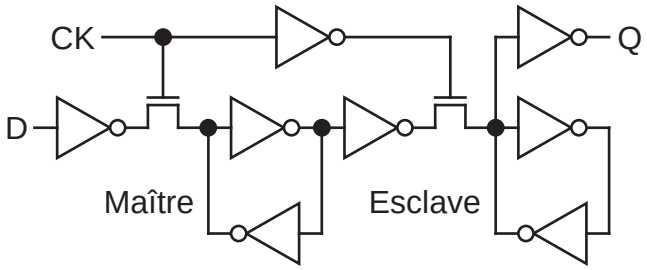


Figure 5-1 : *Réalisation d'une bascule avec deux latchs à conflit*

Dans ce chapitre, nous faisons l'hypothèse qu'un latch est constitué de deux cônes en boucle. Comme nous l'avons signalé dans le chapitre précédent, cette hypothèse a été vérifiée sur tous les circuits que nous avons rencontrés.

Les latchs ne posent pas de problème particulier pour le découpage en cônes. Toutefois, leur présence a une influence sur l'analyse fonctionnelle, car ce sont des points d'arrêt pour la fabrication des graphes de dépendances locales des cônes.

De plus, la procédure de génération du modèle comportemental d'un latch est différente de celle utilisée pour un élément combinatoire.

Il existe une très grande diversité dans les schémas permettant de réaliser un latch en circuiterie CMOS, et l'une des faiblesses de l'outil DESB est son manque de généralité dans l'identification des latchs. La liste des structures que l'outil est capable de reconnaître et de modéliser correctement, est limitée et non-extensible.

5.2 Les Objectifs

Nous suivons donc deux objectifs principaux dans le traitement des latchs. Le premier est d'identifier tous les cônes qui constituent des latchs. Idéalement, cette identification doit avoir lieu aussi tôt que possible pendant la phase de construction des cônes, car cette identification influence la construction des graphes de dépendances locales des cônes.

Cette influence est souvent très subtile. En effet, le fait que l'élément mémorisant corresponde à un point d'arrêt pour les graphes de dépendances locales ne pose pas un grand problème. Mais, les structures rebouclées qui caractérisent les latchs entraînent la construction de fausses branches (voir la Figure 5-2). Dans le cas des latchs « à conflit », il n'est pas possible d'éliminer ce type de branche à l'aide d'une analyse fonctionnelle car ce sont les caractéristiques électriques de l'élément mémorisant qui permettent de conclure qu'il s'agit d'une fausse branche.

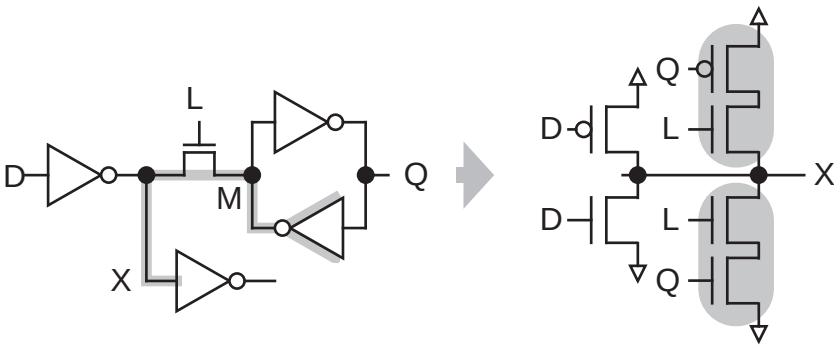


Figure 5-2 : *Les fausses branches dues au point mémorisant*

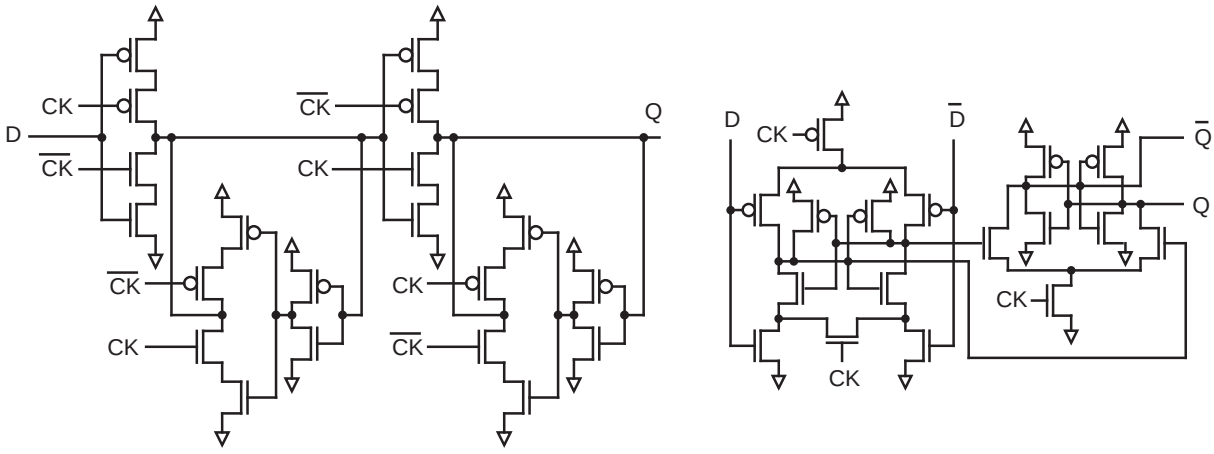
Le deuxième objectif est la génération de modèles comportementaux lisibles. Ces modèles doivent être obtenus directement à partir du réseau de transistors afin de refléter le plus fidèlement le comportement du circuit. Cependant, la génération du modèle comportemental peut nécessiter une étape d'abstraction supplémentaire pour identifier une bascule à échantillonnage sur front à partir de deux latchs.

Les travaux décrits dans ce chapitre se limitent explicitement au traitement des latchs statiques. Le traitement des latchs dynamiques est un problème très différent. La caractéristique distinctive des latchs dynamiques est l'existence d'une charge stockée sur une capacité temporairement en haute impédance. La difficulté principale est donc de différencier les mémoires dynamiques des bus à trois états.

Cette restriction aux latchs statiques nous permet d'exploiter leur particularité qui est l'existence d'une boucle de mémorisation. Dans la pratique, ces boucles contiennent toujours deux cônes, car cela minimise le temps nécessaire pour atteindre la stabilité. Nous utilisons donc l'existence d'une boucle entre deux cônes comme critère de base pour l'identification des latchs statiques.

5.3 Motivations d'un traitement algébrique

Une approche basée sur la reconnaissance de certaines structures dans le réseau de cônes, telle que l'approche mise en œuvre dans DESB, permet un traitement rapide dans un grand nombre de cas pratiques. Cependant, ce type d'approches ne remplit pas l'objectif de généricité de l'abstraction fonctionnelle. En effet, une méthode d'abstraction doit répondre au moins aux deux exigences suivantes.



a) Bascule réalisée avec deux latchs sans conflits b) Bascule différentiel

Figure 5-3 : *Deux types de bascules maitre/esclave*

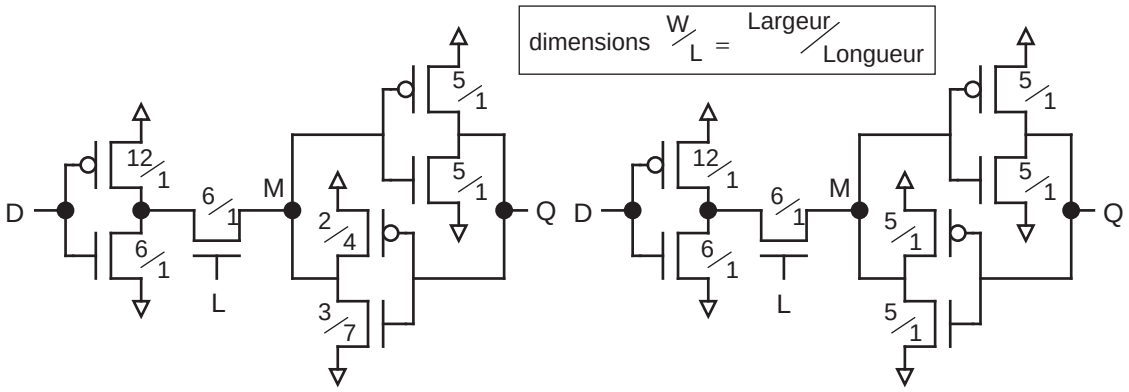
5.3.1 Prise en compte des structures inconnues

Une méthode d'abstraction doit être capable de traiter des éléments séquentiels dont la structure est a priori inconnue. Cette remarque devient encore plus pertinente avec les circuits modernes, car les exigences de vitesse et de réduction de la consommation ont conduit à la conception des structures de plus en plus exotiques comme nous pouvons le voir dans le second exemple de la Figure 5-3. Dans un tel contexte, il devient difficile de programmer « à priori » dans l'outil, l'ensemble des structures reconnues comme des éléments mémorisants.

5.3.2 Prise en compte des caractéristiques physiques

Pour que le résultat de l'abstraction fonctionnelle soit exploitable par des outils de simulation, de preuve formelle ou de synthèse, il n'est pas suffisant d'identifier les éléments séquentiels, il est aussi nécessaire de générer des modèles comportementaux fidèles au comportement réel. Idéalement, ce modèle doit être dérivé, d'une manière algorithmique, de la réalisation physique.

La Figure 5-4 montre un exemple simple de l'influence de la réalisation sur le comportement. L'exemple montre un latch conflictuel ordinaire dans deux versions : la première fonctionne correctement, la seconde non. Le dysfonctionnement est dû au dimensionnement incorrect des transistors : l'inverseur de rebouclage est trop puissant par rapport à l'inverseur de l'émetteur.



a) Latch correct b) Latch mal-dimensionné

Figure 5-4 : *Dimensionnement des transitors pour un latch*

Une solution générale au problème doit tenir compte des aspects physiques tels que les dimensions des transistors.

5.4 L'identification par une méthode algébrique

5.4.1 Introduction

Comme nous l'avons dit, la présence possible d'un latch statique est indiquée par l'existence d'une boucle entre deux cônes. La méthode est basée sur l'utilisation des expressions booléennes pour identifier les conditions de stabilité de la boucle. Ces expressions booléennes sont elles-mêmes obtenues directement à partir du réseau de transistors de chaque cône.

Nous allons illustrer la procédure à l'aide d'un exemple représentatif des problèmes à résoudre. La Figure 5-5 donne le schéma du circuit que nous allons étudier.

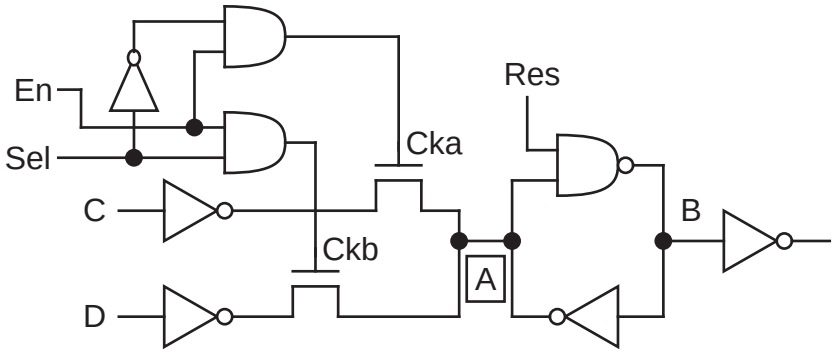


Figure 5-5 : *Exemple de référence pour le traitement générique*

Ce circuit a trois particularités principales :

1) C'est un latch conflictuel. L'écriture d'une nouvelle donnée se fait de manière conflictuelle avec l'inverseur de rebouclage.

2) Le latch possède une « remise à zéro » asynchrone active sur l'état bas.

3) Le latch a été conçu pour faire partie d'un chemin de test (scan path). Il possède deux emetteurs, chacun ayant sa propre horloge. Les deux horloges (*Cka* et *Ckb*) sont mutuellement exclusives, grâce à une logique de contrôle en amont, représentée par le signal *Sel*.

5.4.2 Analyse des conflits dans une boucle

A l'issue de la caractérisation fonctionnelle des cônes, nous avons réussi à établir deux expressions booléennes pour chaque cône (S_{UP} : condition de mise à un, et S_{DN} : condition de mise à zéro).

Dans le cas général, les deux expressions booléennes peuvent ne pas être orthogonales. Autrement dit, il peut y avoir des conditions pour lesquelles $S_{UP} = S_{DN} = 1$.

On peut distinguer trois types de conflits :

1) Les faux conflits fonctionnels : il s'agit de faux conflits qui peuvent être résolus grâce aux corrélations entre les entrées d'un cône.

2) Les faux conflits électriques : il s'agit de vrais conflits fonctionnels mais dont le résultat peut être déterminé grâce à une analyse électrique.

3) Les vrais conflits qui correspondent à des erreurs de conception.

De même, il se peut que les conditions de mise à un et de mise à zéro ne soient pas complètes. Autrement dit que $S_{UP} + S_{DN} \neq 1$. Dans ce cas, il existe des configurations des entrées directes pour lesquelles il n'y a pas de branches actives. On peut distinguer deux cas :

1) Les fausses hautes impédances : il s'agit des configurations des entrées directes qui provoquent une haute impédance mais qui ne peuvent pas se produire à cause des corrélations en amont de la boucle.

2) Les vraies hautes impédances qui correspondent à des erreurs de conception.

5.4.2.1 Les faux conflits fonctionnels

Comme nous l'avons vu dans le chapitre précédent, une des raisons fondamentales pour lesquelles les deux expressions de mise à un et à zéro ne sont pas complémentaires réside dans le fait que les entrées directes ne sont pas indépendantes. Il faut alors tenir compte de ces corrélations. La méthode que nous avons exposée dans le chapitre précédent, et qui consiste à re-exprimer la fonctionnalité du cône en terme de variables primaires, semble bien adaptée.

Toutefois, la présence de la boucle entre les deux cônes complique l'identification des variables primaires. Pour contourner ce problème, nous proposons d'ouvrir virtuellement la boucle avant de construire le graphe de dépendances locales et d'identifier les variables primaires (voir Figure 5-6).

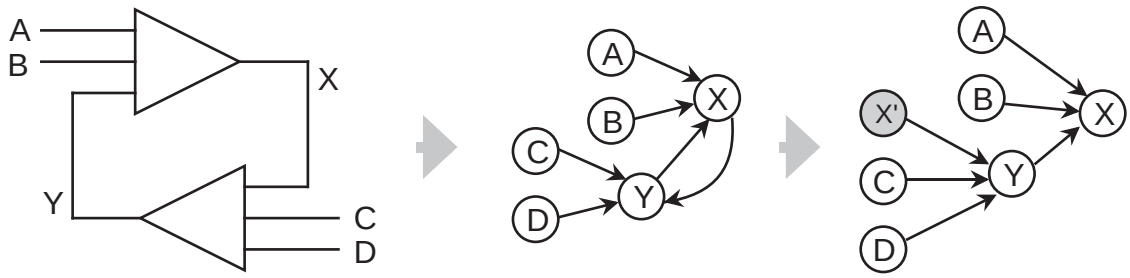


Figure 5-6 : *La boucle et le graphe de dépendance de X*

Cependant, en ouvrant la boucle, on néglige une certaine corrélation. Il s'agit de la corrélation entre le nœud qui a été coupé et les autres entrées de la boucle. Pour résoudre cette difficulté, on impose que le nœud de la boucle qui a été coupé fasse partie des variables primaires. Une autre manière de voir ce problème est de considérer que pour faire apparaître cette corrélation, il faut dérouler la boucle deux fois avant de l'ouvrir.

A l'issue de cette étape, nous exprimons les conditions de « mise à un » et de « mise à zéro » en fonctions des variables primaires afin d'éliminer les conflits non fonctionnels.

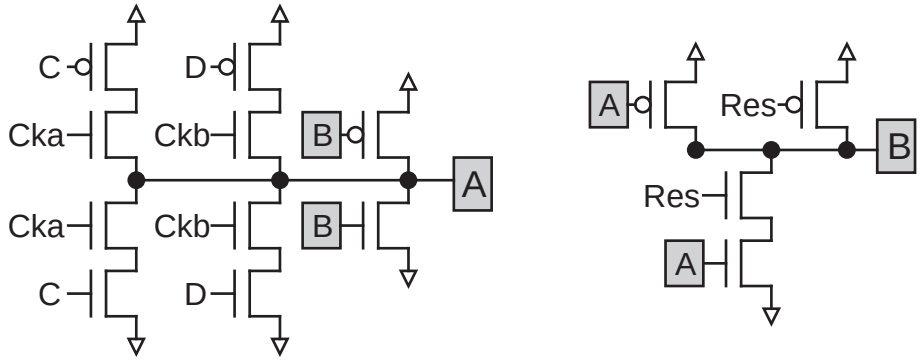


Figure 5-7 : *Les cône construits pour l'exemple de la Figure 5.5*

Pour notre exemple, la Figure 5-7 montre les deux cônes qui composent la boucle. Les conditions de « mise à un » et de « mise à zéro » du nœud *A* sont :

$$\begin{cases} S_{UP} = Cka.\overline{C} + Ckb.\overline{D} + \overline{B} \\ S_{DN} = Cka.C + Ckb.D + B \end{cases} \quad ?$$

En effectuant l'expansion de la boucle, nous obtenons les expressions :

$$\begin{cases} S_{UP} = Cka.\overline{C} + Ckb.\overline{D} + A.Res \\ S_{DN} = Cka.C + Ckb.D + \overline{A.Res} \end{cases} \quad ?$$

Ces deux expressions ne sont pas orthogonales. Une des raisons est l'existence d'une corrélation entre les deux entrées directes $\{Cka, Ckb\}$. Si nous appliquons la méthode de recherche des variables primaires (§4.4.5) nous obtenons l'ensemble $\{En, Sel, C, D, Res, A\}$.

L'expression du comportement du nœud *A* en terme de ces variables de cet ensemble regroupe toute l'information sur les corrélations entre les entrées directes du cône *A*, éliminant ainsi les faux conflits qui en découlent. Pour notre exemple, nous obtenons après l'expansion, les expressions suivantes :

$$\begin{cases} S_{UP} = \overline{Sel}.En.\overline{C} + Sel.En.\overline{D} + A.Res \\ S_{DN} = \overline{Sel}.En.C + Sel.En.D + \overline{A.Res} \end{cases} \quad ?$$

La vérification de l'orthognalité donne le résultat suivant :

$$\begin{aligned} S_{UP}.S_{DN} &= (\overline{Sel}.En.\overline{C} + Sel.En.\overline{D} + A.Res).(\overline{Sel}.En.C + Sel.En.D + \overline{A.Res}) \\ &= \overline{Sel}.En.\overline{C}.\overline{A} + \overline{Sel}.En.\overline{C}.\overline{Res} + Sel.En.\overline{D}.\overline{A} \\ &+ Sel.En.\overline{D}.\overline{Res} + \overline{Sel}.En.C.A.Res + Sel.En.D.A.Res \end{aligned}$$

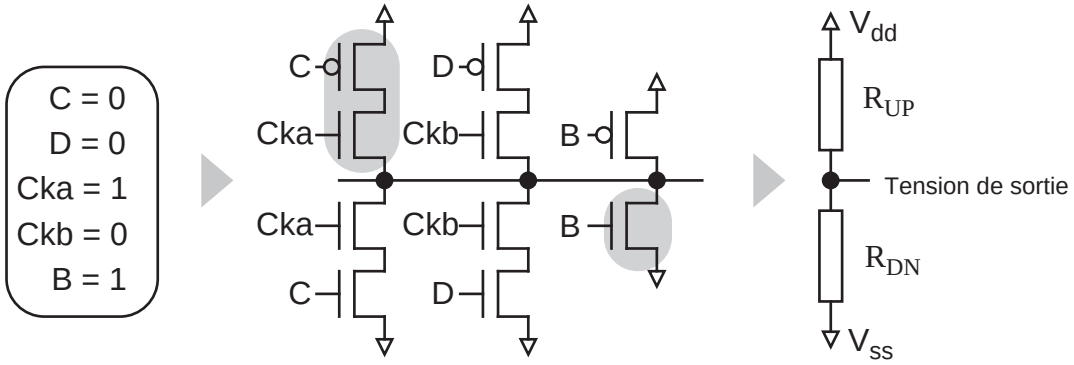
Il existe donc d'autres conflits qui ne sont pas dus à l'existence des corrélations.

5.4.2.2 Les faux conflits électriques

Dans notre exemple, la deuxième source de conflit est due à l'inverseur de maintien dans le rebouclage. Ce rebouclage agit comme un émetteur permanent sur le nœud. Dans un latch à conflit, on suppose que lors de l'écriture, un autre émetteur – plus puissant – entre en conflit avec le rebouclage et impose sa valeur.

La résolution de ce conflit est de type électrique, et une analyse du contexte logique ne permet pas de l'éliminer. La détermination de la valeur qui s'imposera à l'issue du conflit passe donc par une analyse électrique de l'ensemble des pilotes capables d'imposer une valeur.

Dans le modèle des cônes, on peut considérer chaque branche d'un cône comme un émetteur distinct. Pour un cône présentant des conflits électriques, il existe des combinaisons d'entrées qui rendent des branches V_{dd} passantes en même temps que des branches V_{ss}. La Figure 5-8 montre une situation conflictuelle dans le cas de notre exemple.



a) Configuration conflictuelle b) Les branches acitivées c) Pont diviseur équivalent

Figure 5-8 : *La résolution des conflits électriques*

Pour analyser ces conflits, il faut examiner chaque configuration des entrées qui provoque un conflit. Pour chaque configuration, nous pouvons considérer l'ensemble des branches actives comme un pont diviseur de potentiel. L'ensemble des branches vers V_{dd} est représenté par une résistance équivalente entre la sortie et V_{dd}. De même, les branches V_{ss} sont

modélisées par une résistance équivalente entre la sortie et V_{ss}. Evaluer la tension du nœud de sortie revient alors à calculer le rapport entre les résistances reliées à V_{dd} et à V_{ss}, soit :

$$V_{out} = (V_{dd} - V_{ss}) \frac{R_{DN}}{R_{UP} + R_{DN}}$$

- Si V_{out} est suffisamment proche de V_{ss}, on considère qu'on a un état bas.
- Si V_{out} est suffisamment proche de V_{dd}, on considère qu'on a un état haut.
- Sinon, on a un vrai conflit.

On a donc trois cas :

1) $\frac{R_{DN}}{R_{DN} + R_{UP}} < T$ état bas

2) $\frac{R_{DN}}{R_{DN} + R_{UP}} > 1 - T$ état haut

3) $T < \frac{R_{DN}}{R_{DN} + R_{UP}} < 1 - T$ conflit

Le paramètre T est défini par l'utilisateur, une valeur typique est 0,2.

Ainsi nous implémentons la fonction de décision de la Figure 5-9.

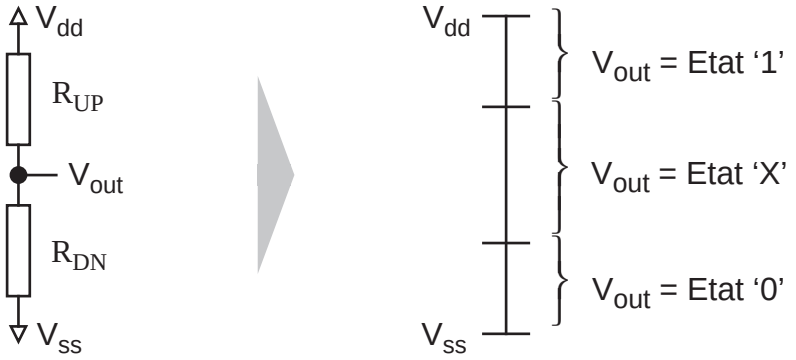


Figure 5-9 : *La fonction de décision*

Nous considérons que la résistance totale d'une branche est équivalente à la somme des résistances des transistors dont elle est composée. La résistance de chaque transistor est déterminée à partir des dimensions de son canal :

$R_N \propto L/W \times \mu_n$ pour un transistor de type N

$R_P \propto L/W \times \mu_p$ pour un transistor de type P

où μ_n et μ_p sont respectivement la mobilité des électrons et des trous.

