UNIVERSITÉ PARIS VI
PIERRE ET MARIE CURIE
U.F.R. D'INFORMATIQUE

THÈSE
Pour obtenir le titre de
DOCTEUR DE L'UNIVERSITÉ PARIS VI

Présentée par
Olivier FLORENT

Sujet :

UNE MÉTHODE DE TEST DES CIRCUITS INTÉGRÉS, BASÉE SUR UN DÉCOUPAGE STRUCTUREL PEU RECOUVRANT

Soutenue le 30 juin 1998,
devant le jury composé de

M. Christian	**LANDRAULT**	Rapporteur
M. Régis	**LEVEUGLE**	Rapporteur
M. Alain	**GREINER**	Examinateur
Mme Anne	**DERIEUX**	Examinateur
M. Christian	**NIQUET**	Invité

Avant-propos

J'ai effectué ces travaux au sein du laboratoire d'informatique de Paris VI de l'université Pierre et Marie Curie, sous la direction de M. le professeur Alain Greiner, responsable de l'équipe Architecture des Systèmes Intégrés et Micro-électronique. Je tiens à le remercier pour m'avoir fait confiance au cours de ces années, et pour le soin tout particulier avec lequel il a suivi la rédaction de ce manuscrit.

Je remercie chaleureusement Mme Anne Derieux, maître de conférences au laboratoire LIP6, qui a consacré beaucoup de temps à l'encadrement de ces travaux. Elle a su, par sa disponibilité, sa bonne humeur et ses conseils, être un soutien indispensable.

Je suis très honoré que M. Landrault, directeur de recherche au CNRS au LIRMM à Montpellier, et M. Leveugle, maître de conférences au laboratoire CSI de l'INPG à Grenoble, aient acceptés d'être les rapporteurs de ces travaux. Je remercie également M. Niquet, de la société BULL, d'avoir bien voulu en être examinateur.

L'environnement de travail au LIP6 est riche de compétences et d'amitié. Ces pages seraient trop courtes pour exprimer ma gratitude à toutes ces personnes qui, de la rotonde à la salle 201 en passant par notre petit bureau, m'ont bien aidé et m'ont permis de passer de très bon moments, et qui, j'en suis sûr, se reconnaissent ici. J'adresse une petite pensée spéciale à Mokhtar, qui m'a initié au joies du test, et à Housseine, qui a été mon compère pendant ces années.

Je tiens à remercier M. Windmiller et mes nouveaux collègues de Hewlett-Packard à Grenoble : ils ont su m'accueillir et me procurer un environnement propice pour que je puisse terminer sereinement ma rédaction.

Enfin, je n'aurai jamais pu aller jusqu'au bout de ce travail sans le soutien quotidien et indéfectible de Katell, et sans le petit sourire de Manon.

Merci à tous.

Résumé

Nous nous plaçons dans le cadre d'un circuit possédant déjà des structures de test. Nous avons observé que les circuits intégrés d'aujourd'hui possèdent tous des chemins de scan-path, certains utilisent le test intégré ou des macro-cellules, et la plupart ont une architecture Boundary-Scan. Dans ces circuits, l'utilisation de blocs issus de bibliothèques ou de fabricants différents, ou encore la réutilisation de composants, ne permet pas de maîtriser l'implantation des structures de test. Nous proposons d'effectuer un découpage du circuit autour de ses structures de test en unités testables indépendantes de complexité raisonnable vis à vis des outils de génération de vecteurs de test.

Nous effectuons une identification automatique des points observables et contrôlables dans la représentation structurelle du circuit à l'aide de simulations. Puis nous découpons le circuit en unités testables. Puisque nous ne maîtrisons pas l'emplacement des points observables et contrôlables du circuit, ces unités testables peuvent être recouvrantes. Nous essayons de rendre le découpage le moins recouvrant possible. Nous n'incluons pas dans ces unités testables les portes venant de l'architecture Boundary-Scan, ni les blocs incluant leur propre mécanisme de test. Les unités testables étant indépendantes, nous pouvons distribuer la génération sur un réseau de stations de travail.