Cependant, nous avons vu que, dans le modèle des cônes, un même transistor peut apparaître dans plusieurs branches. Si pour une certaine configuration des entrées qui provoque un conflit, plusieurs branches sont rendues passantes et si ces branches présentent des transistors partagés alors le calcul de la résistance est faussé. Pour simplifier le traitement, on fait l'approximation suivante :

- Pour chaque configuration des entrées créant un conflit, on ne considère que les deux branches V_{ss} et V_{dd} les moins résistives.

La Figure 5-10 donne le résultat de l'application de cet algorithme sur une des configurations des entrées qui produit un conflit dans notre exemple (voir Figure 5-8).

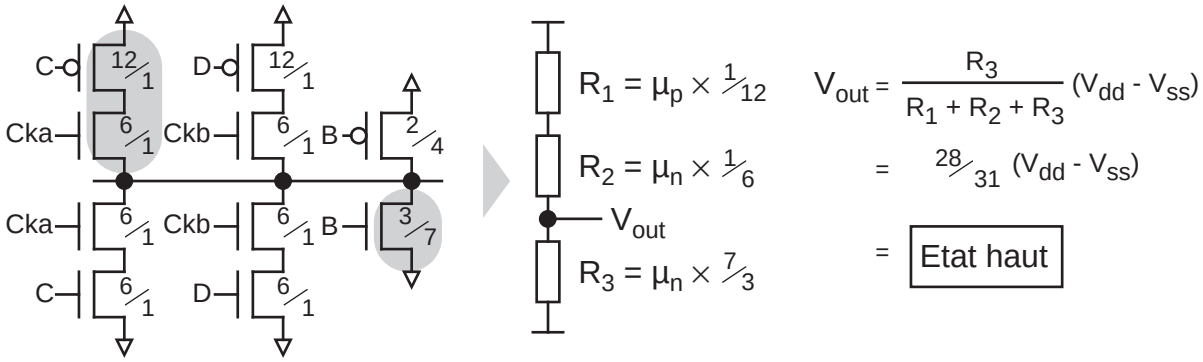


Figure 5-10 : *Le calcul de la résistance*

Ainsi, pour chaque configuration des entrées où il existe un conflit électrique, l'évaluation de résistances permet d'indiquer s'il s'agit d'un vrai ou d'un faux conflit, et lorsqu'il s'agit d'un faux conflit de déterminer la valeur logique du nœud de sortie.

Par la suite, les configurations pour lesquelles la résolution indique que la valeur issue du conflit est un '1' logique doivent être éliminés de la condition de « mise à zéro ». Inversement, lorsque le résultat de la résolution du conflit est un '0' logique, la configuration doit être éliminée de la condition de « mise à un ».

Cette technique de résolution des « faux conflits électriques » est indispensable pour traiter le cas des latchs à conflits, mais elle est très générale et permet, par exemple, de traiter le cas des circuits de type « ou cablé ».

5.4.2.3 Les vrais conflits

Après avoir éliminé les faux conflits fonctionnels et les faux conflits électriques, il se peut que d'autres conflits subsistent. Nous appelons ces conflits les vrais conflits. L'existence de ces conflits peut être liée à plusieurs raisons :

1) Corrélations non détectées : la profondeur choisie pour la construction des graphes de dépendances locales peut ne pas être suffisante pour identifier tous les faux conflits fonctionnels.

2) Contraintes externes : en général, l'abstraction fonctionnelle se fait de manière hiérarchique, en traitant le circuit bloc par bloc. Alors, il peut arriver, que les corrélations qui permettent de résoudre un faux conflit trouvent leur origine à l'extérieur du bloc. Dans ce cas, il se peut que certains conflits non fonctionnels soient considérés comme des vrais conflits.

3) Des erreurs de conception : un vrai conflit peut être dû à une vraie erreur de conception : on génère alors un message erreur.

5.4.2.4 Algorithme de résolution des conflits et des hautes impédances

Le point de départ est l'expression S_{Conf} qui représente toutes les conditions pour lesquelles un conflit existe. Cette expression est donnée par :

$$S_{Conf} = S_{UP}.S_{DN}$$

On exprime S_{Conf} en fonction des variables directes (sans expansion) comme une somme de mintermes. Dans le cas de notre exemple, nous obtenons :

$$\begin{aligned} S_{UP}.S_{DN} &= (Cka.\overline{C} + Ckb.\overline{D} + \overline{B}).(Cka.C + Ckb.D + B) \\ &= Cka.Ckb.\overline{C}.D.\overline{B} + Cka.Ckb.\overline{C}.D.B + Cka.\overline{Ckb}.\overline{C}.\overline{D}.B + ... \end{aligned}$$

On étudie ensuite chaque minterme indépendamment en le ré-exprimant en fonction des variables primaires.

Si l'expansion indique que l'expression obtenue est identiquement nulle il s'agit d'un faux conflit fonctionnel. Sinon, il peut s'agir d'un faux conflit électrique. Dans ce cas,

l'analyse des résistances des branches actives pour chaque configuration permet de déterminer le résultat du conflit.

Ainsi, nous arrivons à distinguer trois expressions mutuellement exclusives dans la condition de conflit.

$$S_{Conf} = S_{UP}.S_{DN} = S_{Conf-up} + S_{Conf-dn} + S_{Conf-x}$$

$S_{Conf\text{-}up}$ indique les faux conflits électriques dont le résultat est un '1'.

$S_{Conf\text{-}dn}$ indique les faux conflits électriques dont le résultat est un '0'.

$S_{Conf\text{-}x}$ exprime les conditions d'un vrai conflit.

Cette dernière expression est utilisée pour générer dans le modèle comportemental des messages de mise en garde (des assertions en VHDL).

L'application sur notre exemple donne les résultats indiqués par les matrices de la Figure 5-11 où chaque ligne correspond à une configuration des entrées directes. La Figure montre aussi comment nous générons l'expression des conflits résolus.

B	*C*	*D*	*Cka*	*Ckb*	V_{out}
1	0	0	1	0	1
1	0	1	1	0	1
1	0	0	0	1	1
1	1	0	0	1	1
0	1	0	1	0	0
0	1	1	1	0	0
0	0	1	0	1	0
0	1	1	0	1	0

a) Enumération des mintermes possibles de $S_{UP}.S_{DN}$

Résolution locale $S_{Conf\text{-}up}$ =

$B.\overline{C}.\overline{D}.Cka.\overline{Ckb}$

+ $B.\overline{C}.D.Cka.\overline{Ckb}$

+ $B.\overline{C}.\overline{D}.\overline{Cka}.Ckb$

+ $B.C.\overline{D}.\overline{Cka}.Ckb$

$= B.(\overline{C}.Cka.\overline{Ckb} + \overline{D}.\overline{Cka}.Ckb)$

b) L'expression des conflits résolus (résultat 1)

Figure 5-11 : *Résultat de la résolution des conflits*

De la même manière que pour les conflits, on peut exprimer la condition de haute impédance :

$$S_{HZ} = \overline{S_{UP}}.\overline{S_{DN}}$$

Un traitement similaire permet d'éliminer les cas de haute impédance non fonctionnels. Comme dans le cas des vrais conflits, les vrais cas de haute impédance peuvent être dus à une profondeur insuffisante, à des contraintes externes ou à une erreur de conception. Un message

de mise en garde est donc généré dans le modèle comportemental pour les vrais cas de haute impédance pour attirer l'attention du concepteur.

5.4.2.5 Génération de l'expression globale

La dernière étape, avant la phase d'analyse de la boucle, est l'accumulation de tous les résultats précédents dans une seule expression. Cette expression est établie en supposant que les conditions d'un vrai conflit ou d'une vraie haute impédance ne sont pas réunies.

Ainsi, nous obtenons l'expression du nœud de la boucle qui a été virtuellement coupé en fonction des variables primaires :

$$S = S_{UP}.\overline{S_{DN}} + S_{Conf-up}$$

Pour l'exemple de la Figure 5-5, nous avons :

$$\begin{cases} S_{UP} = Res.A + En.(Sel.\overline{D} + \overline{Sel}.\overline{C}) \\ S_{DN} = \overline{Res.A} + En.(Sel.D + \overline{Sel}.C) \end{cases} \quad ?$$

Et

$$S_{Conf-up} = B.(\overline{C}.Cka.\overline{Ckb} + \overline{D}.\overline{Cka}.Ckb) = \overline{Res.A}.En.(Sel.\overline{D} + \overline{Sel}.\overline{C})$$

Alors

$$S = Res.A.(\overline{En} + Sel.\overline{D} + \overline{Sel}.\overline{C}) + En.(Sel.\overline{D} + \overline{Sel}.\overline{C})$$

5.4.3 L'analyse de la boucle

Dans cette section, nous exposons la méthode d'analyse de la boucle. Cette analyse permet de déterminer si la boucle correspond effectivement à un élément séquentiel ou pas. Dans un premier temps, nous introduisons la notion de dérivée partielle dans l'algèbre de Boole. Puis nous détaillons notre méthode utilisant les dérivées partielles.

5.4.3.1 La dérivée partielle

Dans l'algèbre classique, la dérivé partielle d'une fonction à plusieurs variables *F*, par rapport à une variable donnée, est la fonction qui représente la variation de la fonction *F* par rapport à cette variable. Dans l'algèbre booléenne, l'équivalent de cette notion de variation est la sensibilité.

Une fonction booléenne est dite ***sensible*** à une variable x si un changement de la valeur de x provoque un changement de la valeur de la fonction. La dérivée partielle d'une fonction par rapport à une variable représente la condition de la sensibilité de la fonction à cette variable.

Soit $F(x_0, \ldots, x_n)$ une fonction booléenne des variables $x_0, \ldots, x_n$. La décomposition de Shannon de F suivant x_i permet d'exprimer F en fonction de deux fonctions uniques H et L telles que H et L soient indépendantes de x_i :

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

La fonction H est obtenue à partir de F en imposant $x_i = 1$. De même la fonction L est obtenue en imposant $x_i = 0$.

- Ainsi, nous notons L par $F_{|x_i=0}$ et H par $F_{|x_i=1}$
- La dérivée partielle de F par rapport à x_i est notée $\frac{\partial F}{\partial x_i}$ et son expression est :

$$\frac{\partial F}{\partial x_i} = F_{|x_i=1} \oplus F_{|x_i=0}$$

Cette expression représente le ou-exclusif des deux cofacteurs de la fonction F par rapport à x_i.

Il est clair que si l'évaluation des deux cofacteurs donne la même valeur, alors l'évaluation de la fonction F est indépendante de x_i. Ainsi la fonction F est sensible à la variable x_i si et seulement si l'évaluation des cofacteurs donne des valeurs différentes.

Considérons la dérivée partielle de F par rapport à x_i :

$$\frac{\partial F}{\partial x_i} = F_{|x_i=1} \oplus F_{|x_i=0} = F_{|x_i=1}.\overline{F_{|x_i=0}} + \overline{F_{|x_i=1}}.F_{|x_i=0}$$

- Nous appelons la partie positive de $\frac{\partial F}{\partial x_i}$, notée $\frac{\partial F^+}{\partial x_i}$, la fonction $F_{|x_i=1}.\overline{F_{|x_i=0}}$
- De même la fonction $\overline{F_{|x_i=1}}.F_{|x_i=0}$ est dite la partie négative de $\frac{\partial F}{\partial x_i}$, et est notée $\frac{\partial F^-}{\partial x_i}$

ainsi $\frac{\partial F}{\partial x_i} = \frac{\partial F^+}{\partial x_i} + \frac{\partial F^-}{\partial x_i}$

Ces deux fonctions sont mutuellement exclusives.

- Si pour une certaine configuration de $x_0, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$ nous avons :

$$\frac{\partial F^+}{\partial x_i} = 1$$

Alors un changement de la valeur de x_i provoque un changement de la valeur de F dans le même sens. On dit que F est ***directement sensible*** à x_i.

- Inversement, si pour une certaine configuration des variables nous avons :

$$\frac{\partial F^-}{\partial x_i} = 1$$

Alors une transition de x_i provoque la transition inverse de F. On dit que F est ***inversement sensible*** à x_i.

5.4.3.2 Les conditions de mémorisation et d'instabilité

Dans la section §5.4.2.5 nous avons établi l'expression du nœud de la boucle. La particularité de cette expression est que son support contient le nœud lui-même. En effet, l'existence de cette dépendance est la manifestation de l'existence d'une boucle.

Si on note a le nœud de la boucle :

$$a = F(a, x_0, \ldots, x_n)$$

où $a, x_0, \ldots, x_n$ représente les variables primaires du nœud a.

La dérivée partielle de F par rapport à a possède une signification particulière. Elle exprime la sensibilité du nœud par rapport à un changement de son propre état. Si, pour une certaine configuration des variables primaires, la valeur de cette dérivée est égale à zéro alors le nœud est insensible à son état actuel : ceci correspond à l'écriture d'une nouvelle valeur.

Si, pour une certaine configuration des variables primaires, la valeur de la dérivée partielle est égale à 1, alors le nœud est sensible à son état actuel à travers la boucle. Cependant, une sensibilité n'implique pas forcément un effet de mémorisation. Si le nœud est inversement sensible à lui-même, alors un changement de la valeur du nœud provoque une transition inverse. Autrement dit, la boucle est un oscillateur. Par contre si le nœud est directement sensible à lui-même, la boucle maintien l'état du nœud. Il s'agit alors d'une boucle de mémorisation. La Figure 5-12 résume les différentes situations.

$\frac{\partial F^+}{\partial a}$	$\frac{\partial F^-}{\partial a}$	$\frac{\partial F}{\partial a}$	Etat
0	0	0	La valeur de *a* est imposée par les autres variables
1	0	1	Boucle mémorisation de *a*
0	1	1	Oscillation de *a*

Figure 5-12: *Table de vérité de la dérivée partielle*

Si la partie négative de la dérivée partielle n'est pas identiquement nulle alors il existe des configurations pour lesquelles la boucle peut entrer en oscillation. Nous considérons qu'il s'agit d'une erreur de conception.

Pour qu'une boucle soit identifiée comme un latch, il faut que :

1) La partie négative de la dérivée partielle soit identiquement nulle.
2) La partie positive de la dérivée partielle ne soit ni identiquement nulle, ni identiquement 1. C'est-à-dire, il faut des configurations pour lesquelles la boucle soit en mémorisation et d'autres configurations pour lesquelles la boucle soit en écriture.

La condition d'écriture est l'inverse de la dérivée partielle.

Nous pouvons illustrer cette analyse sur notre exemple. Le point de départ est l'expression globale obtenue dans §5.4.2.5 :

$$S = Res.A.(\overline{En} + Sel.\overline{D} + \overline{Sel}.\overline{C}) + En.(Sel.\overline{D} + \overline{Sel}.\overline{C})$$

Nous souhaitons vérifier la sensibilité par rapport à la variable *A*, alors

$$S_{|A=0} = En.(Sel.\overline{D} + \overline{Sel}.\overline{C})$$

$$S_{|A=1} = Res.(\overline{En} + Sel.\overline{D} + \overline{Sel}.\overline{C}) + En.(Sel.\overline{D} + \overline{Sel}.\overline{C})$$

Ce qui nous donne les conditions suivantes :

$$\frac{\partial S^-}{\partial A} = S_{|A=0} \cdot \overline{S_{|A=1}} = 0$$

$$\frac{\partial S^+}{\partial A} = \overline{S_{|A=0}} \cdot S_{|A=1} = Res.\overline{En}$$

Cette analyse confirme que notre exemple se comporte comme un élément mémorisant. De plus, si nous prenons l'inverse de la condition de mémorisation, nous obtenons :

$$\text{La condition d'écriture} = S_{Ecrit} = \overline{Res} + En$$

Cette condition correspond exactement à la condition d'écriture dans le latch.

5.4.4 Conclusion

Nous avons proposé une méthode générale pour construire une expression du comportement caractéristique d'une boucle. Cette expression tient compte du contexte logique dans lequel se trouve la boucle, ainsi que la puissance relative des différents émetteurs susceptibles d'imposer un état sur la boucle.

Une méthode faisant appel à la dérivée partielle permet d'analyser si une boucle à une capacité de mémorisation et de vérifier sa stabilité.

5.5 La modélisation automatique

5.5.1 Introduction

Après avoir confirmé que la boucle correspond effectivement à un élément séquentiel, il reste à convertir l'expression caractéristique de la boucle en un modèle comportemental. Ce modèle doit permettre de distinguer les caractéristiques principales telles que la condition d'écriture, la donnée à écrire ou une condition d'initialisation asynchrone.

Dans cette section, nous allons expliquer la méthode que nous avons mise au point pour générer automatiquement ce modèle. Comme pour la technique d'analyse de la boucle, elle repose sur l'utilisation des opérations booléennes. Ainsi, grâce à cette généralité, cette méthode permet la modélisation de structures inconnues.

5.5.2 Objectifs de la modélisation

Dans la modélisation du comportement du latch, le premier problème consiste à identifier le support du modèle, c'est-à-dire la liste des variables dont dépendent les expressions qui font partie du modèle.

Dans le contexte de l'analyse d'une boucle, nous avons identifié deux ensembles de variables susceptibles d'exprimer le comportement du latch : les entrées directes du cône et les variables primaires. Si nous prenons le cas de notre exemple, nous avons les deux ensembles :

Entrées directes : $\{C, D, Cka, Ckb, B\}$

Variables primaires : $\{C. D, En, Sel, Res, A\}$

Le premier ensemble présente l'avantage de permettre l'expression de la fonctionnalité sans incorporer les portes en amont. Cependant, la présence de la boucle pose un problème : dans cette liste apparaît le deuxième nœud de la boucle, le nœud *B*. Ce problème n'apparaît pas si on utilise les variables primaires.

La solution que nous avons choisie consiste à exprimer les fonctions booléennes par rapport à un troisième ensemble de variables, que nous appelons ***les variables du modèle***. Nous considérons que l'élément séquentiel est composé du cône construit sur le nœud mémorisant et du cône de rebouclage. Ainsi, l'ensemble des ***variables du modèle*** s'obtient en remplaçant dans la liste des entrées directes, le second nœud de la boucle par les nœuds dont il dépend directement.

Il est alors possible d'obtenir une expression en termes du nouvel ensemble simplement en remplaçant le second nœud de la boucle par son expression. Notons que, comme pour l'expression globale, elle dépend de l'état du nœud mémorisant.

Dans le cas de notre exemple, nous avons :

Variables du modèle = $\{C, D, Cka, Ckb, Res, A\}$

$$S = \overline{B}.\overline{Ckb}.D.\overline{Cka}.C + \overline{C}.Cka.\overline{Ckb} + \overline{D}.\overline{Cka}.Ckb$$
$$= Res.A.\overline{Ckb}.D.\overline{Cka}.C + \overline{C}.Cka.\overline{Ckb} + \overline{D}.\overline{Cka}.Ckb$$

L'expression *S* modélise correctement l'élément séquentiel, toutefois pour des raisons de lisibilité et des contraintes liées aux outils de synthèse, il est préférable de distinguer les conditions d'écriture et les données à écrire.

5.5.3 Identification des commandes

A partir de la condition d'écriture, nous cherchons à identifier un ensemble d'émetteurs et d'associer à chacun une condition d'émission. La solution que nous proposons est de considérer que chaque minterme qui satisfait la condition d'écriture est potentiellement la condition d'émission d'un émetteur.

La première difficulté est d'obtenir la condition d'écriture en fonction des variables du modèle. L'expression obtenue dans §5.4.3.2 pour cette condition est en termes des variables primaires. Nous allons donc identifier, parmi l'ensemble des variables du modèle, un sous-ensemble qui permet d'exprimer la condition d'écriture. Nous appelons ce sous-ensemble, l'ensemble des commandes.

Nous identifions une commande par le critère suivant : une variable du modèle est considéré comme une commande si elle dépend d'au moins une des variables qui apparaissent dans la condition d'écriture.

Ce critère peut être exprimé formellement par la condition suivante :

Si on considère que $\{a, x_0, \ldots, x_n\}$ représente les variables primaires, et si G_V correspond à l'expression de la variable v du modèle en fonction des variables primaires, alors v est une commande si et seulement si :

$$\exists x_i \in \{x_0,\ldots,x_n\} \text{tel que} \left\{ \frac{\partial G_v}{\partial x_i} \neq 0 \wedge \frac{\partial S_{Ecrit}}{\partial x_i} \neq 0 \right\}$$

où S_{Ecrit} est la condition d'écriture exprimée en fonction des variables primaires.

Dans notre exemple, nous avons : $S_{Ecrit} = \overline{Res} + En$

Les variables primaires dont dépend S_{Ecrit} sont *Res* et *En*

$$\text{Autrement dit : } \frac{\partial G_{Ecrit}}{\partial x_i} \neq 0 \text{ Pour } x_i \in \{Res, En\}$$

Si maintenant nous considèrons les variables du modèle et leurs expressions en fonction des variables primaires nous obtenons :

Variable :	*C*	*D*	*Cka*	*Ckb*	*Res*	*A*
Expression :	*C*	*D*	$En.\overline{Sel}$	*En.Sel*	*Res*	*A*

Les variables qui dépendent de *En* et de *Res* sont donc : {*Cka*, *Ckb*, *Res*}

Cet ensemble est l'ensemble des commandes.

5.5.4 Construction de la liste des émetteurs

La dernière étape de la modélisation concerne l'identification des émetteurs. Dans la section précédente nous avons établi l'ensemble des commandes de l'élément séquentiel. Cet

ensemble correspond au support de la condition d'écriture en fonction des variables du modèle.

Nous considérons chaque minterme de l'ensemble des commandes. Premièrement, il faut vérifier que, compte tenu des corrélations, cette combinaison est possible. Cela se fait directement par une expansion du minterme en fonction des variables primaires. Deuxièmement, il faut vérifier que cette combinaison active la condition d'écriture. A cet effet, nous utilisons de nouveau l'expansion du minterme en fonction des variables primaires.

Le minterme active la condition d'écriture si son expansion est incluse dans la condition. Autrement dit, si on note par M l'expression du minterme en fonction des variables primaires, il faut vérifier que :

$$M.S_{Ecrit} = M$$

Dans ce cas, le minterme correspond effectivement à la condition d'émission d'un émetteur. Il reste à déterminer la valeur émise par cet émetteur. La valeur émise par l'émetteur est obtenue en imposant la condition d'émission de l'émetteur dans l'expression du nœud mémorisant (i.e. l'expression du nœud restreinte par l'expression du minterme).

Dans une ultime étape, les émetteurs qui émettent la même valeur sur le nœud sont regroupés en un seul émetteur. La condition d'émission de cet émetteur « équivalent » est la somme des conditions d'émissions des différents émetteurs.

Dans notre exemple, nous avons identifié trois commandes $\{Cka, Ckb, Res\}$. Il existe donc huit combinaisons possibles. Le tableau suivant montre l'analyse effectuée pour chaque minterme :

Combinaison :	Expansion *(G)*	$G.G_{Ecrit}$	Valeur :
1) $\overline{Cka}.\overline{Ckb}.\overline{Res}$	$\overline{En}.\overline{Res}$	$\overline{En}.\overline{Res}$	0
2) $\overline{Cka}.\overline{Ckb}.Res$	$\overline{En}.Res$	0	-
3) $\overline{Cka}.Ckb.\overline{Res}$	$En.Sel.\overline{Res}$	$En.Sel.\overline{Res}$	$\overline{D}$
4) $\overline{Cka}.Ckb.Res$	$En.Sel.Res$	$En.Sel.Res$	$\overline{D}$
5) $Cka.\overline{Ckb}.\overline{Res}$	$En.\overline{Sel}.\overline{Res}$	$En.\overline{Sel}.\overline{Res}$	$\overline{C}$
6) $Cka.\overline{Ckb}.Res$	$En.\overline{Sel}.Res$	$En.\overline{Sel}.Res$	$\overline{C}$
7) $Cka.Ckb.\overline{Res}$	0	-	-
8) $Cka.Ckb.Res$	0	-	-

Les mintermes 7 et 8 sont impossibles. Le minterme 2 n'active pas la condition d'écriture. Par ailleurs, on peut remarquer que les mintermes 3 et 4 correspondent à l'écriture de la même donnée. Ils peuvent donc être regroupés. De même les mintermes 5 et 6 sont regroupés dans un même émetteur.

Finalement, un modèle comportemental est généré pour chaque émetteur. La Figure 5-13 montre le VHDL généré dans le sous-ensemble de la chaîne ALLIANCE [Gre92].

```
label0 : BLOCK ((not (res) and not (ckb) and not (cka)) = '1')
  BEGIN
    mem <= GUARDED '0';
  END BLOCK label0;

label1 : BLOCK ((ckb and not (cka)) = '1')
 BEGIN
    mem <= GUARDED not (d);
  END BLOCK label1;

label2 : BLOCK ((not (ckb) and cka) = '1')
  BEGIN
    mem <= GUARDED not (c);
  END BLOCK label2;
```

Figure 5-13: *Exemple de VHDL généré*

5.6 Les limitations de l'approche

Cependant, cette approche comporte deux inconvénients qui se traduisent par des perspectives d'amélioration. Le premier est le manque d'une méthode générale pour regrouper plusieurs éléments séquentiels reconnus en un unique élément. Cette facilité est utile par exemple pour générer un modèle de bascule pour deux latchs en série. Cependant, il faut

remarquer qu'un regroupement automatique ne correspond pas toujours au souhait du concepteur.

Le deuxième inconvénient réside dans l'identification des éléments mémorisants dynamiques. Comme nous avons déjà dit dans l'introduction, il est intrinsèquement difficile de les distinguér des bus à trois états.

5.7 Conclusion

Dans ce chapitre, nous avons étudié une méthode totalement automatique pour l'identification et la modélisation des éléments séquentiels.

Nous avons détaillé une méthode formelle qui s'appuie sur l'analyse du réseau booléen comprenant une boucle compte tenu de son contexte logique. Cette méthode prend en compte les caractéristiques électriques des transistors (dans le cas des latchs à conflit). Elle permet de vérifier, indépendamment de la structure, si une boucle quelconque est effectivement une boucle de mémorisation, et d'obtenir les conditions de mémorisation.

Cette méthode d'identification générale est associée à une procédure de génération de modèle simulable, synthétisable, mais aussi lisible, de l'élément séquentiel. Nous employons le mot procédure plutôt que méthode car pour chaque élément séquentiel il existe plusieurs manières de décrire un modèle comportemental. Notre procédure permet de satisfaire en particulier l'exigence de lisibilité sans sacrifier la généralité.

Chapitre 6

ETUDE D'UNE APPROCHE HYBRIDE

Jusqu'à présent, nous avons mis l'accent sur une méthode de désassemblage générique et totalement automatique, sans bibliothèque de formes prédéfinies. Cependant, il existe des situations pour lesquelles la reconnaissance de formes est plus adaptée. Nous examinerons en particulier le traitement des circuits mixtes analogiques-numériques ainsi que les circuits pour lesquels le concepteur pourrait souhaiter contrôler la modélisation. Une approche hybride résoud ces problèmes tout en gardant les avantages de la méthode générique.

6.1 Introduction

Dans les chapitres précédents, nous avons étudié une méthode qui automatise totalement la tâche du désassemblage et de l'abstraction fonctionnelle. L'apport principal de cette méthode est la possibilité de traiter des circuits comportant des structures inconnues sans exiger pour autant une grande expertise de la part de l'utilisateur.

Toutefois, il est parfois intéressant d'offrir plus de contrôle au concepteur, en particulier en ce qui concerne la modélisation comportementale de structures particulières. Les modèles générés automatiquement, ne sont pas toujours adaptés à l'utilisation qu'on souhaite en faire : synthèse, preuve formelle, etc. De plus, les fonctions analogiques ne peuvent être traitées par la méthode des cônes.

Dans ce chapitre, nous présentons une méthode par reconnaissance de formes susceptible de s'articuler efficacement avec l'approche de partitionnement automatique en cônes. La méthode possède les avantages de GROG (voir §3.3.5) du point de vue de flexibilité de la bibliothèque et la possibilité de traiter des structures régulières paramétrables. Cependant, nous apportons l'avantage de l'utilisation du langage VHDL pour la définition des structures à reconnaître plutôt qu'un langage propriétaire.

Mais l'originalité la plus importante réside dans la cohabitation des deux approches. De ce fait l'utilisateur possède le degré de maîtrise qu'il souhaite sur la procédure d'abstraction. La dernière partie de ce chapitre concerne la mise en œuvre de cette approche hybride.

6.2 Motivation d'une approche hybride

L'avantage des approches par la reconnaissance de formes repose essentiellement dans le fait que le concepteur possède une maîtrise totale de la modélisation. Néanmoins, cet avantage est acquis au prix d'un travail important de la part du concepteur pour construire la bibliothèque des structures à reconnaître. Toutefois, nous pouvons identifier deux raisons principales pour lesquelles il est désirable de pouvoir effectuer la reconnaissance de formes.

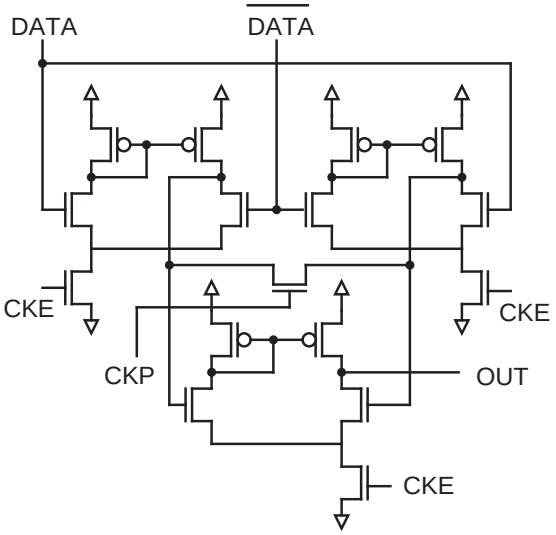
6.2.1 Les structures analogiques

La première raison est de permettre le traitement des circuits de type mixte. Avec la tendance actuelle d'intégrer le maximum de fonctions sur la puce, il devient courant de

trouver des circuits essentiellement numérique, mais qui contiennent des interfaces analogiques.

Pour calculer l'expression comportementale d'un cône, nous faisons l'hypothèse que la fonctionnalité d'un transistor correspond à celui d'un simple interrupteur. Dans le cas des circuits analogiques, cette hypothèse n'est plus valable.

La Figure 6-1 montre l'exemple d'un amplificateur différentiel, avec son modèle comportemental. Bien entendu, ce modèle est un modèle simplifié pour permettre une simulation au niveau RTL. Il ne tient pas compte des aspects analogiques, à savoir l'effet d'amplification permettant de détecter un signal différentiel très faible. En regardant la figure, il est clair que le découpage en cônes ne fournit aucune information qui permettra de générer automatiquement ce comportement. La seule méthode possible est la reconnaissance de cette structure pour lui associer un comportement voulu.




```
ARCHITECTURE behaviour_data_flow OF amp IS

BEGIN

  label0 : BLOCK (cke = '0')
  BEGIN
    dout <= GUARDED '0';
  END BLOCK label0;

  label1 : BLOCK (cke = '1')
  BEGIN
    dout <= GUARDED data;
  END BLOCK label1;

END;
```

Figure 6-1 : *Un amplificateur différentiel et son modèle comportemental*

Une motivation supplémentaire est l'apparition relativement récente de simulateurs capables de traiter des circuits mixtes. Dans ces simulateurs, les parties entièrement numériques du circuit sont traitées par une simulation événementielle à partir d'un modèle de type RTL, tandis qu'une simulation électrique de type SPICE est utilisée pour les parties analogiques. La description globale d'un circuit peut être composée de blocs des deux types.

Pour tirer le bénéfice de cette technologie, il est intéressant d'avoir des modèles analogiques pour les parties analogiques du circuit. Toutefois, pour des raisons d'efficacité de la simulation, un modèle analogique doit être utilisé seulement quand il est strictement

nécessaire. Ce découpage optimal entre les parties analogiques et les parties numériques n'est pas toujours disponible lorsqu'il s'agit d'une abstraction à partir des masques.

6.2.2 Les architectures régulières

La deuxième raison pour incorporer une méthode par reconnaissance de formes est liée à l'exploitation des structures régulières. En effet, il existe de nombreux circuits pour lesquels une partie importante de la complexité est due à une répétition d'une structure simple. La plupart des microprocesseurs actuels, qui possèdent des mémoires « cache ». entre dans ce cas de figure.

Il y a effectivement deux motivations pour lesquelles une approche hybride devient intéressante dans ce contexte. La première est liée à l'efficacité du déssasemblage. En effet, le traitement par une méthode d'analyse formelle des structures régulières telles que les RAMs peut s'avérer très lourd. La Figure 6-1 donne est un exemple typique du problème. On constate que chaque cellule peut écrire dans toutes les autres. C'est le fait qu'une seule commande soit active à la fois qui empêche cela. La méthode formelle est capable de démontrer les conditions sur les commandes, mais il est clair que, si un circuit contient une RAM de taille conséquente, une partie disproportionnée du temps d'exécution serait consacrée à cette analyse.

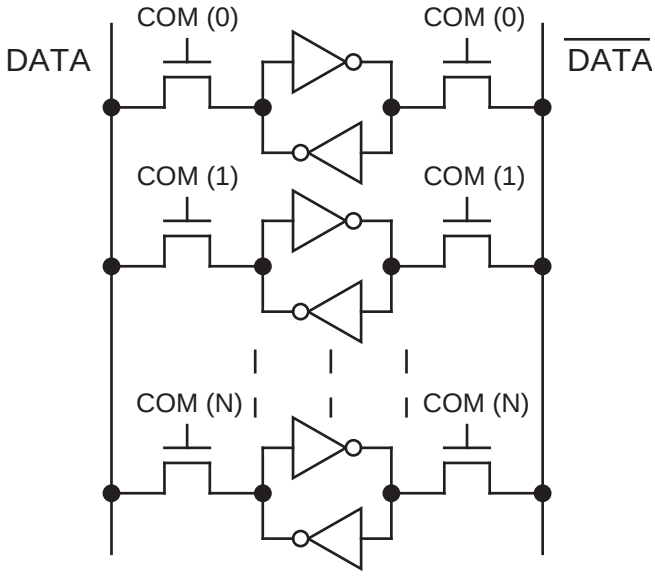


Figure 6-2 : *Une colonne de cellules RAM 1-bit*

La méthode automatique possède l'avantage d'effectuer une vérification formelle de la fonctionnalité de la RAM. Cependant, on peut souhaiter sacrifier la robustesse de cette approche formelle pour accélérer le traitement global du circuit.

L'inconvénient majeur d'une approche par la reconnaissance de formes est le fait que l'utilisateur doit fournir une bibliothèque des structures à reconnaître. Toutefois, dans le cas d'une structure régulière, cette tâche devient assez simple, car il n'y a qu'une seule cellule de base dont il faut définir la structure. La difficulté qui reste est de définir un langage qui offre la possibilité de décrire une structure contenant un nombre inconnu de cette cellule de base de manière répétitive.

La deuxième motivation, dans le contexte des structures régulières, est l'amélioration de la modélisation. Après avoir identifié une structure régulière, il doit être possible de générer un modèle comportemental bien plus compact que celui générer à partir du découpage en cônes. En plus d'une meilleure lisibilité, ce modèle sera plus efficace pour la simulation ou pour la synthèse.

6.3 Reconnaissance des structures élémentaires

Comme nous avons déjà dit dans le chapitre « Etat de l'Art », la base de toute approche de reconnaissance de formes est une méthode qui permet d'identifier un réseau de transistors à reconnaître au sein du réseau de transistors qui représente le circuit.

Cependant, dans le contexte d'une approche hybride, nous nous intéressons principalement à l'identification d'un petit nombre de structures. Il est alors bien plus efficace, pour la reconnaissance de ces structures, d'exploiter un algorithme qui tire partie des particularités de ces structures.

6.3.1 Rappel des méthodes

Dans « l'Etat de l'Art », nous avons détaillé deux algorithmes : BLEX [Lue84] et SubGemini [Ohl93] qui répondent à ce besoin. Les deux méthodes se base sur l'étiquetage des graphes bipartites qui représentent le circuit et le réseau à reconnaître. Il y a donc deux classes de nœuds : les nœuds-transistors et les nœuds-équipotentielle. La Figure 6-3 montre ce graphe pour un point mémoire de la RAM.

Les deux méthodes se distinguent par l'exploitation de cet étiquetage. Dans BLEX, les arcs du graphe sont étiquetés avec une valeur qui indique le type de connecteur du nœud-transistor auquel l'arc est relié. Il n'y a pas de ré-étiquetage après cette phase d'initialisation. Ainsi les étiquettes représentent la connectivité locale. Cette méthode d'étiquetage est efficace quand il s'agit de composants asymétriques comme les transistors bipolaires, mais il s'avère inadapté pour les transistors MOS qui sont des composants symétriques (voir la Figure 6-4).

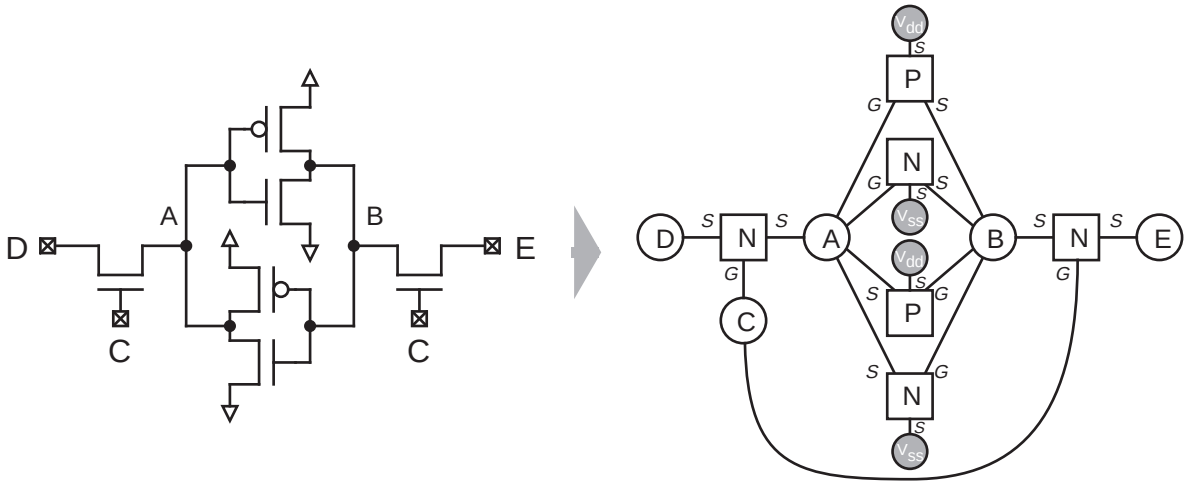
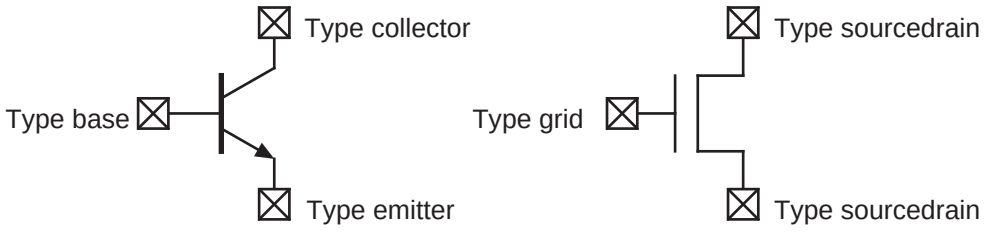


Figure 6-3 : *Le graphe bipartite d'une cellule de RAM*

La méthode de SubGemini est mieux adaptée. Après une phase d'initialisation des étiquettes, elle effectue plusieurs itérations pendant lesquelles l'étiquette d'un nœud influence la valeur d'une nouvelle étiquette pour chacun de ses voisins. Ainsi, les étiquettes reflètent des informations au-delà de la connectivité locale.



a) Transistor bipolaire :
3 connecteurs ⇒ 3 étiquettes

b) Transistor MOS :
3 connecteurs ⇒ 2 étiquettes

Figure 6-4 : *Etiquetage selon les connecteurs de transistors*

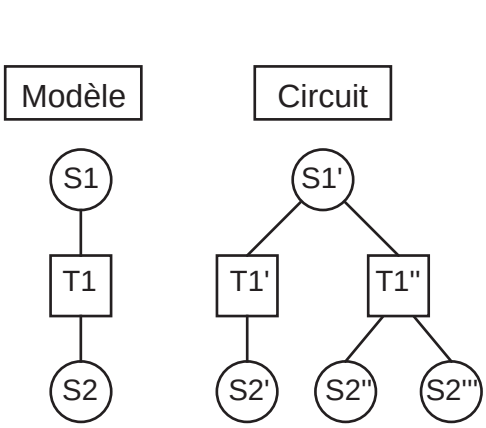
SubGemini se compose de deux phases de ré-étiquetage. Une première phase pour identifier des candidats et une deuxième phase pour vérifier chaque candidat. Son algorithme est très efficace dans la première phase. Le calcul de l'étiquette est peu coûteux et le nombre de calculs est linéaire par rapport à la complexité du circuit. L'efficacité de la deuxième phase

est plus discutable. En effet, l'information fournie par l'étiquette est trop pauvre pour confirmer ou réfuter une correspondance rapidement. De plus, il suffit que deux connecteurs externes du modèle à reconnaître soient reliés pour que la correspondance échoue.

6.3.2 L'algorithme d'isomorphisme de sous-graphe

Nous avons donc retenu la première phase de SubGemini pour la mise en œuvre de notre méthode de reconnaissance. Notre approche suit directement la description fournie dans §3.3.2.2 de « l'Etat de l'Art ». Cependant pour la deuxième phase, c'est-à-dire la vérification de chaque candidat, nous avons choisi de développer notre propre algorithme.

La motivation de la deuxième phase de SubGemini était d'effectuer implicitement un parcours *en largeur*, simultanément dans le circuit et dans le réseau de transistors. C'est un parcours implicite car il n'y a pas de mémorisation du chemin du parcours, à part l'étiquetage des nœuds, qui constitue une information appauvrie. Cet appauvrissement se traduit par des ambiguïtés, difficile et coûteux à gérer.



a) Graphes des débuts des parcours

Parcours modèle	S1	T1	S2		S5	T1	
Solution 1	S1'	T1'	S2'		S5'	T1'	√
Solution 2	S1'	T1'	S2"		S5"	T5'	×
Solution 3	S1'	T1"	S2'"		S5'"	T5"	×

b) Elimination par rebouclage du modèle

Parcours modèle	S1	T1	S2		S5	T1	
Solution 1	S1'	T1'	S2'		S5'	T5'	√
Solution 2	S1'	T1'	S2"		S5"	T1'	×

c) Elimination par rebouclage des solutions

Figure 6-5 : *Parcours en largeur*

Nous avons alors adopté un algorithme plus classique pour cette deuxième phase. Son objectif est d'examiner s'il existe un parcours dans le circuit, à partir d'un nœud-candidat, identique au parcours, à partir du nœud-clé, du réseau de transistors à reconnaître. La justification de cette approche est empirique. Elle se base sur le fait que nous nous situons dans un contexte hybride. En pratique, les éléments que nous cherchons à reconnaître sont essentiellement des éléments analogiques ou des éléments mémorisants. Ces deux types de structures partagent la particularité d'être fortement rebouclées (voir la Figure 6-3). Notre

méthode s'appuie donc sur la vérification qu'une boucle dans le réseau à reconnaître se retrouve dans le circuit.

Notre approche se base sur un parcours en *largeur*, pendant lequel on maintient une matrice dont chaque ligne correspond à une solution possible à l'état courant du parcours. Chaque colonne correspond à une étape du parcours. La Figure 6-5 montre l'évolution de cette matrice avec les étapes d'un parcours dans le graphe du modèle et le graphe du circuit.

Cette matrice de solutions possibles permet de détecter rapidement les boucles dans le circuit et dans les solutions. Ainsi, si le modèle possède une boucle, on supprime les solutions sans boucle identique. Inversement, si une solution possède une boucle, mais il n'y a pas de boucle correspondante dans le circuit, alors on supprime aussi la solution.

Ainsi cette méthode simple nous permet d'éviter de gérer des retours en arrière. La gestion de cette possibilité aurait été nécessaire avec un parcours en profondeur ou même avec un parcours implicite comme SubGemini. Cela est au prix de la consommation mémoire pour la matrice, cependant pour les circuits qui nous concernent, ce prix n'est pas élevé pour les structures que nous visons à reconnaître. Elle reste justifiable en vue du gain obtenu en temps d'exécution.

6.4 Reconnaissance hiérarchique

La deuxième partie de notre méthode de reconnaissance de formes concerne la reconstruction hiérarchique des structures complexes, à partir des structures élémentaires déjà reconnues. Notre intérêt principal est la reconnaissance des structures régulières paramétrables, composées d'un nombre limité de structures simples (souvent une seule) répétées.

Pour satisfaire ce besoin, il nous faut définir deux choses. Premièrement, du point de vue de l'utilisateur, il faut définir un langage qui permet de décrire des structures hiérarchiques régulières génériques. De plus, ce langage doit permettre d'associer des modèles comportementaux aux structures reconnues.

6.4.1 Un langage générique : GENIUS

Dans « l'Etat de l'Art », nous avons étudié deux outils de reconnaissance de formes : VERA et GROG, qui définissent leur propre langage de description de règles. Ces deux langages satisfont les besoins pour la reconnaissance des structures régulières paramétrées.

Cependant, seul le langage de GROG (GRL) permet d'attacher un comportement à une structure reconnue. C'est aussi un langage particulièrement riche en opérateurs, qui autorise des descriptions structurelles complexes.

Le langage GRL présente cependant deux faiblesses :

1) La description des structures régulières paramétrées s'appuie sur l'utilisation des règles récursives. La répétition n'est donc pas explicite, ce qui rend plus difficiles l'écriture et la compréhension d'une bibliothèque de règles.

2) Le langage GRL est un langage dédié. L'utilisation de GROG oblige donc une formation pour comprendre un nouveau langage qui ne ressemble pas aux autres langages utilisés au cours de la conception.

Ces inconvénients nous ont conduit à définir un nouveau langage de description de règles, appelé GENIUS qui utilise le VHDL structurel pour la description des structures régulières. Dans le langage VHDL, la description d'une structure se fait en deux parties :

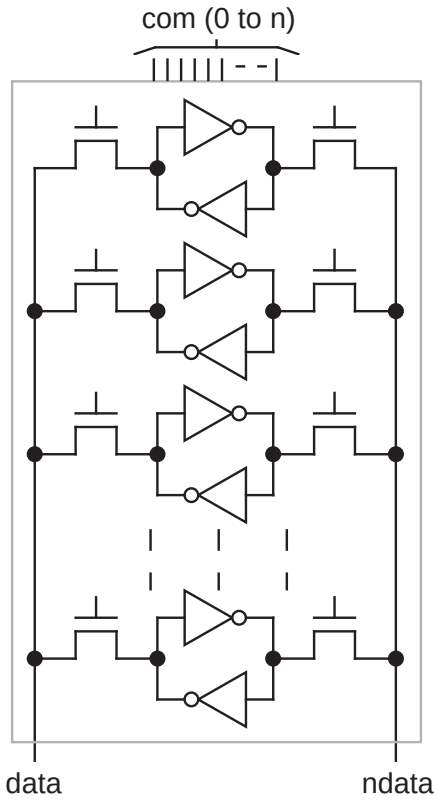
```
ENTITY unité abstraite IS
  <interface>
END;
ARCHITECTURE structure OF unité abstraite IS
  <composants>
begin
  <instances>
END ARCHITECTURE
```

La deuxième partie définit l'interconnexion des composants et représente la structure à reconnaître. La première partie définit l'interface de l'unité abstraite et permet donc son utilisation en tant que composant dans un niveau hiérarchique supérieur.

Le VHDL gère donc naturellement des descriptions structurelles hiérarchiques. Il reste le problème de description des structures régulières paramétrées. La répétition d'une structure peut se faire en VHDL à l'aide de la construction *GENERATE*. Cette construction s'utilise au sein d'une boucle, dont le nombre d'itérations s'exprime par une variable générique. Dans la sémantique de VHDL, il est supposé qui la valeur de cette variable est définie à un niveau supérieur de la description du circuit.