Nous proposons une technique d'assemblage des vecteurs de test et des dictionnaires de fautes issus de la génération, afin d'obtenir une séquence unique et un dictionnaire unique pour l'ensemble du circuit, sans effectuer une simulation de fautes globale. Nous parallélisons les vecteurs issus d'unités testables non-recouvrantes ainsi que ceux issus d'unités testables peu recouvrantes.

Nous avons realisé un prototype logiciel s'appuyant sur la chaîne ALLIANCE et sur le générateur de vecteurs de test commercial HITEST. Ce prototype nous a permis de confronter notre méthode à des circuits réels.

Mots-clés

test des circuits intégrés, Boundary-scan, test intégré, réutilisation de composants, découpage automatique, génération de vecteurs de test, distribution.

A TEST METHOD FOR INTEGRATED CIRCUITS, BASED ON A SPLITTING AND OVERLAPPING STRATEGY

Abstract

Our study presents a "divide and conquer" strategy for digital circuit testing, in order to generate deterministic patterns using classical ATPG tools. Today's integrated circuits have scan-paths, some are using built-in self-test or macro-cells, and almost all have a Boundary-Scan architecture. Unfortunately, design reuse and heterogeneous libraries do not allow designers to choose the style and the location of the test structures. We propose a splitting process which constructs overlapping testable units around the test structures. Theses testable units can be processed independently by ATPG tools across a network of workstations.

The boundaries of the testable units are the observable and controllable signals of the structural flattened view of the circuit. We identify automatically these signals, using logical and symbolic simulations. Next the circuit is split in testable units, and the overlap factor due to this splitting process is minimized. Gates from Boundary-scan logic, or gates from area with built-in self test are not included in testable units. We can now obtain test vectors and a fault dictionary for each testable unit using a classic ATPG tool.

We merge all the test vectors and the fault dictionaries to obtain one global test sequence and one global fault dictionary for the circuit. Test vectors from testable units are run in parallel to save test length.

A set of tools based on our test method has been realized in the framework of the ALLIANCE CAD system, using HITEST, a commercial product, as the APTG tool. With these tools, we have been able to process a real circuit which integrates macro-cells.

Keywords

VLSI testing, Boundary-scan, built-in self-test, design reuse, automatic partitioning, ATPG tool, distribution.

Table des matières

Introduction **13**

1 Problématique **17**
1.1 Les problèmes posés par le test aujourd'hui 17
1.1.1 La complexité du circuit 19
1.1.2 La diversité des structures de test 20
1.1.3 La réutilisation de composants 21
1.2 Notre approche 22

2 État de l'art **23**
2.1 La génération des tests dans l'industrie 23
2.1.1 IBM : le 370 23
2.1.2 Motorola : Le 68040 et le 68060 25
2.1.3 IBM & Motorola : le PowerPC 603 27
2.1.4 Hewlett-Packard : le PA7100LC 28
2.1.5 Texas Instrument : le TMS320C80 29
2.1.6 Sun et Texas-Instruments : le SuperSPARC 30
2.1.7 La génération des tests chez Intel 31
2.1.8 Le Macro Test de Philips 32
2.1.9 Conclusion 34
2.2 Découpage automatique des circuits en vue du test 35
2.2.1 Prospect 35
2.2.2 Crete 36
2.2.3 Contest 38
2.2.4 Partitionnement fonctionnel 39
2.3 Conclusion 40

3 Une méthode de test par découpage recouvrant **43**
3.1 Prérequis 44
3.1.1 Niveau de description du circuit 44
3.1.2 Testabilité du circuit 45
3.1.3 Mise en œuvre de la testabilité 45
3.2 Présentation générale de la méthode 46

3.3 Description du circuit comme un réseau booléen 48
3.3.1 Représentation d'une porte logique 49
3.3.2 Représentation d'un registre ou d'un bus 49
3.3.3 Représentation d'un « tristate » 51
3.3.4 Représentation d'un réseau de portes 52
3.3.5 Marqueurs 53
3.4 Conclusion 53