Pour exploiter cette construction dans le contexte de l'abstraction fonctionnelle. Nous devons attacher une sémantique légèrement différente aux variables génériques. Dans notre cas, ces variables représentent un nombre de répétitions inconnu, dont la valeur sera fixée une

fois que la structure sera reconnu. La Figure 6-6 montre l'exemple de la description générique d'une colonne de « bit-cells » pour une RAM.




```
entity RAM_column is
  generic (
    n        : integer);
  port (
    data     : inout bit;
    ndata    : inout bit;
    com      : in bit_vector (0 to n);
    vdd, vss : in bit);
end entity;

architecture structure of RAM_column is
  component bitcell
    port (
      q        : inout bit;
      nq       : inout bit;
      c        : in bit;
      vdd, vss : in bit);
  end component;

begin
  loop : for i in 0 to n generate
    cell_i : bitcell
      port map (data, ndata, com (i)
                vdd, vss);
  end generate;
end architecture;
```

Figure 6-6 : *VHDL pour les structures régulières*

La notion d'attribution d'une valeur nous emmène au second aspect : il faut permettre au concepteur de spécifier très précisement le modèle comportemental qui doit être associé à la structure reconnue. Bien qu'il soit possible d'exprimer le comportement directement en VHDL, cette solution a de nombreux inconvénients. Premièrement, il n'est pas toujours possible d'exprimer des comportements génériques pour les structures régulières paramétrées. En second lieu VHDL n'est pas adapté aux fonctions analogiques. Enfin, cette approche ne permet pas de créer des modèles dans des langages autres que le VHDL.

Nous avons donc décidé, en ce qui concerne la modélisation, d'utiliser le langage 'C' encapsulé dans une partie *ACTION* associée à la description de l'objet à reconnaître. La Figure 6-7 montre la modélisation de la colonne de RAM. Lors de la reconnaissance d'un modèle, la partie *ACTION* est exécutée et la génération du modèle consiste à l'écriture du texte dans un fichier. Les variables génériques se transforment en constantes dont la valeur est définie à la génération.

```
action model of RAM_column is
  open ("column.vbe")

  fprintf (file, "
entity RAM column is
  port (
    data     : inout bit;
    ndata    : inout bit;
    com      : in bit_vector (0 to %d);
    vdd, vss : in bit);
end entity;

architecture vbe of RAM_column is
  signal mem : reg_vector (0 to %d) register;
begin ", n-1, n-1);

  for (i = 0; i < n; i++) {
    fprintf (file, "
mem_%d : block ( com (%d) = '1' )
begin
  mem (%d) <= guarded data;
end block; ", i, i, i);
  }

  fprintf (file, "
with (com) select
 data <= ");

  for (i = 0; i < n; i++) {
    char *c = one_hot_to_string (n, i);
    fprintf (file, "  mem(%d) when \"%s\",", i, c);
  }

  fprintf (file, "end architecture;");
end action;
```

a) Le code ‘C’ de la partie *ACTION*

```
entity RAM column is
  port (
    data     : inout bit;
    ndata    : inout bit;
    com      : in bit_vector (0 to 31);
    vdd, vss : in bit);
end entity;

architecture vbe of RAM_column is
  signal mem : reg_vector (0 to 31) register;
begin

mem_0 : block ( com (0) = '1' )
begin
  mem (0) <= guarded data;
end block;
        ¦       ¦
        ¦       ¦
        ¦       ¦
mem_31 : block ( com (31) = '1' )
begin
  mem (31) <= guarded data;
end block;

with (com) select
 data <=
  mem(0) when "100000...",
  mem(1) when "010000...",
        ¦       ¦
        ¦       ¦
        ¦       ¦
end architecture;
```

b) Le code VHDL généré

Figure 6-7 : *L’utilisation du ‘C’ pour la modélisation*

La souplesse et la familiarité du langage ‘C’ permet à l’utilisateur de décrire facilement les opérations nécessaires pour la génération des modèles, même pour les modèles génériques. Ainsi le modèle peut être optimisé pour un outil de vérification ou de synthèse.

6.4.2 Ordonnancement de l’extraction

Un des points forts du langage GRL de GROG est la richesse de ses opérateurs de description des structures à reconnaître. Cela permet des descriptions complexes à chaque niveau. Cependant, cette souplesse possède l’inconvénient de rendre l’efficacité de la recherche sensible à la manière dont les règles sont écrites. Cette sensibilité complique la tâche du concepteur qui écrit les règles.

Le problème provient de l’ordonnancement des règles utilisées par le moteur de reconnaissance de formes (une règle correspond à une structure à reconnaître). L’organisation hiérarchique des structures à reconnaître impose un ordre partiel lié aux relations d’instanciations : dans notre exemple de la colonne de RAM, il faut avoir identifier le « bit-cell » avant de pouvoir identifier la colonne.

L’ordonnancement des règles est bien géré dans GROG. Nous avons donc adopté une solution similaire. Cela consiste à générer le graphe de dépendance des règles, comme dans

l'exemple de la Figure 6-8. Comme pour GROG, c'est un parcours en profondeur de ce graphe qui permet de trouver l'ordonnancement optimal des règles.

Toutefois, il reste un cas d'ambiguïté : c'est le cas d'une structure instanciée par plusieurs règles. Sur le graphe de dépendance, cela correspond à un nœud avec plusieurs prédécesseurs. En effet, il existe des parcours valides qui donne des résultats différents. Dans la Figure 6-8, si on active *R1* avant *R2*, la structure de *R2* ne sera reconnu que si *R1* échoue. Inversement, si on active *R2* avant *R1*, la structure de *R2* est reconnue en priorité. Cette ambiguïté arrive car la structure de *R4* possède *R1* et *R2* comme predécesseur. Afin de gérer cette situation, l'utilisateur peut associer une priorité à une règle.

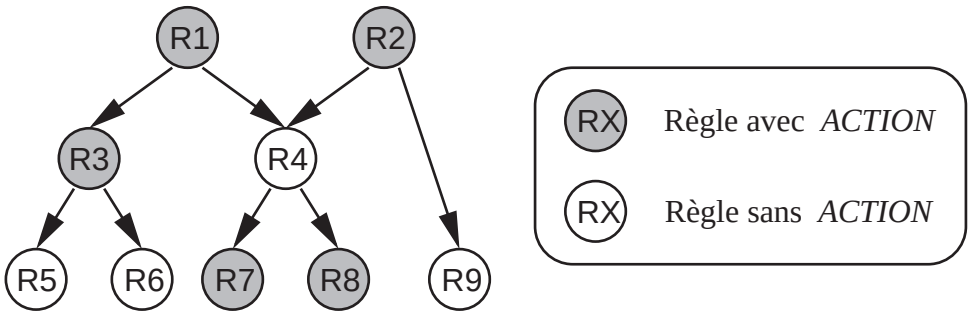


Figure 6-8 : *Ordonnancement de la reconnaissance*

Les règles peuvent être classé en deux type : les règles avec une partie *ACTION* et les règles sans partie *ACTION*. Une règle sans action n'a un sens que si la structure associée est nécessaire pour la reconnaissance d'une autre structure de plus haut niveau. Par exemple, les règles *R5* et *R6* de la Figure 6-8 n'ont qu'un sens que s'il existe la structure représenté par la règle *R3*. Si cette structure n'est pas reconnue, alors l'effet des règles *R5* et *R6* est invalidé.

Nous avons choisi ce comportement, qui se distingue du comportement de GROG car nous sommes dans le contexte d'une approche hybride. L'absence d'une partie action implique l'impossibilité d'associer un comportement pré-défini. Ainsi nous souhaitons que cette partie du circuit soit traitée par la méthode automatique.

Si une règle instancie plusieurs sous-structures, il faut également définir l'ordre dans lequel ces sous-structures vont être recherchées. Dans GROG, l'ordre de recherche de ces sous-structures nécessaires à la validation d'une règle se fait dans l'ordre de l'écriture de la règle. Afin d'optimiser une règle, il est alors nécessaire de connaître la fréquence relative des sous-structures recherchées.

Toutefois, la fréquence relative des structures dépend beaucoup sur le circuit, il est alors difficile de la connaître à priori. Nous avons donc choisi une stratégie de recherche qui modifie dynamiquement l'ordre de la recherche. En effet, l'ordre du parcours du graphe de dépendance des règles oblige à ce que toutes les sous-structures nécessaires à une règle soient reconnues lors de l'activation de cette règle. Pour chaque structure à reconnaître, nous maintenons un compteur de son nombre d'occurrences. Ainsi, l'ordre de la recherche peut être dynamiquement optimisé.

6.4.3 Méthode générale de reconnaissance

Il y a donc deux étapes dans la reconnaissance de formes. La méthode basée sur Sub-Gemini est utilisée pour la reconnaissance des structures représentées par les feuilles du graphe de dépendances des règles. Ces structures ne sont pas génériques et elles sont définies complètement par un réseau de transistors. La méthode d'étiquetage de SubGemini est particulièrement efficace en ce qui concerne l'isolation des bons candidats pour une correspondance dans le circuit. L'algorithme permet d'effectuer rapidement la reconnaissance au niveau du réseau de transistors, cependant il n'est pas capable de traiter des structures régulières génériques.

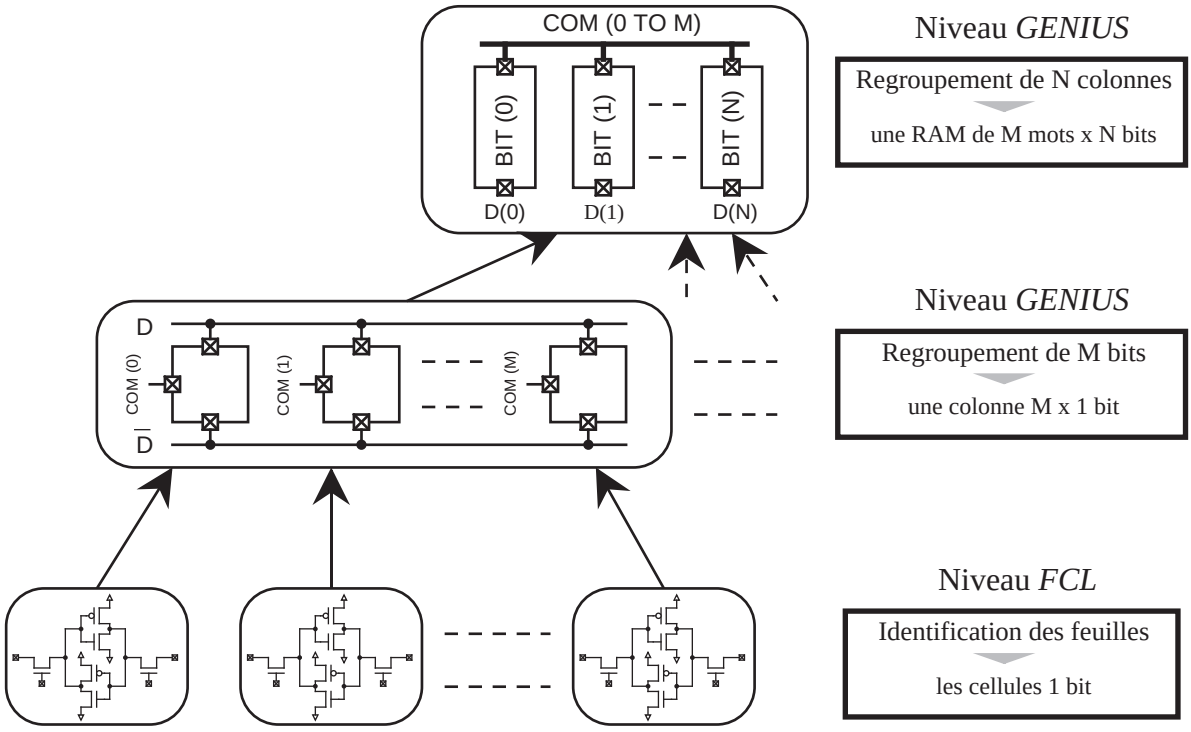


Figure 6-9 : *L'hiérarchie de la reconnaissance*

Les structures de plus haut niveaux sont identifiés par une méthode plus classique de recherche exhaustive qui prend en compte des structures régulières génériques. La méthode reste efficace car elle s'appuie sur les structures déjà identifiées par la méthode de SubGemini.

Quelle que soit la méthode de reconnaissance, la description des structures à reconnaître se fait à la même manière : en VHDL. Pour les structures définies par des interconnexions des transistors, nous avons deux modèles pré-définis qui représentent les deux types de transistors MOS.

A chaque structure reconnue, quel que soit son niveau hiérarchique, on associe une « boîte noire » avec l'interface du bloc et la liste des transistors y appartenant. Le réseau de transistors du circuit n'est pas modifié pendant la phase de reconnaissance. Cela permet de revenir en arrière sur une abstraction si on n'arrive pas à lui associer une *ACTION*.

6.5 Mise en œuvre de l'approche hybride

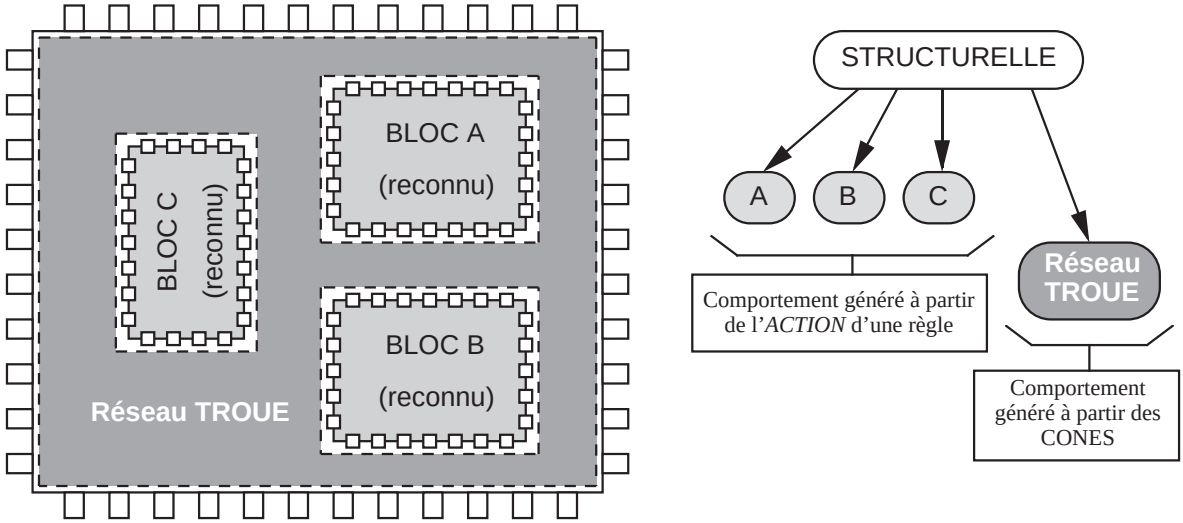
La mise en œuvre de cette approche hybride consiste à réunir deux stratégies distinctes d'abstraction fonctionnelle : la méthode de reconnaissance de formes et la méthode de partitionnement automatique.

L'articulation entre les deux approches est le point délicat : il y deux questions auxquelles il faut répondre. Premièrement, quelle est la méthode qu'il faut appliquer en premier. Deuxièmement, comment gérer l'interfaçage des deux méthodes.

Pour répondre à la première question, il faut regarder les motivations de l'approche hybride. L'objectif est d'utiliser la stratégie qui convient le mieux pour une partie du circuit donné. Pour la reconnaissance de formes, le concepteur à une influence, à travers les règles qu'il écrit, sur la partie du circuit qui est traitée. Si les structures élémentaires sont suffisament distinctive, leur identification est rapide et très peu influencée par la compléxité du circuit. Les structures que nous souhaitons reconnaître - essentiellement des composants analogiques ou des éléments mémorisants – satisfont bien ce critère.

La phase de reconnaissance de formes est alors effectuée avant la phase de partitionnement automatique, jusqu'à épuisement de toutes les règles. Le résultat de cette première phase est un ensemble de « boîtes noires » (ou d'instances) qui représentent les structures du plus

haut niveau auxquelles il est possible d'associer des modèles comportementaux. Notre objectif est alors de générer un modèle uniquement pour les transistors extérieurs aux instances reconnues.



a) Découpage du circuit b) Hiérarchie de la description

Figure 6-10 : *Le réseau de transistors troué*

Le deuxième problème est la réunion des deux stratégies, qui sert à définir l'approche hybride. Après la phase de reconnaissance, pour chacune des instances reconnues, nous disposons de son interface : c'est-à-dire la liste de ces connecteurs ainsi que les équipotentielles du circuit auxquelles ils doivent être reliés. Chaque instance est aussi associée avec la liste de transistors faisant partie de la structure reconnue. Ces informations nous permet de modifier le réseau de transistors du circuit pour retirer les transistors qui correspondent aux instances reconnues et déjà modélisées (voir la Figure 6-10a). L'algorithme de partitionnement en cônes est alors lancé sur ce nouveau réseau de transistors « troué » afin de générer son modèle comportemental. Le modèle final consiste alors à une description hiérarchique (voir la Figure 6-10b) qui regroupe les modèles des instances reconnues avec le modèle généré automatiquement.

6.6 Bilan sur le traitement des registres

Pour terminer ce chapitre, nous souhaitons faire une parenthèse pour attirer l'attention à l'ensemble des méthodes proposées pour le traitement des registres. En effet, nous mettons deux approches bien distinctes à la disponibilité du concepteur :

1) Une première méthode complètement automatique. Cette approche est totalement générique, ainsi elle permet l'identification et la modélisation des éléments séquentiels à priori inconnus. L'élaboration de cette méthode formelle (chapitre 5) constitue une des originalités de ce travail.

2) Une deuxième méthode par reconnaissance de formes. Nous avons introduit cette méthode dans ce chapitre dans le contexte de la reconnaissance des blocs de RAM. Toutefois la méthode permet le traitement des éléments séquentiels individuellement.

La première méthode offre l'avantage de simplifier la tâche du concepteur qui n'a pas à décrire les structures et les modèles des éléments séquentiels du circuit. Ainsi, il devient possible de traiter des circuits d'origines diverses sans adaptation d'une bibliothèque de formes. De plus, l'analyse générique, liée avec la modélisation automatique permet une vraie vérification, car les modèles sont générés à partir de la réalisation plutôt que la spécification.

L'utilisation de la deuxième méthode, la reconnaissance de formes, permet au concepteur d'exercer un contrôle sur la phase de modélisation. Cela permet de combler le défaut principal de la méthode générique, c'est-à-dire son incapacité de réunir deux latchs pour générer un modèle d'une bascule à échantillonage sur front.

Toutefois, le concepteur est libre de mélanger les deux méthodes pour un circuit donné, car elle sont complémentaires. En effet, si un élément mémorisant n'est pas traité par un modèle fourni par le concepteur, alors cet élément serait systématiquement traité par la méthode générique.

6.7 Conclusion

Dans ce chapitre, nous avons montré qu'il est possible de réunir la puissance d'un outil d'abstraction fonctionnelle reposant sur le partitionnement automatique avec la souplesse et la contrôlabilité d'une approche par reconnaissance de formes. Nous avons expliqué les étapes d'une méthode à la fois souple et efficace pour la reconnaissance de formes ainsi que son intégration au sein d'un prototype logiciel.

Pour justifier cette approche hybride nous avons cité deux situations pour lesquelles une méthode totalement automatique est au mieux mal adapté et au pire impossible. Ces deux cas sont les circuits mixtes analogique-numériques et les structures régulières de type mémoire.

Pour le premier cas, l'algorithme basé sur SubGemini est bien adapté à la recherche des composants analogiques. Ces composants sont, en général, composés d'un petit nombre de transistors, mais avec une interconnexion distinctive. C'est exactement le type de structure pour laquelle cet algorithme est optimisé.

Pour le deuxième cas, nous avons montré que le langage VHDL permet de décrire des structures hiérarchiques et génériques. En ce qui s'agit de la modélisation de ces éléments, la partie *ACTION* de la règle qui décrit le composant permet à l'utilisateur de facilement associer la description qu'il souhaite.

Chapitre 7

MISE EN OEUVRE LOGICIELLE

Dans ce chapitre, nous présentons la mise en œuvre du prototype logiciel qui nous a permis de valider les méthodes et les algorithmes présentés dans les chapitres antérieurs. Nous commençons par une présentation de la chaîne ALLIANCE, qui constitue la base de notre plate-forme d'expérimentation. Par la suite, nous montrons le contexte d'utilisation du désassemblage au sein de notre laboratoire, avant de présenter l'architecture de notre réalisation.

7.1 Introduction

La mise en œuvre logicielle a constitué une partie importante des travaux effectués dans le cadre du présent travail. Nous présentons l'architecture et apportons quelques précisions sur la rélisation du prototype logiciel YAGLE (Yet Another Gate-Level Extractor), développé dans le cadre de cette thèse.

Dans un premier temps nous présentons l'environnement de développement sur lequel nous nous sommes basés. Il s'agit principalement de la chaîne ALLIANCE. Parmi d'autres fonctionnalités, cette chaîne offre des bibliothèques de manipulation des expressions booléennes, indispensables pour la mise en œuvre des méthodes expliquées dans les chapitres précédents.

Puis, nous exposons la bibliothèque CNS (Cone Netlist Structure). Cette bibliothèque, ainsi que la structure de données qui lui est associée, ont été définies pour satisfaire les besoins du désassemblage et de la représentation des cônes.

7.2 La plate-forme ALLIANCE

ALLIANCE [Gre92] est un ensemble d'outils et de bibliothèques qui constituent une chaîne complète pour la conception des circuits intégrés. La réalisation d'un logiciel de CAO dans le contexte de cette plate-forme, bénéficie de plusieurs avantages. En contrepartie, elle doit obéir à des contraintes liées à l'intégration des logiciels d'origine diverse dans un environnement unique.

7.2.1 Un environnement de conception complet

Le premier avantage d'ALLIANCE est de fournir un ensemble d'outils nécessaires à la conception d'un circuit intégré. Pour la phase de conception proprement dite, ALLIANCE comprend :

1) Les outils de synthèse et de placement/routage ainsi que les cellules pour la conception des circuits à partir d'une bibliothèque de cellules précaractérisées.

2) Des générateurs paramétrables de blocs « full-custom » tels que des multiplieurs, des décaleurs et des bancs de registres.

3) Un générateur de chemin de données.

En ce qui concerne la vérification, la chaîne ALLIANCE dispose d'un ensemble d'outils nécessaires pour valider un circuit intégré. Elle comprend en particulier les quatre outils suivants :

LYNX Un outil d'extraction. A partir de la description physique, il extrait un réseau de transistors utilisé en entrée de YAGLE.

ASIMUT Un simulateur VHDL de type événementiel. Il permet de simuler une description RTL telle que le modèle comportemental généré par YAGLE.

PROOF Un outil de preuve formelle. Il permet de vérifier que deux circuits ont la même fonctionnalité, du moment que leurs entrées primaires (les registres et les connecteurs externes) sont identiques.

TAS Un outil d'analyse temporelle statique. Il s'appuie sur le réseau de cônes généré par YAGLE pour identifier les chemins critiques dans un circuit.

7.2.2 Des structures de données harmonisées

Tous les outils de la chaîne ALLIANCE s'appuient sur des structures de données communes. La manipulation de ces structures est facilitée par des bibliothèques de fonctions dédiées à chaque structure. Les trois structures les plus importantes représentent les trois « vues » traditionnelles d'un circuit.

BEFIG Cette structure représente la vue comportementale. Il s'agit d'un réseau booléen où, à chaque nœud, est associée une expression booléenne en fonction de ses prédécesseurs.

LOFIG Cette structure correspond à la vue structurelle. Ainsi elle permet de représenter des interconnexions de portes ou de transistors.

PHFIG Cette structure correspond à la vue physique. Il s'agit d'une représentation symbolique des masques.

C'est l'outil LYNX qui permet de générer la LOFIG représentant un réseau de transistors à partir de la PHFIG. L'abstraction fonctionnelle consiste à générer la BEFIG qui modélise le comportement de ce réseau de transistors. YAGLE génère cette structure à partir

du réseau de cônes obtenu dans la phase de désassemblage. La structure CNSFIG, que nous allons décrire dans §7.4, représente ce réseau de cônes.

7.2.3 Les Parsers / Drivers

Un autre avantage d'ALLIANCE réside dans la gestion des fichiers d'entrées / sorties. Chacune des structures de données est associée à un ou plusieurs parsers / drivers qui permettent de la traduire en une représentation textuelle et vice versa. En ce qui concerne YAGLE, l'utilisation de la LOFIG comme structure d'entrée fait que YAGLE accepte en entrée tous les formats de description de réseau de transistors supportés par ALLIANCE.

D'une manière similaire, la génération de la BEFIG permet de profiter du driver VHDL associé à cette structure pour créer le fichier VHDL en sortie.

7.3 La manipulation des expressions booléennes

Dans les phases de fabrication de cônes et de l'analyse des éléments séquentiels, les méthodes que nous proposons reposent sur la manipulation des expressions booléennes. Dans la chaîne ALLIANCE, deux bibliothèques de fonctions ont été développées pour permettre la représentation des expressions booléennes.

7.3.1 La représentation préfixée

La première est une représentation préfixée. Dans cette représentation, une expression booléenne est une liste dont le premier élément correspond à l'opérateur et les éléments suivants aux opérandes. Chaque opérande peut elle-même correspondre à une expression booléenne ou à un élément terminal (une variable). Cette représentation, que l'on appelle ***ABL***, est donc une représentation utilisant des listes hiérarchiques. Chaque niveau de cette liste correspond à une opération booléenne. La Figure 7-1 montre un exemple de cette représentation.

Cette représentation traduit bien la représentation textuelle d'une expression. Elle peut être facilement construite par les parsers et manipulée par les drivers. Cette aussi une représentation compacte qui s'avère très commode pour stocker des expressions.

Toutefois, avec une telle structure, il est difficile de définir des algorithmes généraux pour effectuer des manipulations booléennes. En premier lieu le manque de canonicité (une

même fonction peut avoir plusieurs représentations différentes) rend difficile les opérations essentielles, telles que la vérification d'équivalence et la satisfiabilité.

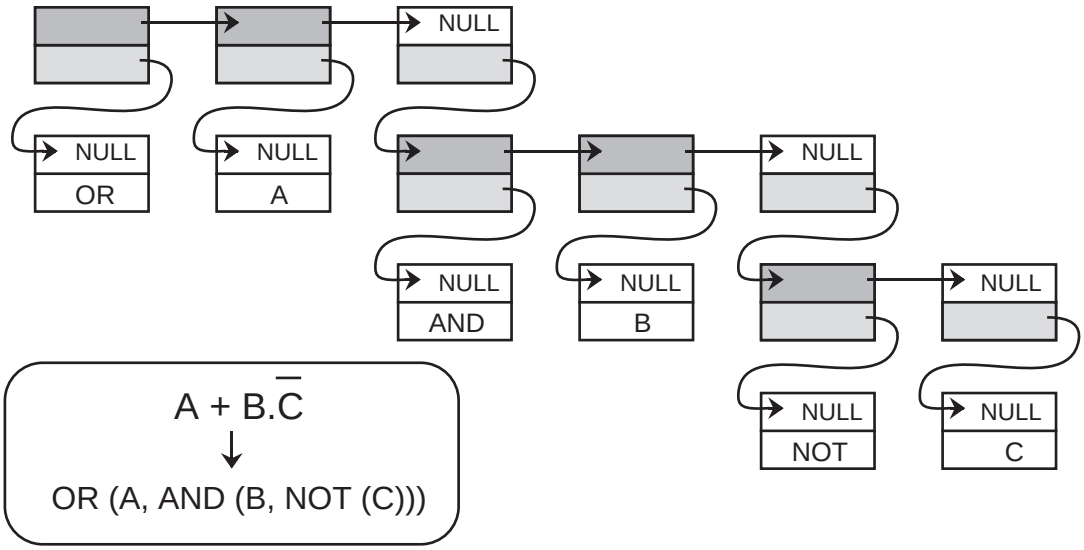


Figure 7-1 : *Exemple de représentation à l'aide d'ABL*

7.3.2 L'arbre de décision binaire

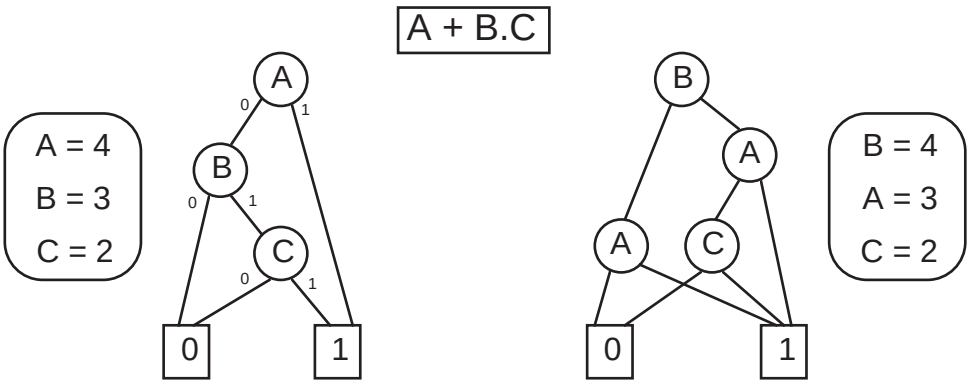
ALLIANCE dispose d'une seconde bibliothèque pour la représentation et la manipulation des expressions booléennes. Cette une représentation en arbre de décision binaire ***ROBDD*** (Reduced Ordered Binary Decision Diagram) (voir la Figure 7-2).

Cette représentation utilise la décomposition de Shannon. Dans cette représentation, une expression booléenne est un arbre où les nœuds terminaux représentent les constantes '0' et '1' et les nœuds non-terminaux représentent des décisions sur l'état d'une des variables dont dépend l'expression.

Les *ROBDD* sont une classe particulière des arbres de décision binaire. Ils obéissent à une contrainte supplémentaire. Chaque variable dont dépend la fonction est associée à un indice. Si les nœuds terminaux '0' et '1' sont associés respectivement aux indices '0' et '1' alors les indices décroissent de manière strictement monotone sur tous les chemins qui commencent à la racine de l'arbre et qui se terminent sur un nœud terminal. Il est possible de démontrer [Bry86] que cette contrainte rend la représentation canonique.

Ainsi, la nature des *BDD* facilite l'élaboration des algorithmes généraux pour la manipulation des fonctions booléennes. En plus la canonicité des *ROBDD* simplifie la vérification de la satisfiabilité d'une fonction, car une fonction non satisfiable est représentée par l'unique

arbre dont la racine est le nœud terminal ‘0’. C’est cette opération qu’on effectue pour vérifier la conductivité d’une branche en cours de construction (§4.5.3.2).



a) Ordonnancement optimal b) Mauvais ordonnancement

Figure 7-2 : *Exemple de la représention ROBDD*

7.3.3 L’ordonnancement des variables

La canonicité est une propriété intéressante des *ROBDD* lorsque l’on est amené à manipuler des expressions booléennes. Toutefois la canonicité n’est obtenue que si à chaque variable est associé un indice unique. Autrement dit, pour qu’une représentation en arbre de décision binaire soit canonique, il faut établir une relation d’ordre stricte entre toutes les variables. L’inconvénient majeur des *ROBDD* est le fait que leur complexité est très sensible à cette relation d’ordre.

L’identification de l’ordonnancement optimal reste un problème difficile et pour lequel il n’existe pas de solution algorithmique [Fri87]. Cependant, un grand nombre d’heuristiques ont été étudiées pour trouver des ordonnancements satisfaisants dans un grand nombre de cas [But91]. La plupart des approches sont basées sur des parcours en profondeur ou en largeur de la logique d’un circuit. La mise en œuvre de ces heuristiques peut, dans certains cas aboutir à un meilleur ordonnancement. Toutefois, ces résultats sont difficilement généralisables, et des heuristiques satisfaisants pour certains circuits peuvent s’avérer catastrophiques pour d’autres.

Face à cette incertitude, nous avons préféré utiliser tout simplement l’ordonnancement obtenu par parcours en profondeur. Pour chaque cône, l’ordonnancement ne concerne que les variables primaires. Il est obtenu par la fonction d’identification des variables primaires, qui parcourt en profondeur le graphe de dépendances locales.

7.4 La représentation des cônes – la figure CNS

La figure CNS (Cone Netlist Structure) est la structure de données principale générée par YAGLE. C'est une structure conçue suivant la même stratégie que la BEFIG, la LOFIG et la PHFIG, et fait appel à une bibliothèque de fonctions pour la création et la manipulation de cette structure, ainsi qu'à un parser / driver pour permettre une représentation textuelle.

7.4.1 La hiérarchie de la structure

La figure CNS est une structure hiérarchique hétérogène. Chaque niveau possède des structures de données particulières et correspond à une description de plus en plus affinée. La Figure 7-3 montre les objets de cette hiérarchie.

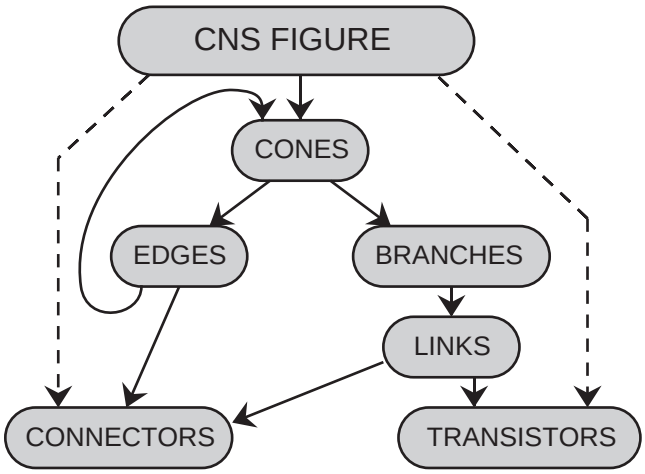


Figure 7-3 : *La hiérarchie de la figure CNS*

La structure (ou l'objet) de plus haut niveau est la *CNSFIG* proprement dite. Cette figure correspond à la vue désassemblée d'un circuit entier. Comme la *LOFIG*, l'interface du circuit est représentée par une liste de structures qui représentent les connecteurs logiques. Elle peut contenir aussi le réseau de transistors à partir duquel la *CNSFIG* est générée. Les transistors et les connecteurs sont alors référencés par l'objet du plus haut niveau mais aussi par des objets de plus bas niveau.

La *CNSFIG* contient aussi la liste des cônes, les objets de base du désassemblage. Chaque cône contient la liste de ses entrées et de ses sorties (*EDGES*) et une liste distincte pour les différents types de branches. Chaque *EDGE* est une « union » qui référence soit un cône soit un connecteur externe. Ainsi, il est simple d'effectuer un parcours du graphe de dépendance dans les deux sens.

Chaque branche constitue un chemin entre la sortie du cône et une source de tension. La liste à laquelle elle appartient dépend du type de la source (V_{dd}, V_{ss}, ou connecteur externe). Une branche est composée d'une liste de maillons (*LINKS*) qui représentent les éléments qui composent le chemin. Un *LINK* est aussi une « union ». Dans la plupart des cas, l'élément référencé est un transistor, mais, dans le cas d'une branche qui termine sur un connecteur externe, le dernier maillon est un connecteur.

Champ	**Description**
NAME	Nom de la figure
LOCON	La liste des connecteurs externes
LOTRS	La liste des transistors
CONE	La liste des cônes
LOFIG	Interconnexion de portes
BEFIG	Réseau booléen issu de l'abstraction

Figure 7-4 : *La structure CNSFIG*

La Figure 7-4 montre les champs de la structure *CNSFIG*. L'utilité des quatre premiers champs est claire. Les deux derniers champs sont des enrichissements optionnels de la structure.

Les champs *BEFIG* et *LOFIG* sont remplis par YAGLE à la demande de l'utilisateur. La *BEFIG* correspond au résultat final de l'abstraction fonctionnelle, il s'agit du modèle comportemental généré pour le réseau de cônes. La *LOFIG* correspond au réseau de portes issu du désassemblage. Voir §7.5.1 pour plus de détails.

7.4.2 La structure d'un cône

Dans la structure CNS, un cône est un élément d'une liste chaînée. Cette organisation facilite l'allocation des structures et le parcours de l'ensemble des cônes. Pour établir les champs de la structure, il y avait deux motivations principales :

1) Simplifier le parcours du graphe de dépendance globale du circuit dans les deux sens, c'est-à-dire des entrées aux sorties ou des sorties aux entrées.

2) Simplifier l'accès aux informations topologiques et électriques nécessaires à la caractérisation d'un cône : fonctionnelle, temporelle, consommation ou autre.

La structure est indiquée dans la Figure 7-5.

- Les champs INCONE est OUTCONE sont des listes de type *EDGE* et correspondent respectivement à la liste des entrées du cône et à la liste des cônes dont une entrée est la sortie du cône.

- Les trois types de branches sont représentés par les trois champs BRVDD, BRVSS et BREXT qui sont des listes de type *BRANCH*.

- Chaque branche contient une liste de type *LINK*, dont les éléments sont des transistors et éventuellement un connecteur faisant partie de la branche. Le champ CAPA indique la capacité attachée à l'équipotentielle la plus proche de la sortie du cône.

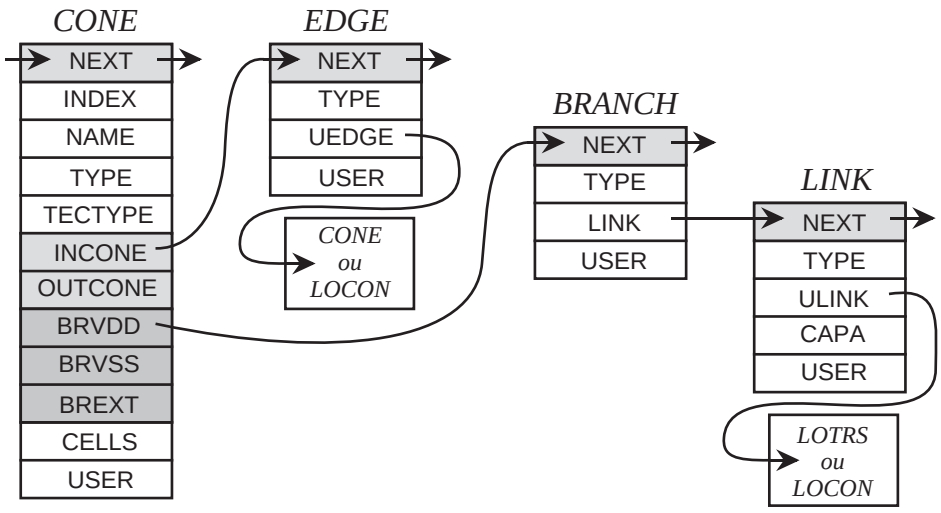


Figure 7-5 : *La structure d'un cône*

- Le champ TYPE est un indicateur qui résume le résultat des analyses génériques, telles que l'analyse de la conflictualité ou l'identification des éléments séquentiels.

- Le champ TECTYPE est un indicateur qui précise certaines caractéristiques reconnues qui sont dépendant de la technologie du circuit.

- Le champ USER regroupe des informations topologiques supplémentaires qui ont été détectées par YAGLE. Les plus importants sont :

  a) La liste des ensembles de branches parallèles (les entrées en commun).
  b) La liste des branches qui correspondent à des « bleeders ».
  c) La liste des pairs de transistors connectés en tant que switch CMOS.

7.5 L'architecture de YAGLE

Dans les sections précédentes, nous avons décrit toutes les bibliothèques et les structures de données nécessaires pour la réalisation du logiciel YAGLE. Un effort important a été consenti pour cette réalisation car, comme on voit à la Figure 7-6, YAGLE est exploité au laboratoire LIP6 par de nombreux outils, dans le cadre de divers projets de recherche.

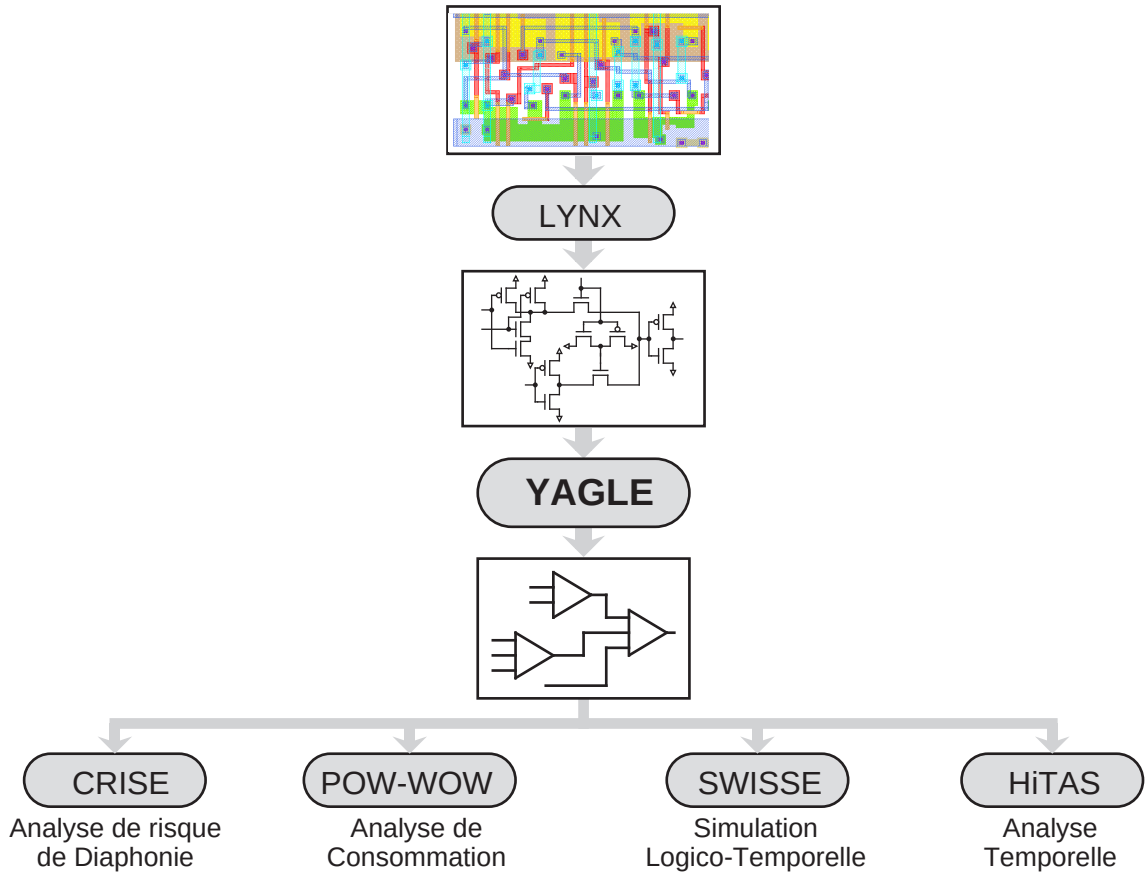


Figure 7-6 : *L'utilisation de YAGLE au LIP6*

Le module de désassemblage en cônes et l'analyse algébrique des éléments séquentiels fait 25 000 lignes de 'C' à lui seul. Les bibliothèques de reconnaissance FCL et GENIUS (voir la chapitre 6), ainsi que la bibliothèque CNS, regroupe 30 000 lignes, sans compter les bibliothèques et parsers / drivers de la chaîne ALLIANCE.

7.5.1 Les fichiers d'entrées et de sorties

YAGLE prend en entrée un réseau de transistor à l'aide du parser de la *LOFIG*, ce qui lui permet de lire la plupart des formats standards. Si le concepteur souhait exploiter le reconnaissance de formes, il doit fournir les fichier qui décrivent les structures à reconnaître et les actions associées.

Les fichiers en sorties sont les suivants :

1) ***Le fichier de rapport*** – Ce fichier est toujours créé. Il contient des informations sur le réseau de transistors et sur le réseau de cônes. Pour les transistors, il signal principalement les transistors non utilisés, les connecteurs non connectés, et les transistors montés en résistance, capacité ou diode. Pour les cônes, il indique les résultats de l'analyse globale, de l'analyse des éléments séquentiels, ainsi que la connectivité du réseau.

2) ***Le modèle comportemental*** – Ce fichier est créé sur la demande de l'utilisateur par le driver de la *BEFIG* à partir du modèle comportemental généré par YAGLE.

3) ***Le réseau de portes*** – Ce fichier est créé sur la demande de l'utilisateur par le driver de la *LOFIG*. Il s'agit d'un réseau de portes classique, où chaque porte correspond à une instance d'un modèle. YAGLE attribue le même modèle aux cônes ayant des comportements identiques et des structures similaires. Un fichier comportemental est créé pour chacun des modèles.

4) ***La hiérarchie de l'approche hybride*** – Il s'agit de la description structurelle qui regroupe les instances reconnues avec la partie du circuit traité par le partitionnement en cônes.

5) ***Le fichier CNS*** –Créé sur la demande de l'utilisateur par le driver de la *CNSFIG.*

7.5.2 Les principaux modules du logiciel

La Figure 7-7 montre les principales étapes de traitement dans YAGLE. L'étape de chargement du réseau de transistors est effectuée par les parsers de la *LOFIG*. Sur ce réseau de transistors, YAGLE effectue quelques pré-traitements. Le plus important est la détection et le marquage des signaux d'alimentations. Cette étape est indispensable car la fabrication des branches et la caractérisation fonctionnelle dépendent de cette information.

Après ce marquage, le désassemblage proprement dit peut commencer :

1) ***Reconnaissance de formes*** – Si le concepteur fourni une bibliothèque de structures à reconnaître, alors YAGLE fait appel à GENIUS pour lire la bibliothèque, identifier les structures, et exécuter les *ACTIONs*. GENIUS fait appel à FCL pour la reconnaissance des structures élémentaires. Si des structures sont reconnues, alors les transistors en faisant partie de cette structure sont retirés du circuit et le traitement se poursuit avec le réseau de transistors troué.

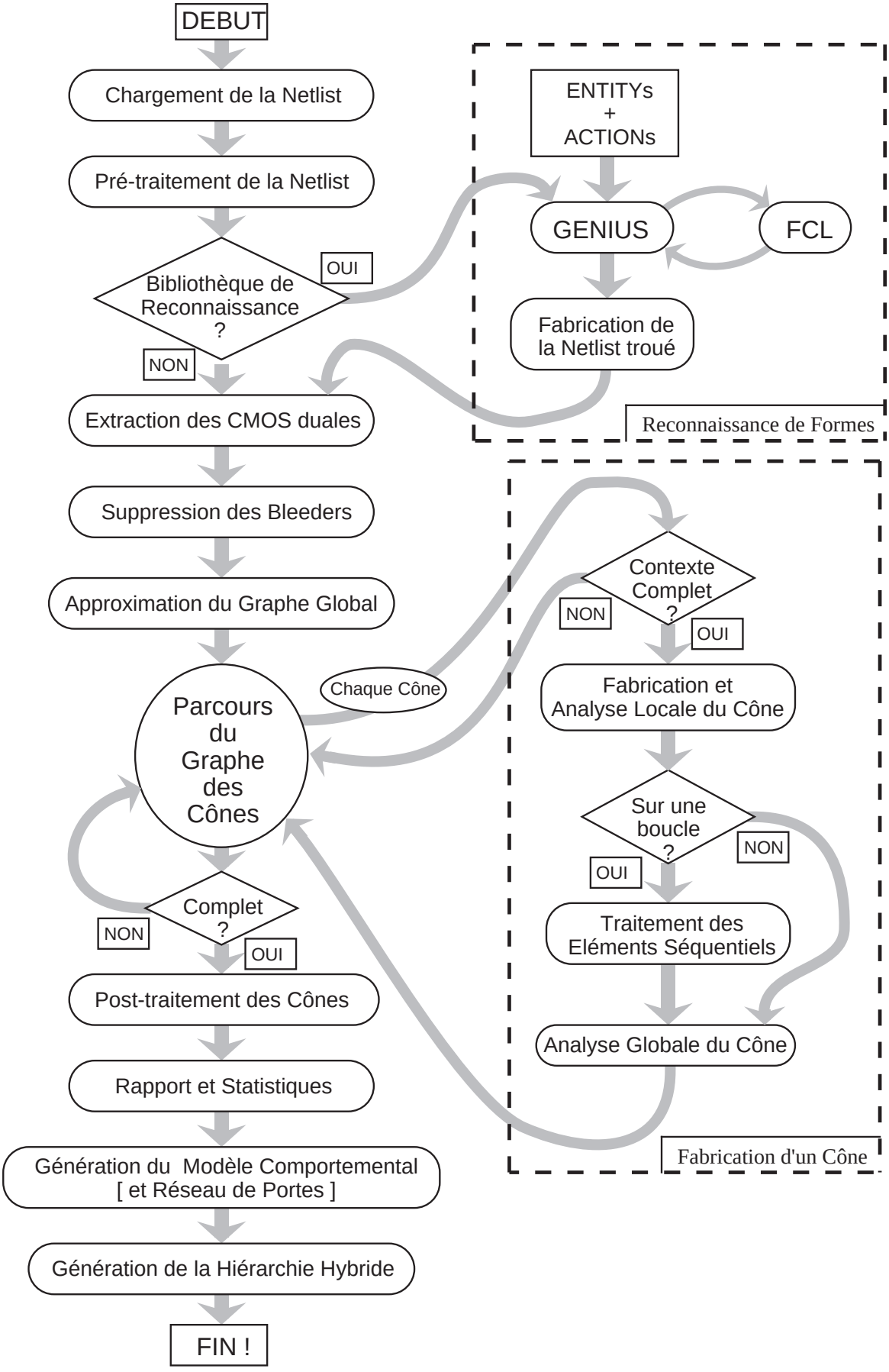


Figure 7-7 : *Les étapes de YAGLE*

2) ***Extraction des portes CMOS duales*** – Tous les nœuds sur lesquels un cône doit être construit sont identifiés. Des critères sévères (voir §4.5.2) sont appliqué pour construire les cônes qui correspond à des portes CMOS duales.