4 Reconnaissance automatique de signaux observables et contrôlables 55
4.1 Techniques de simulation 56
4.1.1 La simplification du réseau booléen 56
4.1.2 La simulation symbolique 59
4.2 Application à la reconnaissance automatique des signaux observables et contrôlables dans un circuit 63
4.2.1 Identification des registres observables et contrôlables . . . 65
4.2.2 Identification des signaux observables et contrôlables . . . 67
4.2.3 Identification de la logique de contrôle 68
4.2.4 Exemple 68
4.3 Conclusion 69

5 Découpage en unités testables faiblement recouvrantes 71
5.1 Découpage en unités testables élémentaires 71
5.1.1 Algorithme de découpage 72
5.1.2 Description simplifiée d'une unité testable élémentaire . . . 74
5.2 Regroupement d'unités testables élémentaires 75
5.2.1 Fort recouvrement des unités testables élémentaires 75
5.2.2 Complexité et nombre des unités testables élémentaires . . 77
5.2.3 Formation des unités testables 77
5.3 Conclusion 84

6 Prise en compte de blocs auto-testés 85
6.1 Présentation générale de l'auto-test 85
6.1.1 Génération intégrée de vecteurs de test 86
6.1.2 Analyse intégrée des résultats 87
6.2 L'auto-test déterministe des mémoires 88
6.3 Auto-test pseudo-aléatoire 91
6.4 Interaction avec la méthode de découpage 92
6.4.1 Isolement intégré de blocs auto-testés 94
6.4.2 Isolement périphérique de blocs auto-testés 95
6.4.3 Génération de vecteurs de test avec un dictionnaire initial 96
6.4.4 Construction d'un dictionnaire de fautes homogène 97
6.5 Conclusion 97

7 Assemblage de test **99**
7.1 Assemblage des vecteurs d'unités testables 99
7.1.1 Traduction de vecteurs 100
7.1.2 Parallélisation du test 103
7.1.3 Multiples *scan-path* 107
7.1.4 Efficacité de la parallélisation 108
7.1.5 Interaction du découpage sur la longueur de test 109
7.1.6 Ajout des vecteurs des unités auto-testées 110
7.2 Prise en compte du *Boundary-Scan* 111
7.2.1 Description de l'architecture *Boundary-Scan* 111
7.2.2 Méthode de reconnaissance 113
7.2.3 Interaction du *Boundary-Scan* avec la méthode de découpage et d'assemblage 116
7.3 Assemblage des dictionnaires de fautes 116
7.4 Conclusion 118

8 Mise en œuvre **119**
8.1 Prototype logiciel 119
8.1.1 L'outil de découpage 119
8.1.2 L'outil *HITEST* de la chaine HILO 123
8.1.3 L'outil d'assemblage des tests 125
8.2 Validation de la méthode sur un circuit réel : PCI-DDC 126
8.2.1 Présentation du circuit 126
8.2.2 Testabilité du circuit 127
8.2.3 Description du circuit 130
8.2.4 Application de la méthode sur PCI-DDC 130
8.3 Conclusion 132

Conclusion **133**

Annexe 1 **137**

Bibliographie **141**

Introduction

Dans le domaine de la production des tests pour les circuits intégrés, deux écoles se sont toujours opposées. La plus ancienne rassemble les ingénieurs de conception. Ils considèrent souvent que la mise au point des vecteurs de test est une étape purement technique qui intervient en fin de conception et qui doit s'adapter à leurs circuits. De l'autre coté, les ingénieurs de test revendiquent une prise en compte des problèmes liés au test de plus en plus tôt, et de manière de plus en plus contraignante. L'évolution actuelle de la complexité des circuits, et les coûts engendrés par le test donnent de plus en plus souvent raison à ces derniers. Nous passons donc progressivement d'une approche de conception « suivi du test » à une approche de conception « en vue du test ». C'est ainsi qu'une grande majorité des circuits conçus aujourd'hui possèdent des chemins de *scan-path*, que depuis sa normalisation le *Boundary-Scan* est présent dans tous les circuits commerciaux, et que le test intégré se démocratise.