3) ***Détection des Bleeders*** – Marquage des transistors montés en bleeders (§4.6.1).

4) ***Approximation du graphe global*** – Construction du graphe de dépendances globales du circuit en identifiant, pour chaque cône, la liste des entrées directes éventuelles.

5) ***Phase de fabrication des cônes*** – Il s'agit de la phase principale de YAGLE. C'est un parcours récursif en profondeur d'abord du réseau de cônes (le graphe de dépendances globales). Pour chaque cône visité, on construit son graphe de dépendances locales. Si ce graphe est suffisamment complet, alors on procède à la fabrication et à l'analyse fonctionnelle du cône. Il est possible que plusieurs parcours récursifs soient nécessaires à cause des boucles dans le graphe global.

6) ***Traitement des éléments séquentiels*** – Cette analyse se fait en même temps que la phase de fabrication car il est utile de connaître, dès que possible, si un cône correspond à un élément mémorisant. L'analyse est déclenchée, après la fabrication d'un cône, si celui-ci se trouve sur une boucle.

7) ***Analyse globale du cône*** – On vérifie la fonctionnalité du cône en fonctions de ses variables primaires (voir §4.5.4).

8) ***Post-traitement des cônes*** – Un ensemble de traitements est appliqué uniquement après la fabrication complète de tous les cônes. Ces traitements incluent la fabrication de la liste des sorties (OUTCONE) pour chaque cône, l'orientation des connecteurs externes, et la suppression des cônes vides.

Les dernières étapes concernent la fabrication des fichiers de sorties (voir §7.5.1). Un fichier de rapport est systématiquement généré. Il fournit des renseignements importants pour la vérification du circuit. Les autres fichiers (modèle comportemental, réseau de portes, hiérarchie hybride, ou même le fichier CNS) sont générés selon les options d'utilisation de YAGLE.

7.5.3 Les contraintes des concepteurs

La fabrication correcte et efficace des cônes par YAGLE dépend de l'existence de suffisamment d'informations sur les corrélations. Toutefois, dans un contexte d'utilisation industrielle, on peut identifier deux situations pour lesquelles cette information peut être incomplète.

1) Si le réseau de transistors fournit à YAGLE est un bloc plutôt que le circuit entier, alors il se peut qu'il y a des contraintes liées au contexte d'utilisation de ce bloc. Ces contraintes se traduisent par des corrélations entre les signaux sur l'interface du bloc et qui n'apparaissent pas à l'intérieur.

2) Il se peut que certaines corrélations se trouvent en amont des éléments mémorisants. Il s'agit de corrélations à travers les cycles. Notre méthode d'analyse ne permet pas de prendre en compte ce genre de corrélations. En effet, nous considérons les éléments séquentiels comme des points d'arrêt dans les graphes de dépendances locales.

Pour gérer ces situations, il est nécessaire de donner aux concepteurs la possibilité de fournir l'information qui manque. Pour se faire, nous avons utilisé et enrichi le concept du fichier d'informations de DESB. Ce fichier permet de décrire certaines contraintes sur un ensemble de signaux :

a) Seulement un des signaux est à '0' (ou à '1').
b) Pas plus d'un des signaux est à '0' (ou à '1').

Les signaux sur lesquels ces contraintes sont spécifiées peuvent appartenir à l'interface du circuit ou être un signal interne. L'exploitation de ces contraintes se fait pendant la phase de fabrication des cônes et de l'analyse globale. Chaque contrainte est traduite en une expression booléenne. Si un graphe de dépendances locales contient un signal sur lequel une contrainte est imposée, alors ce signal fait partie des variables primaires. Toutes les expressions qui utilisent cette variable sont alors contraintes par l'expression booléenne qui représente la contrainte.

7.5.4 Intégration avec FCL et GENIUS

La gestion de l'approche hybride se manifeste à deux moments distincts dans YAGLE. Au début du traitement les modules de reconnaissance, FCL et GENIUS, sont appelés et à la fin du traitement une description hiérarchique est générée en regroupant les modèles des

instances reconnues avec le modèle comportemental généré pour le réseau de cônes (voir §6.5).

La difficulté principale est la génération du réseau de transistors troué lors de l'appel des modules de reconnaissance de formes. Cette tâche a été facilitée par l'adoption d'une méthode de communication simple entre FCL, GENIUS et YAGLE. Pour des raisons de simplicité et cohérence, nous avons souhaité utiliser le même type d'objet pour dans les interfaces entre FCL et GENIUS et entre GENIUS et YAGLE. Cet objet permet de définir la structure reconnue et doit contenir les informations suivantes :

1) Les signaux du circuit qui correspondent à l'interface de la structure reconnue.
2) Un type qui indique le modèle de la structure.
3) La liste de transistors faisant partie de la structure.

Cet objet est donc identique à la structure utilisée dans la *LOFIG* pour représenter une instance enrichie avec la liste des transistors. La communication entre les trois modules se fait alors comme indiquée dans la Figure 7-8.

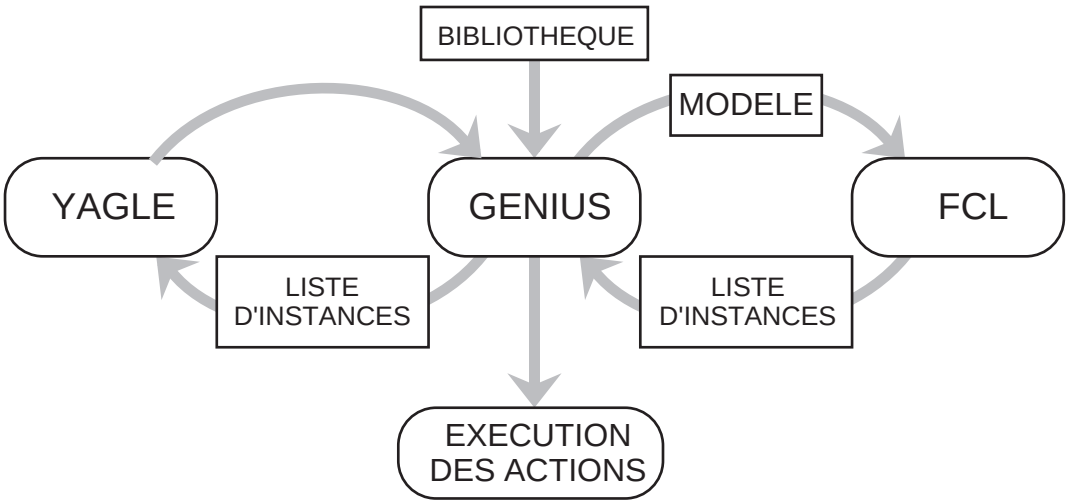


Figure 7-8 : *La communication entre YAGLE, GENIUS, et FCL*

7.6 Conclusion

Dans ce chapitre, nous avons montré la mise en œuvre d'un logiciel, YAGLE, qui réalise le désassemblage et l'abstraction fonctionnelle en utilisant les méthodes expliquées en détail dans les chapitres précédents.

Ce logiciel s'appuie sur la plate-forme ALLIANCE pour profiter de la cohérence des structures de données, des parsers/drivers et des bibliothèques de fonctions de manipulation

des expressions booléennes. Nous avons adopté la philosophie d'ALLIANCE dans la mesure ou nous avons rendu « ouvert » la structure de données principale générée par YAGLE, la figure CNS. Cette stratégie a facilité la tâche de nombreux projets de recherche qui s'appuie sur le résultat fourni par YAGLE.

Ce logiciel à fait ses preuves dans un vrai contexte industriel chez BULL. En effet, afin d'effectuer l'analyse temporelle d'un processeur dédié, BULL à fait appel à HiTAS, l'outil d'analyse temporelle statique développé au laboratoire LIP6, qui s'appuie sur le réseau de cônes en sortie de YAGLE. Ce processeur, conçu pour leurs serveurs, à une complexité de plusieurs millions de transistors.

Chapitre 8

VALIDATION ET RESULTATS

Dans ce chapitre, nous exposons quelques résultats obtenus par le logiciel YAGLE. L'objectif est de mettre à l'épreuve, sur des circuits de taille réelle, les techniques que nous avons proposé aux chapitres 4, 5 et 6. Nous commençons donc par présenter les méthodes utilisées pour cette évaluation ainsi que les spécifications de notre plate-forme d'expérimentation.

8.1 Introduction

Dans les chapitres précédents, nous avons présenté nos apports dans les domaines du désassemblage et de l'abstraction fonctionnelle. Ces apports se situent autour de trois axes principaux : la prise en compte des corrélations, l'analyse algébrique des éléments séquentiels, et l'approche hybride. Nous avons donc organisé ce chapitre autour de trois sections. Chaque section est consacrée à la mise à l'épreuve d'un des aspects développés dans cette thèse.

La première section concerne la validation du traitement des corrélations. Ce traitement apporte une première amélioration par rapport aux techniques mises en œuvre dans DESB[Lau94]. Notre point de référence est donc DESB, le précurseur de YAGLE qui a aussi été développé au laboratoire LIP6. Dans cette section, nous allons essayer de montrer que la prise en compte systématique des corrélations ne se traduit pas en une augmentation du temps de calcul. Au contraire elle permet de traiter une plus grande diversité de circuits.

La deuxième section s'intéresse à l'analyse algébrique des éléments séquentiels. Plusieurs aspects de cette analyse peuvent êtres examinés. Le premier point concerne l'identification. Nous allons examiner si notre méthode permet de traiter tous les types de latchs que nous avons à notre disposition. De plus, nous allons étudier l'impact de cette analyse algébrique sur le temps de calcul. Deuxièmement, nous allons montrer que les modèles comportementaux obtenus satisfont les besoins de fidélité et de lisibilité.

Dans la dernière section, nous illustrons l'utilité de l'approche hybride en nous servant de deux exemples de circuits réels auxquels nous nous sommes confrontés.

8.2 La plate-forme d'expérimentation

Pour des raisons de cohérence, nous avons adopté une plate-forme matérielle unique pour effectuer nos expériences. Il s'agit d'une station de travail UltraSparc 5 avec 256 Mo de mémoire vive, et 500Mo de swap, doté d'un processeur UltraSparc IIi, cadencé à 333MHz. Cette machine a une performance de 14,1 SPECint95.

Les circuits utilisés pour les essais sont des réseaux de transistors à plat de plusieurs origines :

1) Laboratoire LIP6 : Format ALLIANCE
2) BULL : Format VTI
3) ST Microelectronic : Format SPICE

8.3 Comparaison des performances avec DESB

8.3.1 Introduction

Dans le but d'obtenir des résultats cohérents, nous avons recompilé DESB dans une version optimisée pour notre plate-forme d'expérimentation.

Dans cette section, nous comparons deux méthodes de construction des cônes : celle de DESB et celle éposée au chapitre 4. La méthode de DESB est beaucoup plus simple. Toutes les branches sont d'abord construites. Puis à la fin du traitement les fausses branches sont éliminées sur les cônes conflictuels ou hautes impédances. YAGLE, par contre, exploite les corrélations systématiquement pendant la fabrication des branches.

Nous utilisons deux ensembles de circuits d'origine différente : un ensemble de circuits conçus au laboratoire LIP6 à l'aide des outils et des cellules de la chaîne ALLIANCE, et quelques blocs provenant d'un processeur conçu chez BULL.

8.3.2 Les circuits conçus au LIP6

Dans un premier temps, nous avons effectué des expériences avec les blocs issus des générateurs de la chaîne ALLIANCE. Nous illustrons les résultats obtenus pour le générateur de ROM. En effet, grâce à ce générateur, il est possible de construire rapidement un grand nombre de circuits avec des tailles différentes. De plus, l'utilisation de la logique DOMINO préchargée est intéressante du point de vue de l'analyse fonctionnelle.

La Figure 8-1 compare les temps de désassemblage de YAGLE et du DESB pour plusieurs tailles de ROM. La complexité en nombre de transistors paraît élevée, toutefois il faut remarquer que la moitié des transistors correspond au contenu de la ROM. Les deux outils sont performants avec un léger avantage à YAGLE. Le temps de traitement reste linéaire avec la complexité pour les deux outils.

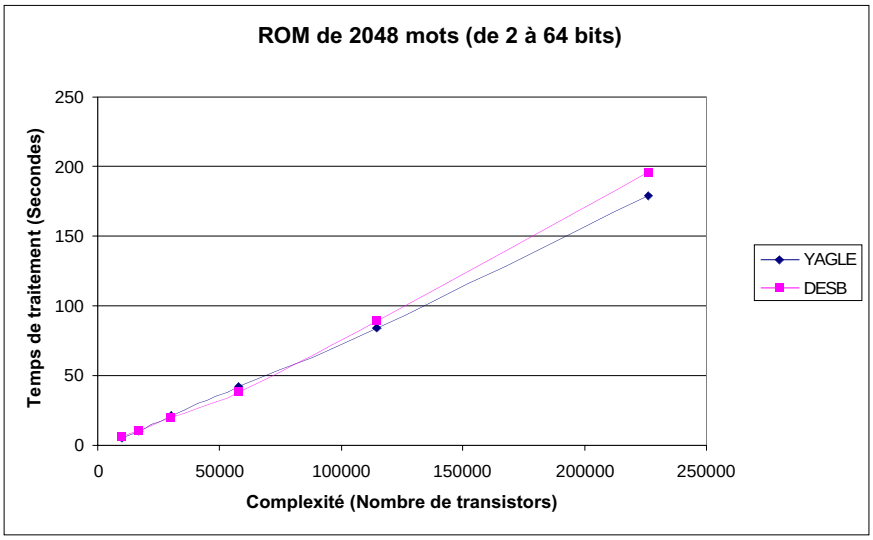


Figure 8-1 : *Comparaison du temps de traitement des ROMs*

La Figure 8-2 montre la consommation mémoire pour les traitements des mêmes ROMs. Ici encore le comportement des deux outils est linéaire, toutefois on remarque que YAGLE est deux fois plus efficace que DESB.

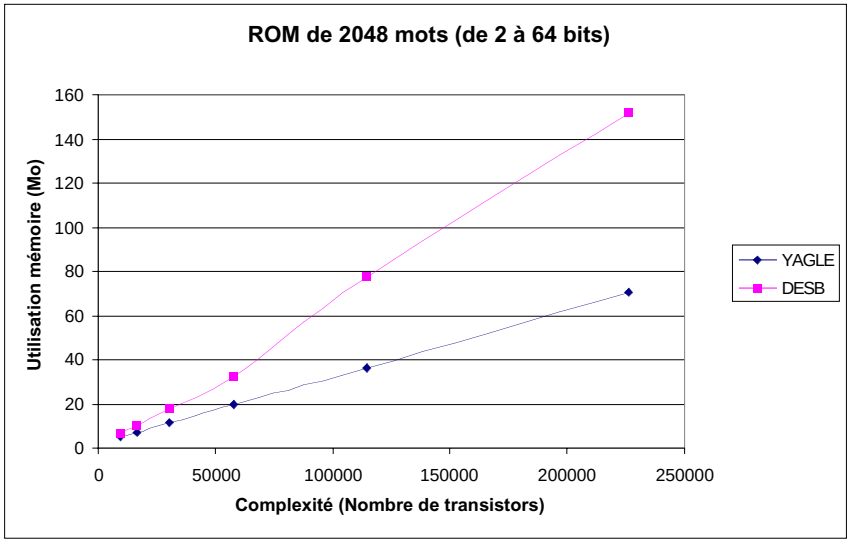


Figure 8-2 : *Comparaison de l'utilisation mémoire*

Ces exemples simples montrent des performances similaires pour des circuits qui peuvent être traités par les deux méthodes. Les exemples qui suivent tentent de démontrer les différences dans la généralité des deux méthodes. Nous nous appuyons sur un ensemble de circuits complets développés au laboratoire LIP6.

Les circuits sont les suivants :

AMD2901 (5 336 transistors) – Un tutoriel ALLIANCE pour illustrer la conception en « chemin de données ».

DLXm (37 794 transistors) – Une réalisation du processeur DLX Microprogrammé par les outils de la chaîne ALLIANCE. Ce circuit comporte un « chemin de données », une partie « standard-cell » et un automate d'états.

MIPSR3000 (46 327 transistors) Une réalisation Microprogrammée du processeur Mips R3000.

DMAW (107 866 transistors) – Un des blocs d'un circuit en cours de développement au LIP6. Ce circuit réalise l'interface entre un bus PCI et des liaisons série à 1 Gbit.

PCIDDC (202 264 transistors) – L'interface PCI pour le router RCUBE, développé au LIP6.

Pour chacun de ces circuits, nous comparons principalement le temps de désassemblage et l'utilisation mémoire par chacun des deux outils. Nous illustrons également la qualité du désassemblage par deux aspects. Premièrement nous comptabilisons le nombre de boucles entre deux cônes qui n'ont pas été reconnues en tant que bleeder ou latch. Deuxièmement le nombre de cônes conflictuels et à haute impédance obtenus après l'élimination des fausses branches. Un nombre important de boucles ou de cônes conflictuels ou à haute impédance indique une analyse fonctionnelle sous-optimale et rend le modèle comportemental moins efficace.

AMD2901

	Temps du désassemblage	***Utilisation mémoire***	***Latchs reconnus***	***Boucles inconnues***	***Conflits non-résolus***	***Hautes Impédances***
YAGLE	1,2s	3,58Mo	136	0	0	8
DESB	1,2s	3,12Mo	136	0	0	8

DLXm

	Temps du désassemblage	*Utilisation mémoire*	*Latchs reconnus*	*Boucles inconnues*	*Conflits non-résolus*	*Hautes Impédances*
YAGLE	81,3s	20,81Mo	1508	0	129	192

MIPSR3000

	Temps du désassemblage	*Utilisation mémoire*	*Latchs reconnus*	*Boucles inconnues*	*Conflits non-résolus*	*Hautes Impédances*
YAGLE	133s	25,1Mo	1814	0	128	192

- DESB échoue sur les deux circuits précédents. Les deux échecs sont en effet dus à la même raison : l'existence d'un décaleur 32 bits. La construction des branches, en ignorant les corrélations, entraîne la création d'un nombre très important de fausses branches. L'analyse fonctionnelle a posteriori est trop coûteuse et provoque une explosion combinatoire.

DMAW

	Temps du désassemblage	*Utilisation mémoire*	*Latchs reconnus*	*Boucles inconnues*	*Conflits non-résolus*	*Hautes Impédances*
YAGLE	629,5s	300Mo	7 382	0	2	181
DESB	934s	219Mo	6 707	675	32	981

PCIDDC

	Temps du désassemblage	*Utilisation mémoire*	*Latchs reconnus*	*Boucles inconnues*	*Conflits non-résolus*	*Hautes Impédances*
YAGLE	466,8s	300Mo	11 109	88	132	246
DESB	548s	238Mo	10 171	1026	132	360

Ces deux derniers circuits sont intéressants à cause de leur complexité et du grand nombre d'éléments séquentiels qu'ils comportent. Les deux outils réussissent à construire correctement les cônes car, malgré un grand nombre de fausses branches, les corrélations

restent relativement locales. L'utilisation mémoire de YAGLE est plus élevée à cause de sa structure de données qui contient plus d'informations.

La recherche des éléments séquentiels dans le réseau de cônes avant l'élimination des fausses branches est le facteur qui ralentit le traitement dans DESB. Cette difficulté, liée au nombre insuffisant de structures reconnues par DESB, entraîne l'échec de la reconnaissance de tous les éléments séquentiels.

8.3.3 Les circuits de BULL

Nous allons maintenant comparer les performances de YAGLE et de DESB sur un ensemble important de circuits provenant d'une source extérieure au laboratoire LIP6. Il s'agit de blocs fonctionnels d'un processeur de plusieurs millions de transistors conçu chez BULL.

Les blocs sont d'une conception « full-custom ». De ce point de vue, ils sont très différents des circuits développés au laboratoire LIP6. Ils contiennent très souvent des transistors de passage qui peuvent provoquer la construction de nombreuses fausses branches.

Ainsi, la prise en compte des corrélations est primordiale pour un traitement correct et efficace de ces circuits.

Statistiques Globales

- Le processeur comprend 93 blocs fonctionnels différents, dont une matrice de ROM, et trois blocs analogiques.
- La complexité totale des 90 blocs numériques est de ***678 339*** transistors
- Le bloc le plus complexe (à part la ROM) contient ***27 255*** transistors

Résultats Globaux

Le tableau suivant montre une vue d'ensemble du désassemblage de ces blocs par YAGLE et par DESB à l'exception des blocs analogiques :

	Nombre d'échecs	***Temps total***	***Mémoire max.***	***Mémoire moyenne***
YAGLE	1	2 316,6s	146,6Mo	11,5Mo
DESB	10	5 057,3s	161,2Mo	14,5Mo

- Un échec correspond à un traitement nécessitant plus d'une heure de temps CPU.
- Le temps total indique la somme des temps de traitement pour tous les blocs où le traitement n'a pas échoué.
- De même, pour la consommation mémoire (cons. Max et Moyenne), seuls les blocs pour lesquels le traitement à réussi ont été comptabilisés.

Pour certains de ces blocs, le nombre de conflits non-résolus peut être important. En effet, dans ces blocs, souvent la source de corrélation se trouve à l'extérieur du bloc. Ainsi, l'ignorance de ces corrélations (absence de contraintes externes) provoque la construction de fausses branches et le résultat obtenu est sous-optimal. En fait, le seul cas d'échec de YAGLE est effectivement lié à l'absence de cette information de corrélation sur les entrées du bloc.

Résultats Détaillés

Il serait trop lourd de montrer les détails de l'intégralité des essais effectués sur les 90 blocs. Toutefois, les performances obtenues sur les quinze blocs les plus complexes sont représentatives de la performance globale. Ceci est d'autant plus vrai que ces quinze blocs représentent près de 40% de la complexité totale.

La Figure 8-3 compare les performances de YAGLE et de DESB du point de vue du temps de traitement et de la consommation mémoire. Les graphiques de la Figure 8-4 et la Figure 8-5 montrent les temps de calcul obtenus et la consommation mémoire pour les 90 blocs de l'ensemble.