Pour concevoir un circuit en vue du test il faut maîtriser totalement sa testabilité. Il est alors possible de faire les choix qui permettront de réduire les coûts liés au test. Cela se traduit par l'utilisation de composants à l'intérieur du circuit qui ne serviront qu'à le tester. Ce sont, par exemple, des portes logiques supplémentaires qui permettent de partitionner le circuit en sous-parties indépendantes pour le tester.

Cela n'est plus vrai lorsque les concepteurs ne maîtrisent qu'une partie de la testabilité, par exemple lors de l'utilisation de macro-cellules ou de bibliothèques issues de fabricants différents. Le test doit, dans ce cas, de nouveau s'adapter au circuit. Les outils actuels de génération automatique de vecteurs de test et de simulation de fautes ne sont cependant pas adaptés aux complexités de ces circuits qui dépassent souvent le million de transistors. De plus, la diversité des structures de test employées complique encore la tâche de ces outils. Il est donc indispensable de réduire cette complexité grâce à un découpage du circuit en parties plus modestes.

Dans cette thèse, nous nous plaçons dans le cadre d'un circuit possédant déjà des structures de test. Nous proposons d'effectuer un découpage du circuit autour de ses structures de test en unités testables indépendantes de complexité raisonnable vis à vis des outils automatiques de génération de vecteurs de test.

Nous effectuons une identification automatique des points observables et contrôlables dans la représentation structurelle du circuit. Puis nous découpons le circuit en unités testables. Puisque nous ne maîtrisons pas l'emplacement des points observables et contrôlables, ces unités peuvent être recouvrantes. Nous essayons de rendre le découpage le moins recouvrant possible, et nous n'incluons pas dans les unités testables les portes venant d'une architecture *Boundary-Scan*, ni celles appartenant à des parties qui possèdent leur propre mécanisme de test. Les unités testables étant indépendantes, nous pouvons distribuer la génération sur un réseau de machines.

Nous proposons une technique d'assemblage des vecteurs de test et des dictionnaires de fautes issus de la génération afin d'obtenir une séquence unique et un seul dictionnaire de fautes pour le circuit. Cette technique ne nécessite pas de faire une simulation de fautes sur l'ensemble du circuit. Nous parallélisons l'application des vecteurs issus d'unités testables non recouvrantes, ainsi que ceux issus d'unités testables peu recouvrantes. Enfin, nous complétons la séquence par des vecteurs dédiés aux structures de test elles-mêmes.

Dans le premier chapitre de ce document, nous définissons les types de circuit auxquels s'appliquent notre méthode, et nous définissons les objectifs de notre travail. Le second chapitre donne un aperçu des méthodes de test utilisées pour des circuits intégrés complexes conçus récemment, puis présente plusieurs outils universitaires proposant de partitionner automatiquement un circuit en vue de son test.

Notre méthode de test des circuits intégrés, basée sur un découpage structurel peu recouvrant, est présentée dans le chapitre 3. Les principales difficultés liées à ce découpage sont mises en valeur, et nous montrons comment représenter un circuit sous la forme d'un réseau booléen orienté.

Le chapitre 4 est consacré à la reconnaissance automatique de signaux observables et contrôlables, basée sur des techniques de simulations logiques et symboliques développées dans le cadre de la vérification de règles de testabilité. Ces techniques nous permettent de nous affranchir de la connaissance fine des structures de test présentes dans le circuit.

Le chapitre 5 décrit les algorithmes permettant de découper efficacement un circuit en unités testables peu recouvrantes en vue de la génération des vecteurs de test. Nous effectuons ce découpage en deux temps : tout d'abord le circuit est virtuellement découpé en unités testables élémentaires ne comportant qu'une seule sortie. Puis un mécanisme permet de regrouper ces unités testables de façon à minimiser le caractère recouvrant du découpage.

Dans le chapitre 6, nous nous intéressons à l'interaction de ce découpage avec la présence de blocs auto-testés et de macro-cellules. Enfin, nous présentons au cours du chapitre 7 les mécanismes permettant d'assembler les vecteurs de test et les dictionnaires de fautes, et nous décrivons l'intégration d'une architecture *Boundary-Scan* avec notre méthode.

Dans le dernier chapitre, nous présentons le prototype logiciel s'appuyant sur la chaine de CAO ALLIANCE. Nous confrontons ainsi notre méthode à PCI-DDC, un circuit de 200.000 transistors, intégrant des macro-cellules, conçu dans notre laboratoire.

Chapitre 1

Problématique

Ce travail a pour cadre la génération des tests pour les circuits intégrés digitaux complexes. Le test intervient après la fabrication des circuits. Il a pour but d'éliminer les circuits possédant un ou plusieurs défauts introduits par le processus de fabrication. Ce test a pour forme un ensemble de stimuli à appliquer soit sur la tranche de silicium, soit sur le circuit encapsulé. Le *taux de couverture* du test reflète la capacité des stimuli à détecter l'ensemble des défauts potentiels issus de la fabrication.

Aujourd'hui, la conception d'un circuit intégré se fait de plus en plus rapidement malgré des tailles et des performances toujours croissantes. Cela est possible grâce à l'existence d'outils de CAO puissants, mais aussi par la réutilisation de parties développées au cours de projets antérieurs. Pour que la génération des stimuli de test se fasse elle aussi de manière rapide, les concepteurs ont adopté des règles de conception *en vue du test*. Elles consistent à prendre en compte le test dès le début de la conception et à éviter ainsi de faire des choix rendant le test difficile. Ces règles préconisent, en outre, de rajouter au circuit des structures de test qui ne serviront qu'à le tester.

Malgré ces précautions, la mise au point de ces stimuli reste une étape coûteuse dans le cycle de conception du fait de la taille et de la diversité des circuits actuels : cette phase représente encore près de la moitié de l'effort total de conception d'un circuit.

1.1 Les problèmes posés par le test aujourd'hui

Nous avons rapporté dans le tableau 1.1 pour quelques grands projets, outre la complexité en nombre de transistors, la présence ou non de trois structures de test les plus répandues :

Nom	Date	Taille	Scan-Path	Auto-Test	Boundary-Scan
68040	1989	1,2 MT	Oui	Non	Oui
80486	1990	1,2 MT	Non	Oui	Non
IBM 370	1990	1,0 MT	Oui	Oui	Oui
HP PA7100LC	1993	0,9 MT	Oui	Non	Oui
SuperSPARC	1993	3,1 MT	Oui	Non	Oui
PowerPC 603	1994	1,6 MT	Oui	Oui	Oui
MicroSPARC	1994	2,3 MT	Oui	Non	Oui
68060	1994	2,5 MT	Oui	Oui	Oui
Alpha 21164	1994	9,3 MT	Oui	Oui	Oui
TMS320C80	1995	3,9 MT	Oui	Non	Oui
UltraSPARC	1995	5,2 MT	Oui	Non	Oui

[*Sources : ITC, CICC, ICCAD, EDT&C, IBM R&D*]

TAB. 1.1 –: *Quelques exemples de circuits actuels*

1. le *scan-path* : cette technique de test consiste à transformer les points mémoires d'un circuit pourqu'en mode « test » ils forment un registre à décalage. Ainsi chaque point mémoire est directement contrôlable et observable depuis l'interface en décalant ce registre. Un *scan-path* peut être complet ou partiel suivant que tous les points mémoires sont dans le registre à décalage ou seulement une partie. Enfin le chemin de *scan-path* peut être divisé en plusieurs parties pour en réduire la longueur : nous parlons alors de *scan-path* multiple.

2. le *test intégré* : son principe est d'inclure dans le circuit lui-même des structures spécialisées pour générer les vecteurs qui vont le tester d'une part, et vérifier les réponses attendues d'autre part. Ainsi, le circuit est capable de se tester de façon autonome. Le test intégré, aussi appelé *auto-test*, est très utilisé pour le test des structures régulières comme les mémoires.

3. le *Boundary-Scan* : c'est une norme récente [IEE90] qui vise à simplifier le test des circuits sur une carte, et à procurer une interface normalisée pour mettre en œuvre les tests du circuit. Cette norme définit un certain nombre d'instructions de test normalisées, certaines obligatoires, d'autres optionnelles, et permet de rajouter un nombre illimité d'instructions de test spécifiques.