	Nombre de transistors	*Temps YAGLE*	*Mémoire YAGLE*	*Temps DESB*	*Mémoire DESB*
fpm4_p	27 255	375,1s	34,7 Mo	ECHEC	ECHEC
mfroml_p	24 620	4s	12,7 Mo	4,9	13,3
fromr_p	24 610	4,1s	12,5 Mo	4,8	13,2
ted_j	24 527	198,8s	71,9 Mo	ECHEC	ECHEC
ddsbm_k	16 374	16,5s	20,3 Mo	14,9	20,8
dnc2_n	16 025	39,7s	16,0 Mo	10,6	16,3
po_d	15 703	175,1s	146,6 Mo	495,1	161,2
dirlru_i	15 310	53,6s	36,4 Mo	ECHEC	ECHEC
cdad_k	13 220	127,8s	25,1 Mo	33,7	34,5
dddp_k	12 587	134,7s	21,2 Mo	139,4	50,5
blk5_n	12 330	31,1s	28,9 Mo	ECHEC	ECHEC
ddkl_k	12 098	1s	6,6 Mo	2,2	6,7
ddku_k	12 098	1,3s	6,6 Mo	2,7	6,9
dnc1_n	12 098	13,8s	11,7 Mo	7,5	11,4
fr1a_p	12 098	1,1s	6,6 Mo	2,1	6,8
TOTALE	**250 953**	**1177,7s**		**717,9**	

Figure 8-3 : *Le traitement des 15 blocs les plus complexes*

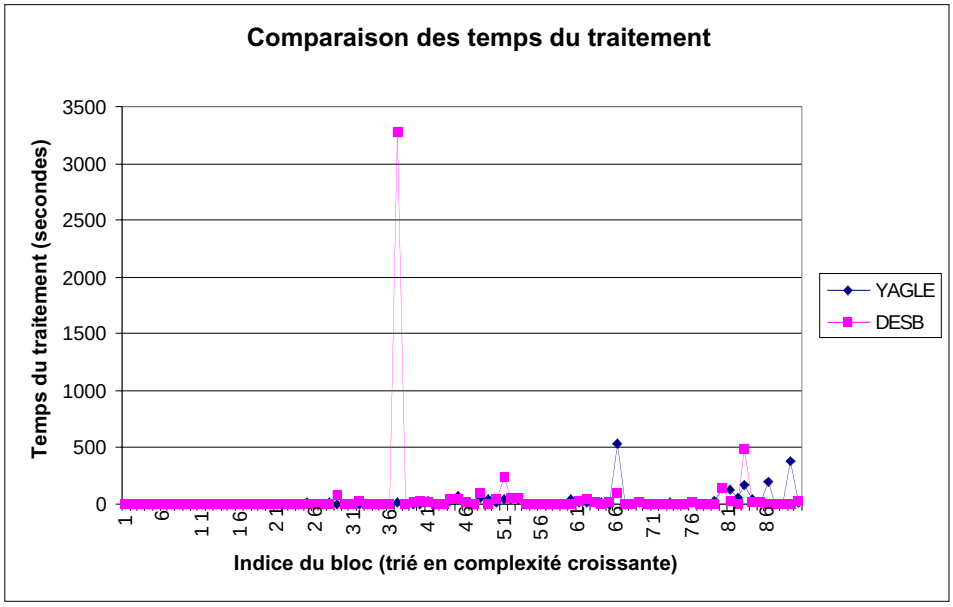


Figure 8-4 : *Comparaison des temps de calculs sur l'ensemble des blocs*

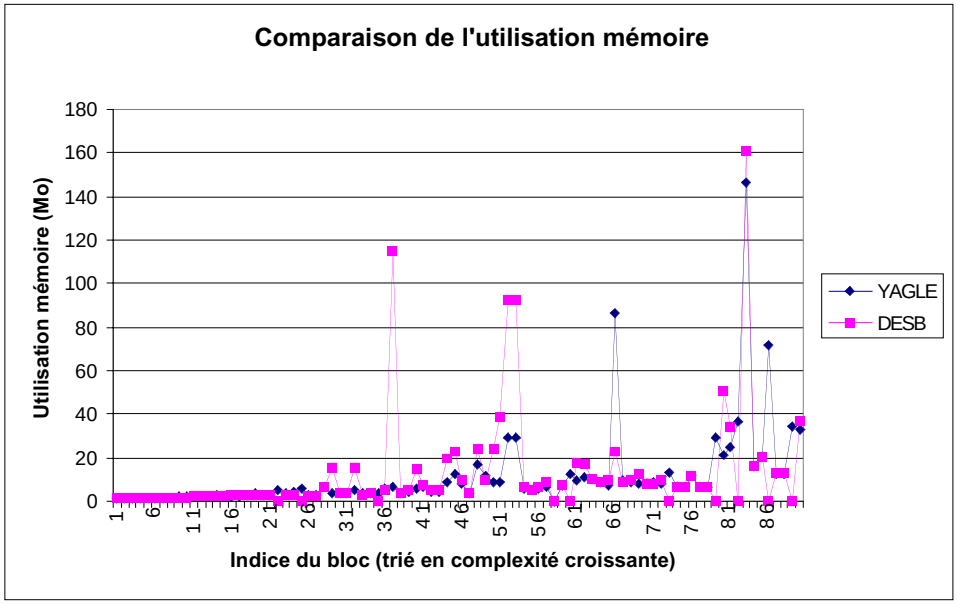


Figure 8-5 : *Comparaison de l'utilisation mémoire*

Pour les circuits de BULL, le résultat le plus frappant est le taux d'échec de DESB. Ces échecs sont essentiellement dus à l'incapacité de DESB de construire toutes les branches d'un cône avant de procéder à l'élimination des fausses branches. En effet, la présence des transistors de passage provoque la construction de nombreuses fausses branches. Toutefois, même si on ne considère que les réussites, les performances de YAGLE sont bien plus proche que DESB d'une variation linéaire du temps de calcul et de l'utilisation mémoire en fonction de la complexité.

8.3.4 Conclusion

Les résultats montrent que dans un bon nombre de cas DESB est capable de traiter le circuit rapidement et avec une consommation raisonnable de mémoire. D'ailleurs, il y a des cas ou ses performances sont plus élevées que YAGLE.

Toutefois, son approche de construction des branches en ignorant les corrélations montre sa faiblesse dans les étapes qui suivent la construction. En effet, dans les circuits où le nombre de fausses branches peut être élevé, la performance de DESB chute dans les phases de détection des latchs et de la suppression des fausses branches. L'efficacité de DESB est alors fortement influencée par la structure du circuit. Cette inefficacité peut même aller jusqu'à l'incapacité de traiter le circuit dans un temps raisonnable.

Par contre, l'approche de YAGLE, plus prudente, basée sur l'exploitation systématique des corrélations pendant la construction des cônes, se montre bien plus générale. La possibilité d'échec sur les structures complexes est fortement réduite. Cela se traduit aussi en une amélioration de la linéarité indépendamment de la structure. Même sur des structures ayant peu de fausses branches, YAGLE maintient, en général, une supériorité nette, malgré son algorithme plus complexe.

8.4 Analyse algébrique des éléments séquentiels

Dans la section précédente, nous avons montré comment la méthode de construction des cônes mise en œuvre par YAGLE peut améliorer les performances du désassemblage. Dans cette section, nous montrons comment la méthode algébrique utilisée pour l'identification et la modélisation des éléments séquentiels, peut acroître la généralité du désassemblage et de l'abstraction fonctionnelle.

8.4.1 La méthode de validation

Pour valider la méthode détaillée dans le chapitre 5, il faut considérer deux aspects. Premièrement, il faut comparer le taux de réussite et la performance de l'identification des éléments séquentiels. Deuxièmement, il faut valider la qualité du modèle généré automatiquement, c'est-à-dire sa correspondance avec la réalité et sa lisibilité.

Pour la performance, nous nous sommes appuyés sur un ensemble de circuits. Cet ensemble a été choisi de manière à présenter la plus grande diversité d'éléments séquentiels. Il permet ainsi de montrer le coût supplémentaire de l'analyse algébrique pour des structures diverses. De plus, ce sont des circuits pour lesquels le nombre d'éléments séquentiels, ainsi que leur structure, sont connus.

La qualité de la modélisation est illustrée dans la dernière partie de cette section à l'aide de deux exemples d'éléments séquentiels.

8.4.2 Les performances de la méthode algébrique

- Parmi les circuits du laboratoire LIP6, nous avons choisi les trois circuits suivant :
  1) ***AMD2901 :*** 5 336 transistors avec 136 latchs.
  2) ***MIPSR3000 :*** 46 327 transistors avec 1 814 latchs.
  3) ***PCIDDC :*** 202 264 transistors avec 11 109 latchs.

Pour chacun des trois circuits, nous avons comparé le temps de traitement total ainsi que la consommation mémoire en utilisant deux méthodes de désassemblage différentes : reconnaissance de formes et analyse algébrique. En effet, l'une et l'autre méthode sont mises en œuvre dans YAGLE. La première méthode reflète l'héritage de DESB. La seconde représente la méthode exposée au chapitre 5.

Ainsi nous effectuons une évaluation du coût de la méthode algébrique pour le temps de calcul et pour la consommation mémoire exprimé en pourcentage. Notons qu'il n'est pas possible de distinguer le temps de fabrication des cônes et le temps d'identification des latchs car, dans la méthode algébrique, l'analyse des boucles se fait pendant la phase de fabrication. La Figure 8-6 montre le résultat de l'analyse pour les trois circuits du laboratoire LIP6.

	Reconnaissance		***Méthode algébrique***		***Les coûts***	
	Temps du traitement	***Utilisation mémoire***	***Temps du traitement***	***Utilisation mémoire***	***Coût en temps***	***Coût en mémoire***
AMD2901	1,2s	3,6 Mo	1,4s	3,7 Mo	17 %	2 %
MIPSR3000	133,0s	25,1 Mo	110,0s	25,8 Mo	- 17 %	3 %
PCIDDC	466,8s	300,2 Mo	760,2s	347,0 Mo	63 %	16 %

Figure 8-6 : *Analyse algébrique des circuits du LIP6*

La Figure 8-6 montre que le coût de l'analyse algébrique reste abordable même dans le cas des circuits complexes contenant un grand nombre de latchs. C'est le cas du PCIDDC qui contient plusieurs FIFOs asynchrones. Le résultat le plus surprenant est la réduction du temps du traitement pour le MIPSR3000. En effet, le coût de l'analyse algébrique est largement compensé par le fait que l'analyse se fait pendant la phase de fabrication des cônes. De ce fait, on évite la construction des fausses branches qui passent par les nœuds mémorisants.

- Nous avons également effectué les mêmes essais sur trois circuits de BULL :

1) ***BLK4_N :*** 8 534 transistors avec 704 latchs.
2) ***DIRLRU_I :*** 15 310 transistors avec 1 547 latchs.
3) ***TED_J :*** 24 527 transistors avec 2 594 latchs.

La Figure 8-7 montre les résultats des mêmes comparaisons effectuées sur ces trois blocs. Comme pour les circuits du laboratoire LIP6, tous les latchs sont identifiés sans recours à la reconnaissance.

	Reconnaissance		*Méthode algébrique*		*Les coûts*	
	Temps du traitement	***Utilisation mémoire***	***Temps du traitement***	***Utilisation mémoire***	***Coût en temps***	***Coût en mémoire***
BLK4_N	9,6s	10,3 Mo	23,3s	12,6 Mo	143 %	23 %
DIRLRU_I	53,6s	36,4 Mo	99,0s	48,1 Mo	85 %	32 %
TED_J	198,8s	71,9 Mo	196,9s	91,7 Mo	- 1 %	27 %

Figure 8-7: *Analyse algébrique des blocs de BULL*

Pour ces circuits, on constate également un cas de réduction du temps de calcul. Toutefois les deux autres cas manifestent une augmentation du temps de traitement. Cette augmentation est effectivement due à la structure de ces blocs. Du fait de la présence des transistors de passage les cônes construits pour les éléments séquentiels possèdent un grand nombre d'entrées directes. L'énumération de ces entrées directes est le facteur principal de l'augmentation du temps de calcul. Cela explique aussi pourquoi le coût en mémoire ne suit pas l'augmentation du temps de calcul.

8.4.3 La qualité de la modélisation

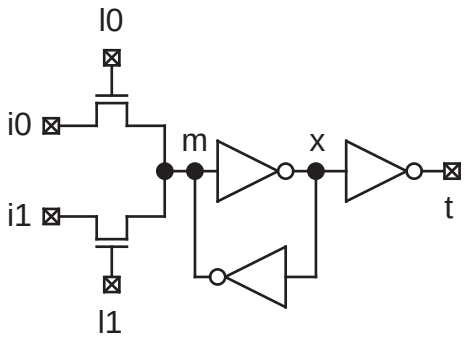
Il est difficile de donner une évaluation objective de la qualité des modèles générés. En effet, nous espérons que la méthode permet de générer des modèles cohérents pour les structures a priori inconnues. Cependant, il faut nous contenter des structures à notre disposition.

En plus, la notion de qualité est subjective. Pour une réalisation de latch donnée, il existe souvent plusieurs modèles correctes. L'objectif de la modélisation automatique est de générer un modèle qui soit non seulement fidèle à la réalisation, mais qui indique clairement le comportement du latch.

Nous avons appliqué la méthode algébrique d'identification des latchs sur l'ensemble des structures d'éléments mémorisants auxquels nous avions accès (une trentaine au total). Nous n'avons constaté aucun cas d'échec pour l'ensemble de ces structures.

Nous présentons ici deux types de latch avec le modèle comportemental généré par YAGLE, afin d'illustrer quelques aspects de la « qualité ».

1) Latch à deux entrées



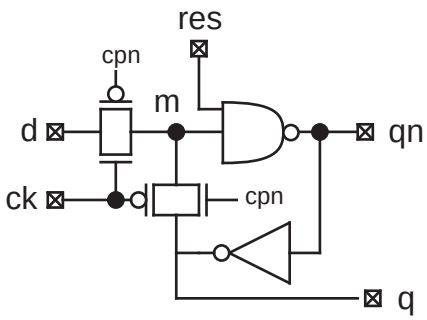

```
BEGIN
  ASSERT ((l0 and l1) = '1')
    REPORT "conflict writing to m"
    SEVERITY ERROR;

  x <= not (m);
  label0 : BLOCK ((l0 and not (l1)) = '1')
  BEGIN
    m <= GUARDED i0;
  END BLOCK label0;
  label1 : BLOCK ((not (l0) and l1) = '1')
  BEGIN
    m <= GUARDED i1;
  END BLOCK label1;

t <= not (x);
END;
```

Dans cet exemple, le latch possède deux émetteurs. Deux expressions conditionnelle sont alors générées. Il n'existe aucune logique pour empêcher que les deux émetteurs soient actifs simultanément. Toutefois, cette possibilité de conflit est détectée lors de l'analyse et une assertion VHDL est générée.

2) Latch avec RESET asynchrone




```
BEGIN
  cpn <= not (ck);
  label0 : BLOCK ((res and not (ck)) = '1')
  BEGIN
    m <= GUARDED d;
  END BLOCK label0;
  label1 : BLOCK ((not (res) and ck) = '1')
  BEGIN
    m <= GUARDED '0';
  END BLOCK label1;

q <= not (qn);

qn <= (not (res) or not (m));
END;
```

Cette structure est un latch avec un reset asynchrone. Le modèle généré reflète le comportement réel, c'est à dire que le reset intervient d'une manière synchrone sur l'état du nœud mémorisant, mais d'une manière asynchrone sur les sortie 'q' et 'qn'.

8.4.4 Conclusion

Dans cette section, nous avons vu que la méthode algébrique permet d'identifier tous les types de latchs utilisés dans les circuits du laboratoire LIP6, et tous les types de latchs

utilisés chez BULL, et que nous avons rencontrés dans les blocs que nous avions à notre disposition.

Ainsi, avec cette méthode, nous possédons un moyen d'identifier les latchs. Evidemment, cette méthode couvre le cas des latchs dont la structure est connue et qui peuvent donc être identifiés par la reconnaissance de formes. Avec cette approche, nous disposons en plus d'une méthode de vérification du comportement mémorisant du latch. Toutefois, l'avantage principal reste que cette méthode permettra de traiter de nouvelles structures sans êtres obligé de les identifier auparavant.

8.5 L'utilisation de l'approche hybride

Cette section est consacrée à la validation de l'approche hybride, qui rassemble la méthode générique et la méthode de reconnaissance programmable à l'aide d'un langage similaire au VHDL.

Pour valider le bon fonctionnement de la méthode, nous devons nous appuyer sur un circuit susceptible d'être traité par YAGLE avec ou sans la méthode de reconnaissance hiérarchique. Nous utilisons à cet effet le générateur de RAM de la chaîne ALLIANCE.

Dans les deux dernières parties, nous illustrons l'utilité d'une approche hybride dans le cas d'un circuit réel pour lequel il était difficile ou même impossible de procéder autrement.

8.5.1 Le générateur de RAM

Le générateur de RAM permet de créer les vues physiques et les spécifications comportementales de plusieurs tailles de RAM. Ces structures sont représentatives de l'architecture classique d'une RAM décrite au chapitre 6. Une RAM est composée de plusieurs colonnes de « bit-cells », avec un mécanisme de précharge et un amplificateur différentiel pour chaque colonne.

En utilisant l'approche hybride, il existe deux manières d'effectuer une abstraction fonctionnelle correcte :

1) Reconnaissance des amplificateurs différentiels et des précharges à l'aide de FCL et l'utilisation de la méthode générique pour le reste du circuit.

2) Reconnaissance hiérarchique par GENIUS de toute la matrice de « bit-cells » avec les amplificateurs différentiels et les précharges.

Dans les deux cas, une description comportementale simulable est générée. Cette description peut être comparée avec la spécification initiale. Nous avons effectué des essais avec plusieurs tailles différentes, nous montrons, à la Figure 8-8, les résultats pour trois tailles différentes afin d'illustrer les tendances.

		Temps du traitement	*Utilisation mémoire*	*Nombre de lignes VHDL*
128 mots, 4 bits	**FCL**	42,8s	13,5 Mo	6 434
(4 516 transistors)	**FCL+GENIUS**	2,9s	4,6 Mo	1 856
256 mots, 4 bits	**FCL**	187.8s	45,7 Mo	12 521
(8 740 transistors)	**FCL+GENIUS**	7.6s	7,6 Mo	4 185
512 mots, 4 bits	**FCL**	859,1s	98,0 Mo	24 752
(17 258 transistors)	**FCL+GENIUS**	24,8s	12,9 Mo	6 757

Figure 8-8 : *L'abstraction fonctionnelle de la RAM*

Les résultats de la Figure 8-8 montre clairement l'avantage de la reconnaissance hiérarchique en temps de traitement et en utilisation mémoire. Ce gain est d'autant plus important si on remarque que les performances ne sont pas linéaires avec la complexité.

De plus, il est clair que l'utilisation de la méthode hiérarchique permet de générer un modèle VHDL bien plus compact et donc beaucoup plus lisible. Il est à noter que le nombre de lignes indiqué pour la méthode hiérarchique correspond à la somme des nombres de lignes dans les trois fichiers générés (comportement de la RAM, comportement du reste et la description structurelle réunissant les deux modèles).

8.5.2 Le circuit ST6

Le circuit ST6 est un cœur de microprocesseur conçu par ST Microelectronic. C'est un circuit « full-custom » de 10 465 transistors, réalisé dans une ancienne technologie de fabrication. Il a été fourni au laboratoire LIP6 dans le cadre d'un contrat de collaboration pour effectuer la migration technologique. Le circuit fourni était sous forme d'un réseau de transistors sans aucune documentation sur les structures internes.

Une première abstraction de ce circuit à été réalisée à l'aide de la méthode générique. Nous avons pu constater que le temps d'exécution et la complexité du code VHDL généré

était excessivement élevés par rapport à la compléxite du circuit (plusieurs heures de calcul et une vingtaine de Mo de code VHDL).

En analysant le réseau obtenu, nous avons identifié plusieurs instances de la structure de la Figure 8-9. Cette structure fortement rebouclée, liée à un style de conception qui s'appuie beaucoup sur les transistors de passage, se prête très mal à un partitionnement en cônes.

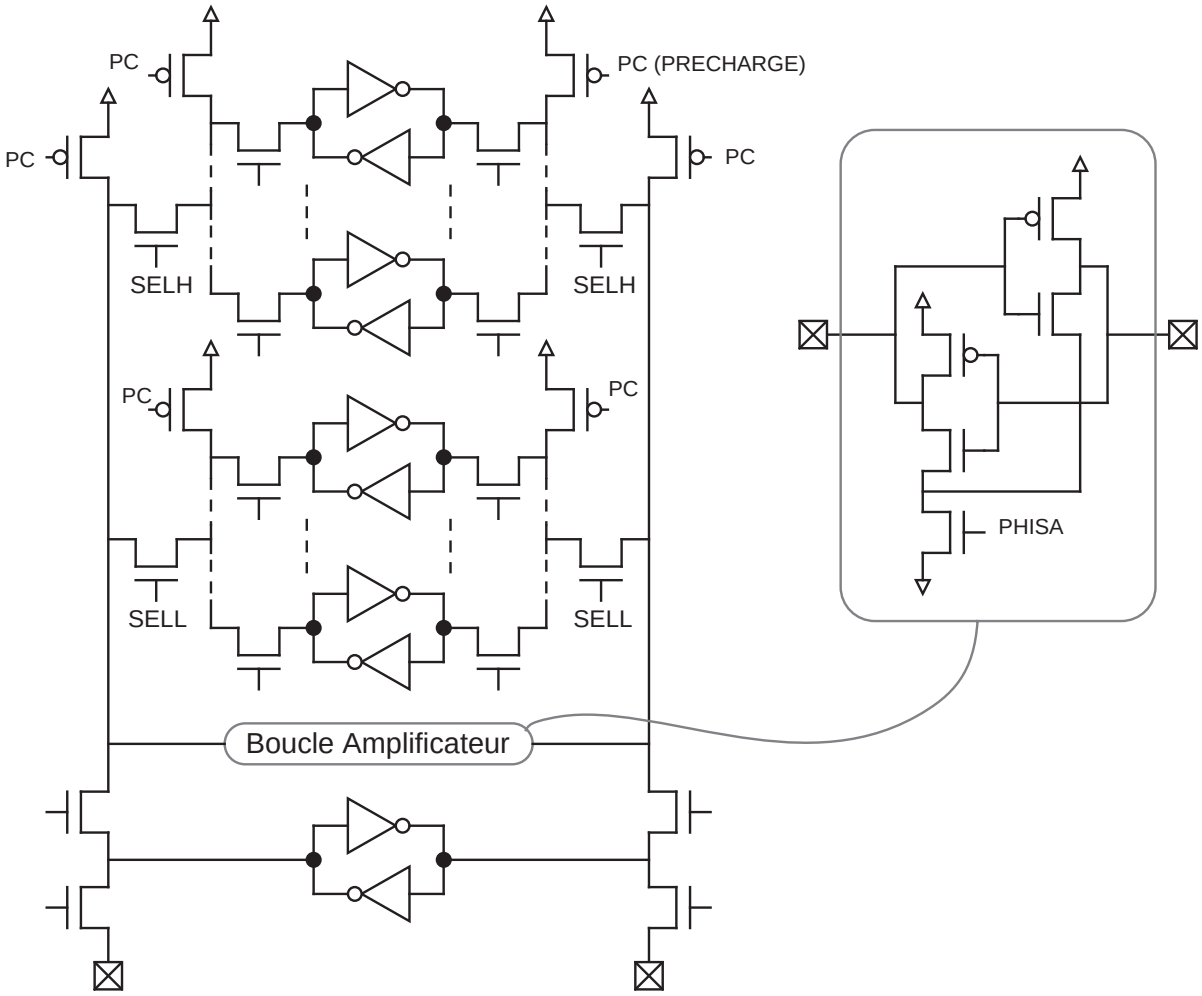


Figure 8-9: *Les registres dans ST6*

Toutefois avec GENIUS, une telle structure est très simple à décrire. Nous avons donc fait l'expérience d'extraire ce composant pour observer l'effet sur la vitesse du désassemblage. La différence dans le temps de traitement et la mémoire consommée démontre clairement l'intérêt de l'approche hybride. Nous avons passé de plusieurs heures à approximativement une minute. De plus le code VHDL obtenu à été largement réduit (à environ 1 Mo).

8.6 Conclusion

Dans ce chapitre, nous avons montré des résultats pendant la validation des trois apports principaux du présent travail. Dans la première section, il s'agit d'une comparaison avec DESB, l'outil précurseur de YAGLE. Cette comparaison à permis de démontrer l'avantage de la gestion des corrélations effectuer par YAGLE.