A partir des informations contenues dans le tableau 1.1, qui sont issues de récentes publications, il est possible de dresser le portrait type d'un circuit intégré digital complexe actuel :

– une taille de quelques millions de transistors,

- un *scan-path*,
- des mémoires intégrées au circuit et équipées d'un mécanisme de test intégré,
- et une implémentation de la norme *Boundary-Scan*.

Des problèmes spécifiques apparaissent avec ce type de circuits et limitent la génération automatique des stimuli : ils sont dus à la complexité du circuit, à la diversité des structures de test utilisées et à la réutilisation de composants existants.

1.1.1 La complexité du circuit

La complexité du circuit influe principalement sur deux facteurs : la capacité à générer les vecteurs de test pour le circuit et le temps et les ressources exigés par la simulation de faute.

La génération automatique de vecteurs de test

Tant que la complexité des circuits n'excède pas quelques dizaines de milliers de transistors, des outils de génération automatique de vecteurs de test sont capables de fournir automatiquement des stimuli atteignant un taux de couverture satisfaisant. Cela est particulièrement vrai pour les circuits combinatoires, ou rendus combinatoires par l'utilisation d'un *scan-path* complet.

Aujourd'hui les circuits comportent plusieurs millions de transistors; cette taille dépasse les capacités de traitement des outils de génération automatique car la complexité des circuits augmente plus vite que la puissance des stations de travail. De plus, il n'est pas toujours possible d'équiper ce type de circuit d'un *scan-path* complet, souvent pour des raisons de performances. Le *niveau de séquentialité* d'un circuit, c'est à dire le nombre minimum de cycles requis pour observer ou contrôler n'importe quel registre non *scannable* d'un circuit séquentiel, complique beaucoup la tâche des outils de génération automatique de vecteurs de test [Bot86] [MDNSV88].

Deux types de solutions sont généralement utilisées pour pallier aux limitations de ces outils : la première approche consiste à remplacer la génération de vecteurs de test déterministes par une approche aléatoire ou à utiliser une approche fonctionnelle, en gardant une vision globale du circuit. Malheureusement avec les circuits complexes cette technique n'apporte pas les garanties suffisantes en terme de longueur de test et de taux de couverture.

L'autre méthode consiste à générer des stimuli sur des parties du circuit plus petites, puis à les assembler pour obtenir la séquence finale. En utilisant cette technique du « diviser pour régner », il est possible d'utiliser des outils de génération aux capacités plus limitées, voire de distribuer les tâches sur un réseau de machines. La difficulté consiste alors à définir les frontières de ce découpage du circuit. Elles peuvent être définies de façon fonctionnelle, suivant ainsi la hiérarchie inhérente à la conception du circuit, ou structurelle, guidées par des critères de testabilité. Nous nous situons clairement dans le cadre de cette seconde approche.

La simulation de fautes

Une fois les stimuli obtenus, il faut être capable d'en mesurer l'efficacité. Cela se fait grâce à l'utilisation d'un simulateur de fautes qui va associer aux stimuli un taux de couverture ainsi qu'un dictionnaire de fautes pour un modèle de fautes donné. Le *dictionnaire de fautes* précise pour chaque faute si elle est ou non détectée par les stimuli. Il est utilisé pour guider la génération de nouveaux stimuli en vue d'augmenter le taux de couverture. La simulation de faute est coûteuse : elle doit en effet se faire avec une description « bas niveau », en général un réseau de portes.

Il est pourtant indispensable d'obtenir cette mesure de la qualité du test sous peine de laisser passer des circuits ayant un défaut de fabrication lors du test.

Enfin dans le cadre d'une génération sur des sous-parties du circuit, nous nous trouvons confronté au problème d'obtenir un dictionnaire de fautes global à partir des dictionnaires partiels associés à chacune de ces sous-parties.

1.1.2 La diversité des structures de test

Les techniques de conception en vue du test sont nées avec l'idée qu'il fallait inclure dans le circuit lui-même des mécanismes qui facilitent son test.