Avec certaines architectures, DESB maintien une performance linéaire par rapport à la complexité du circuit. Toutefois, cette linéarité est fortement dépendant de l'architecture. Sur des circuits pour lesquels le nombre de fausses branches peut être élevé, les performances chutent avec, dans les pires des cas, la possibilité d'échec selon les ressources disponible. Notre approche, mise en œuvre par YAGLE, adresse ce problème en exploitant systématiquement les corrélations pendant la phase de fabrication des cônes. Cette méthode ajoute une couche de complexité importante sur la fabrication, mais elle permet d'éliminer les fausses branches le plus tôt que possible. Les résultats montrent que cette approche effectivement élimine le risque d'explosion manifesté par DESB. Ainsi la généralité du désassemblage est améliorée considérablement.

Dans la deuxième section de ce chapitre, nous avons essayé de démontrer la faisabilité d'une approche algébrique pour l'identification et la modélisation des éléments séquentiels. Afin de justifier cette méthode originale, il faut non seulement montrer qu'elle permet de reconnaître tous les latchs déjà connus, mais aussi que le coût de cette manipulation supplémentaire reste abordable. Les résultats montrent que, même dans le cas des circuits très complexes, la méthode permet d'identifier tous les latchs avec des coûts tout à fait acceptables. Ainsi on obtient une généralité complète sur le désassemblage des circuits synchrone.

Dans la section finale, nous avons illustré l'utilisation de l'approche hybride sur des exemples réels. Premièrement, nous avons validé la méthode à l'aide du générateur de RAM. Cela a montré qu'il est possible de créer des modèles compacts et simulable avec cette approche. En plus, il est possible, avec une utilisation intelligente de la part du concepteur, d'accélérer le désassemblage. Toutefois, l'aspect le plus intéressant reste la possibilité d'effectuer l'abstraction fonctionnelle sur des circuits pour lesquels cela serait impossible autrement.

Chapitre 9

CONCLUSION

Aujourd'hui la vérification des circuits intégrés couvre des aspects très divers tels que l'analyse de la consommation, la validation des spécifications temporelles ou l'évaluation des bruits. Toutefois, la plupart de ces vérifications nécessitent une étape préliminaire identique : le désassemblage consiste à reconstruire un réseau de portes à partir du réseau de transistors extrait des masques de fabrication. L'abstraction fonctionnelle quant à elle, permet d'identifier le comportement de chaque porte et de lui associer une fonction booléenne.

Dans ce contexte, une méthode de désaasemblage doit être capable de traiter des circuits entiers ou des blocs de plusieurs centaines de milliers, voire des millions, de transistors. De plus, le temps, ainsi que les ressources mémoires nécessaires à ce traitement, doivent rester dans la mesure du possible linéaires par rapport à la complexité du circuit. Cette linéarité ne doit pas dépendre du style de conception du circuit.

Nous avons vu dans « l'état de l'art », qu'il existe deux approches distinctes pour effectuer le désassemblage et l'abstraction fonctionnelle : la reconnaissance de formes et les méthodes génériques. En principe, ce sont les méthodes génériques qui satisfont le mieux l'exigence de la généralité.

Dans cette thèse, nous nous plaçons dans le cas des technologies CMOS qui domine largement le marché des circuits intégrés, et nous nous sommes intéressés au développement d'une méthode de désassemblage capable de répondre aux besoins des outils de vérification et des concepteurs.

Au Laboratoire LIP6, un premier outil de désassemblage, appelé DESB, a été développé. La problématique du présent travail était d'apporter des solutions aux deux faiblesses principales de la méthode de désassemblage mise en œuvre par DESB. La première faiblesse se situe dans l'ignorance des corrélations pendant la fabrication des cônes. La deuxième réside dans la méthode d'identification des éléments séquentiels.

En ce qui concerne la première faiblesse- l'exploitation des corrélations- la solution que nous avons apportée est une restructuration complète de la méthode de fabrication des cônes. Avec la méthode mise en œuvre par DESB, la phase de fabrication des cônes consistait simplement à tracer tous les chemins (branches) entre les nœuds qui attaquent au moins une grille de transistor et les sources de tension externes. C'est uniquement à la fin du désassemblage qu'un post-traitement est effectué pour essayer d'éliminer les fausses branches. Cette

stratégie a rendu les performances de DESB très dépendantes du style de conception du circuit.

Au chapitre 4 nous avons proposé une solution consistant à conditionner la construction des branches à l'existence d'un contexte logique. Pour chaque cône nous identifions, avant de commencer sa construction, les corrélations qui pourraient empêcher la construction de certaines branches. Ainsi la construction des fausses branches est bloquée dès que possible. Les résultats ont montré que cette approche apporte non seulement une meilleure performance en générale, mais aussi réduit de manière significative l'influence du style de conception sur les performances.

Pour palier à la deuxième faiblesse – l'identification des éléments séquentiels – nous avons conçu une approche totalement originale. Cette approche s'appuie sur une analyse des expressions booléennes qui caractérise une boucle dans un circuit. Cette analyse consiste à la résolution des faux conflits fonctionnels et des faux conflits électriques afin de générer une expression booléenne pour la boucle. Ensuite, nous utilisons la dérivée partielle booléenne pour effectuer une analyse de la stabilité et de la capacité de mémorisation d'une boucle.

Nos résultats ont montré que l'approche permet, au prix d'un coût qui reste abordable, d'identifier et de modéliser tous les éléments séquentiels que nous avons pu rencontrer. L'avantage clé de cette approche repose sur sa généralité. L'utilisation de cette méthode d'analyse algébrique, combinée avec la méthode de l'exploitation des corrélations pendant la fabrication des cônes améliore considérablement la généralité du désassemblage.

Toutefois, il reste toujours des cas pour lesquels il est difficile, impossible ou même indésirable de traiter le circuit à l'aide d'une approche totalement automatique. Pour affronter ces situations, nous avons développé une approche hybride et nous avons montré qu'une approche par reconnaissance de formes n'est pas forcément incompatible avec une méthode générique. Nous pensons qu'une réponse pragmatique au problème du désassemblage et de l'abstraction fonctionnelle réside dans l'utilisation d'une approche hybride. En effet, une telle approche offre au concepteur la possibilité de contrôler le processus de désassemblage et de trouver ainsi un compromis acceptable entre les performancesde l'outil (le temps de traitement, la consommation mémoire et la qualité du résultat) et la quantité d'informations à fournir.

BIBLIOGRAPHIE

[Abd95a] N. Abdallah, J. Dunoyer, P. Bazargan-Sabet, "A New Generation of VLSI CAD Tools Based on Probability", *In Proceedings of 27th IEEE Southeastern Symposium on System Theory*, pp. 348-352, Mississippi, USA, Mars 1995.

[Abd95b] N. Abdallah, P. Bazargan-Sabet, A. Greiner, "On the Design of Mixed-Mode Sirnulators for Modern VLSI Circuits", *In Proceedings of 38th IEEE Midwest Symposium on Circuits and Systems*, pp. 1168-1171, Rio de Janeiro, Brazil, Août 1995.

[Abd98] N. Abdallah, "Méthode de simulation logico-temporelle de circuits numériques complexes prenant en compte le front des signaux et les collisiuns dans le cadre de la simulation mixte analogique-numérique", Thèse de Doctorat, Université Pierre et Marie Curie, Laboratoire LIP6, Février 1998.

[Abl81] I. Ablasser, U. Jäger, "Circuit Recognition and Verification Based on Layout Information", *In Proceedings of 18th ACM/IEEE Design Automation Conference*, Nashville, Tennessee, USA, Juin 1981.

[Ake78] S. B. Akers, "Binary Decision Diagrams", *IEEE Transactions on Computers*, Vol. 27, No. 6, pp. 509-516, Juin 1978.

[Ama93] A. Amara, A. Greiner, L. Lucas, F. Pétrot, Franck Wajsbürt, L. Winckel, "The Portable Libraries of the Alliance CAD System", *In Proceedings of VIII SB Micro International Conference on Microelectronics and Packaging*, pp. III.14-III.18, Curitiba, Brazil, Septembre 1993.

[Apt82] R. Apte, N. S. Chang, J. A. Abraharn, "Logic Function Extraction for Nmos Circuits", *In Proceedings of IEEE International Conference on Circuits and Computers*, New York, USA, Octobre 1982.

[Bar88] Z. Barzilai, D. K. Beece, L. M. Huisman, V. S. Iyengar, G. M. Silberman, "SLS – A Fast Switch-Level Simulator", *IEEE Transactions on Computer Aided Design*, Vol. 7, No. 8, pp. 838-849, Aout 1988.

[Baz93] P. Bazargan-Sabet, A. Greiner et H. N. Vuong, "ASIMUT: A Public Domain VHDL Sirnulation Tool", *In Proceedings of 4th Eurochip Workshop*, pp. 62-65, Toledo, Spain, Septembre 1993.

[Bla89] D. T. Blaauw, D. G. Saab, R. B. Mueller-Thuns, J. A. Abraham, J. T. Rahmeh, "Automatic Generation of Behavioural Models from Switch-Level Description", *In Proceedings of 26th ACM/IEEE Design Automation Conference*, Las Vegas, USA, Juin 1989.

[Bla90] D. T. Blaauw, R. B. Mueller-Thuns, D. G. Saab, P. Banerjee, "SNEL- A Switch-Level Sirnulator using Multiple Levels of Functional Abstraction", *In Proceedings of IEEE International Conference on Computer Aided Design*, Santa Clara, CA, USA, Novembre 1990.

[Bla91] D. T. Blaauw, D. G. Saab, P. Banerjee, J. A. Abraham, "Functional Abstraction of Logic Gates for Switch-Level Simulation", *In Proceedings of 2nd IEEE European Design Automation Conference*, Amsterdam, Netherlands, Février 1991.

[Boe88] M. Boehner, "LOGEX - An Automatic Extractor from Transistor to Gate Level for CMOS Technology", *In Proceedings of 25th ACM/IEEE Design Automation Conference*, pp. 517-522, Anaheim, CA, USA, Juin 1988.

[Bol89] I. Bolsen, W. DeRammelaere, L. Claesen, H. DeMan, "Electrical debugging of synchronous MOS VLSI circuits exploiting analysis of the intended behaviour", *In Proceedings of 26th ACM/IEEE Design Automation Conference*, Las Vegas, Juin 1989.

[Bry81] R. E. Bryant, "MOSSIM: A Switch-Level Simulator for MOS LSI", *In Proceedings of 18th ACM/IEEE Design Automation Conference*, Nashville, Tennessee, USA, Juin 1981.

[Bry84] R. E. Bryant, "A Switch-Level Model and Simulator for MOS Digital Systems", *IEEE Transactions on Computers*, Vol. 33, No. 2, pp. 160-177, Février 1984.

[Bry86] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation", *IEEE Transaction on Computers*, Vol. 35, No. 8, pp 677-691, Août 1986.

[Bry87a] R. E. Bryant, "Algorithmic Aspects of Symbolic Switch Network Analysis", *IEEE Transactions on Computer-Aided Design*, Vol. 6, No. 4, pp. 618-633, Juillet 1987.

[Bry87b] R. E. Bryant, "Boolean Analysis of MOS Circuits", *IEEE Transactions on Computer-Aided Design*, Vol. 6, No. 4, pp. 634-649, Juillet 1987.

[Bry87c] R. E. Bryant, "COSMOS : a compiler simulator for MOS circuits", *In Proceedings of 24th ACM/IEEE Design Automation Conference*, pp. 9-16, Miami Beach, FL, USA, Juin 1987.

[Bry91] R. E. Bryant, "Extraction of Gate Level Models from Transistors Circuits by Four Valued Symbolic Analysis", *In Proceedings of IEEE International Conference on Computer Aided Design*, Santa Clara, CA, USA, Novembre 1991.

[Bur83] G. Burrow, R. Apte, "Topology Reduction for Cmos and Dynamic Nmos VLSI Circuits", *In Proceedings of IEEE International Conference on Computer Aided Design*, Santa Clara, CA, USA, Septembre 1983.

[But91] K. M. Butler, D. E. Ross, R. Kapur, M. R. Mercer, "Heuristics to Compute Variable Orderings for Efficient Manipulation of Ordered Binary Decision Diagrams", *In Proceedings of 28th ACM/IEEE Design Automation Conference*, San Francisco, CA, USA, Juin 1991.

[Cas96] F. Casaubieilh, et al., "Functional Verification Methodology of Chameleon Processor", *In Proceedings of 33rd ACM/IEEE Design Automation Conference*, pp. 421-426, Las Vegas, USA, Juin 1987.

[Cor80] D. G. Corneil, D. G. Kirkpatrick, "A theoretical analysis of various heuristics for the graph isomorphism problem", *SIAM Journal on Computing*, Vol. 9, No. 2, pp. 281-297, Mai 1980.

[Deb91] A. Debreil, "Vhdl Formal Proof", Bull New CAD Project, Les Clayes sous Bois, 1991.

[Dev92] P. Deverchère, J. C. Madre, J. B. Guignet, M. Currat, "Functional Abstraction and Formal Proof of Digital Circuits", *In Proceedings of 3rd IEEE European Design Automation Conference*, Brussels, Belgium, Mars 1992.

[Dit83] G. Ditlow, W. Donath, A. Ruehli, "Logic Equations for Mosfet Circuits", *In Proceedings of IEEE International Symposium on Circuits and Systems*, pp.752-755, Newport Beach, CA, USA, Mai 1983.

[Dun99] J. Dunoyer, "Modèles et methodes probabilistes pour l'evaluation de la consommation des circuits integres VLSI", Thèse de Doctorat, Université Pierre et Marie Curie, Laboratoire LIP6, Juillet 1999.

[Ebe83] C. Ebeling, O. Zajicek, "Validating VLSI Circuit Layout by Wirelist Comparison", *In Proceedings of IEEE International Conference on Computer Aided Design*, Santa Clara, CA, USA, Septembre 1983.

[Ebe88] C. Ebeling, "GeminiII: A Second Generation Layout Validation Program", *In Proceedings of IEEE International Conference on Computer Aided Design*, pp. 322-325, Santa Clara, CA, USA, Novembre 1988.

[Epi97] EPIC Design Technology, "PathMill User's Manual", 1997.

[Fri87] S. J. Friedman, K. J. Supowit, "Finding the Optimal Variable Orderiing for Binary Decision Diagrams", *In Proceedings of 24th ACM/IEEE Design Automation Conference*, Miami Beach, Florida, USA, Juin 1987.

[Geo94] B. J. George, D. Gossain, S. C. Tyler, M. G. Wloka, G. K. H. Yeap, "Power analysis and characterization for semi-custom design", *In Proceedings of International Workshop on Low-Power Design*, pp. 215-218, Napa, CA, USA, Avril 1994.

[Gib85] A. Gibbons, "Algorithmic Graph Theory", Cambridge University Press, 1985.

[Gin86] J. Van Ginderdeuren et al, "A High Quality Digital Audio Filter Set Designed by Silicon Compiler CATHEDRAL-1", *IEEE Journal of Solid State Circuits*, December 1986.

[Gon79] M. Gontron, M. Minoux, "Graphes et Algorithmes", Eyrolles, 1979.

[Gre92] A. Greiner, F. Pécheux, "Alliance: A Complete Set of CAD Tools for Teaching Digital VLSI Design", *In Proceedings of 3rd Eurochip Workshop on VLSI Design Training*, pp. 230-237, Grenoble, France, Septembre 1992.

[Gre94] A. Greiner, F. Pétrot, "Designing Portable Module Generator Using The Alliance CAD system", *In Proceedings of 26th Southeastern Symposium on System Theory*, pp. 395-399, Athens, Ohio, USA, Mars 1994.

[Gru97] W. J. Grundmann, D. Dobberpuhl, R. L. Allmon, N. L. Rethman, "Designing High Performance CMOS Microprocessors Using Full Custom Techniques", *In Proceedings of 34th ACM/IEEE Design Automation Conference*, pp. 722-727, Anaheim, CA, USA, Juin 1997.

[Gui98] J.-B. Guignet, "Abstraction Fonctionnelle des Composants VLSI", Thèse de Doctorat, Université Pierre et Marie Curie, Laboratoire LIP6, Mars 1998.

[Haj83] I. N. Hajj, D. Saab, "Symbolic Logic Simulation of MOS Circuits", *In Proceedings of IEEE International Symposium on Circuits and Systems*, pp.246-249, Newport Beach, CA, USA, Mai 1983.

[Haj91] A. Hajjar, R. Marbot, A. Greiner, P. Kiani, "TAS: An Accurate Timing Analyzer for CMOS VLSI", *In Proceedings of 2nd IEEE European Design Automation Conference*, pp. 261-265, Amsterdam, Netherlands, Février 1991.

[Hui88] J. A. Huisken et al. , "Design of DSP Systems using the PIRAMID Library and Design Tools", *In Proceedings of IEEE VLSI Signal Processing III*, 1988.

[IEE88] IEEE Computer Society, "IEEE Standard VHDL Language Reference Manual", The Institute of Electrical and Electronic Engineering, New York, Publ. No: IEEE Standards Coordinating committee 20, 1988.

[Jai91] A. Jain, R. E. Bryant, "Mapping Switch-Level Simulation onto Hardware Accelerators", *In Proceedings of 28th ACM/IEEE Design Automation Conference*, San Francisco, CA, USA, Juin 1991.

[Jou87] N. P. Jouppi, "Derivation of Signal Flow Direction in MOS VLSI", *IEEE Transactions on Computer Aided Design*, Vol. 6, No. 3, pp. 480-490, Mai 1987.

[Kaw82] M. Kawamura, K. Hirabayashi, "Logical Verification of LSI Mask Artwork by Mixed Level Simulation", *In Proceedings of IEEE International Symposium on Circuits and Systems*, Rome, Italy, Mai 1982.

[Kis83] A. Kishimoto, K. Mori, S. Takashi, H. Kawanishi, "A Logic Function Extraction Algorithm for MOS VLSI", *In Proceedings of IEEE International Conference on Computer Aided Design*, Santa Clara, CA, USA, Septembre 1983.

[Kos91] T. Kostelijk, B. De Loore, "Automatic Verification of Library-Based IC Designs", *IEEE Journal of Solid-State Circuits*, Vol. 26, No. 3, pp. 394-403, Mars 1991.

[Lau92] M. Laurentin, A. Greiner, R. Marbot, "DESB, a Functional Abstractor for CMOS VLSI", *In Proceedings of 3rd IEEE European Design Automation Conference*, Brussels, Belgium, Mars 1992.

[Lau94] M. Laurentin, "Abstraction fonctionnelle de circuits intégrés CMOS", Thèse de Doctorat, Université Pierre et Marie Curie, Laboratoire MASI, Décembre 1994.

[Les97] A. Lester, P. Bazargan-Sabet, "Un Outil d'Evaluation de la Consommation Basé sur l'Extraction d'un Réseau de Portes Caractérisées", *Colloque CAO de circuits intégrés et systèmes*, pp. 128-131, Grenoble, France, Janvier 1997.

[Les98a] A. Lester, P. Bazargan-Sabet, A. Greiner, "Circuit Disassembly for Verification and Functional Abstraction of CMOS Circuits", *In Proceedings of 1st Sophia Antipolis Forum on Microelectronics*, Sophia Antipolis, France, Octobre 1998.

[Les98b] A. Lester, P. Bazargan-Sabet, A. Greiner, "YAGLE, a Second Generation Functional Abstractor for CMOS VLSI Circuits", *In Proceedings of 10th International Conference on Microelectronics*, Monastir, Tunisia, Décembre 1998.

[Lue84] F. Luellau, T. Hoepken, E. Barke, "A Technology Independent Block Extraction Algorithm", *In Proceedings of 21st ACM/IEEE Design Automation Conference*, Albuquerque, New Mexico, USA, Juin 1984.

[Mad88] J. C. Madre, J. P. Billon, "Proving Circuit Correctness using Formal Comparison Between Expected and Extracted Behavior", *In Proceedings of 25th ACM/IEEE Design Automation Conference*, Juin 1988.

[Man85] H. J. de Man et al, "DIALOG, an expert debugging system for CMOS VLSI design", *IEEE transactions on Computer Aided Design*, Vol. 4, No. 3, pp. 303-311, Juillet 1985.

[Mar94] R. Marculescu, D. Marculescu, M. Pedram, "Logic Level Power Estimation Considering Spatiotemporal Correlations", *In Proceedings of IEEE International Conference on Computer Aided Design*, San Jose, CA, USA, Novembre 1994.

[Mcl96] J. McLeod, "The State of the Art in IC Layout Migration", *Integrated System Design Magazine*, Juin 1996.

[Nag75] L. W. Nagel, "SPICE2: A computer program to simulate semiconductor circuits", ERL Memo ERL-M520, University of Berkekey, California, Mai 1975.

[Naj90] F. N. Najm, R. Burch, P. Yang, 1. N. Hajj, "Probalistic simulation for reliability analyses of CMOS VLSI circuits", *IEEE Transactions on Computer-Aided Design*, Vol. 9, No. 4, pp. 439-450, Avril 1990.

[Naj91] F. N. Najm, "Transition density, a stochastic measure of activity in digital circuits", *In Proceedings of 28th ACM/IEEE Design Automation Conference*, pp. 644-649, San Francisco, CA, USA, Juin 1991.

[Neb87] W. Nebel, R. W. Hartenstein, "Functional Design Verification by Register Transfer Net Extraction", *In Proceedings of COMPEURO*, Hamburg, Germany, Mai 1987.

[Ohl93] M. Ohlrich, C. Ebeling, E. Ginting, L. Sather, "SubGemini: Identifying Subcircuits using a Fast Subgraph Isomorphism Algorithm", *In Proceedings of 30th ACM/IEEE Design Automation Conference*, Dallas, Texas, USA, Juin 1993.

[Ous85] J. K. Ousterhout, "A Switch-Level Timing Verifier for Digital MOS VLSI", *IEEE Transactions on Computer Aided Design*, Vol. 4, No. 3, pp. 336-349, Juillet 1985.

[Pel91] G. Pelz, U. Roettcher, "Circuit Comparison by Hierarchical Pattern Matching", *In Proceedings of IEEE International Conference on Computer Aided Design*, pp. 290-293, Santa Clara, CA, USA, Novembre 1991.

[Rea77] R. C. Read, D. G. Corneil, "The Graph Isomorphism Disease", *Journal of Graph Theory*, pp. 339-363, Janvier 1977.

[Ree87] D. S. Reeves, M. J. Irwin, "Fast Methods for Switch-Level Verification of MOS Circuits", *IEEE Transactions on Computer-Aided Design*, Vol. 6, No. 5, pp. 766-779, Septembre 1987.

[Saa82] D. G. Saab, I. N. Hajj, "A Logic Expression Generator for MOS Circuits", *In Proceedings of IEEE International Conference on Circuits and Computers*, pp.328-331, New York, USA, Octobre 1982.

[Shi68] H. Shichman, and D. A. Hodges, "Modeling and Simulation of Insulated Gate Field Effect Transistor Switching Circuits", *IEEE Journal of Solid State Circuits*, Septembre 1968.

[Ste92] R. Stewart, V. Anjubault, P. Garcin, J. Benkoski, "Automatic Import of Custom Designs into Cell-Based Environment using Switch-Level and Circuit Simulation", *In Proceedings of 3rd IEEE European Design Automation Conference*, Brussels, Belgium, Mars 1992.

[Szy73] S. A. Szygenda, A. A. Lekkos, "Integrated Techniques for Functional and Gate-Level Digital Logic Simulation", *In Proceedings of 10th ACM/IEEE Design Automation Workshop*, Juin 1973.

[Tak82] M. Takashima, T. Ntsthuhashi, T. Chiba, K. Yoshida, "Programs for Verifying Cicuit Connectivity of MOS/LSI Mask Artwork", *In Proceedings of 19th ACM/IEEE Design Automation Conference*, Las Vegas, USA, Juin 1982.

[Wu87] C. F. E. Wu, L. M. Ni, A. S. Wojcik, "Functional Recognition of Static Cmos Circuits", *In Proceedings of IEEE International Conference on Computer Aided Design*, Santa Clara, CA, USA, Novembre 1987.

[Wu91] C. F. E. Wu, A. S. Wojcik, L. M. Ni, "Rule-Based Circuit Representation for Automated CMOS Design and Verification", *International Journal of Computer Aided VLSI Design*, Vol. 3, No. 2, pp. 217-240, 1991.