Les différentes parties des circuits actuels sont faites avec des techniques de conception différentes : synthèse logique multiniveaux s'appuyant sur une bibliothèque de cellules pré-caractérisées pour la logique de contrôle, générateurs paramétrables d'opérateurs optimisés pour les chemins de données, circuiterie optimisée pour les mémoires, etc ...

Or, un banc de registres multi-accès, un multiplieur, une RAM, ou encore de la logique de contrôle, ne sont pas testés de la même façon. Il existe un grand nombre de techniques de test adaptées à chaque type d'architecture. Les quatre techniques les plus utilisées sont :

- le test fonctionnel, où les vecteurs développés pour mettre au point le circuit sont repris dans le cadre du test,
- le test structurel déterministe, où des générateurs automatiques de vecteurs de test sont utilisés, et dont le travail est simplifié par l'utilisation de chemins de *scan-path*,
- le test intégré déterministe, pour les mémoires où les zones difficilement testables qui consiste à stocker dans le circuit lui-même des vecteurs précalculés,
- et le test intégré pseudo-aléatoire pour les chemins de données ou la logique de contrôle. Un mécanisme de génération aléatoire de vecteurs est intégré ici au circuit, en espérant atteindre un taux de couverture suffisant.

Chaque technique de test, à l'exclusion du test fonctionnel, s'appuie sur des structures de test. Lors du découpage en vue de la génération automatique des stimuli, il faut être capable de reconnaître ces structures pour les exploiter correctement. Entre autre il faut gérer la mise en œuvre de tous ces tests au niveau du circuit en favorisant leur parallélisation. Enfin, il faut être en mesure d'exploiter les structures de test au niveau du circuit comme le *Boundary-Scan*, sans les considérer comme de la logique séquentielle indifférenciée.

1.1.3 La réutilisation de composants

Une des façon d'accélérer la conception de circuits aujourd'hui est de réutiliser pour un projet des blocs ou des macro-blocs conçus pour des projets précédents. Cela peut prendre trois formes : la migration technologique consiste à adapter un bloc prévu pour une technologie antérieure à la technologie courante; la deuxième est l'utilisation de générateurs paramétrables [GP94] qui permettent d'obtenir des architectures régulières optimisées; enfin il est possible d'acheter un bloc complet chez un vendeur de silicium. Par exemple pour la conception d'un circuit de codage de la voix, Thomson-CSF a acheté à SGS-Thomson un cœur complet de DSP avec ses vecteurs de test [Pic96].

Dans ces trois cas, les concepteurs sont face à un nouveau problème : ils n'ont pas la maîtrise de l'architecture interne du bloc et doivent se contenter de la technique de test prévue à la conception, toute modification étant impossible. C'est le contrepied de la conception en vue du test puisqu'ici c'est la stratégie de test qui doit s'adapter.

1.2 Notre approche

Pour résoudre ces difficultés, plusieurs solutions ont été proposées soit par les industriels fabricants de circuits soit par des études universitaires.

Nous présentons ici une méthode visant à favoriser la génération automatique de vecteurs de test sur des circuits digitaux complexes de grande taille. En nous inspirant des méthodes utilisées pour les microprocesseurs, nous essayons de l'étendre aux circuits obtenus par l'assemblage de blocs issus de bibliothèques et de fabricants différents. Cette méthode s'appuie sur un découpage du circuit en unités testables autour de ses structures de test. Un de nos buts est de pouvoir distribuer la génération automatique des vecteurs de test sur un réseau de machines.

La méthode générale est la suivante :

– un découpage automatique du circuit en unités testables permettant aux générateurs automatiques de vecteurs de test classiques de fonctionner, et permettant une génération distribuée sur un ensemble de machines,

– une reconnaissance automatique des structures de test classiques (*scan-path*, test intégré, *Boundary-Scan*) pour guider le découpage et faciliter le travail des outils de génération automatique de vecteurs de test,

– et une obtention du taux de couverture et du dictionnaire de fautes sans passer par une simulation de fautes sur le circuit complet, par assemblage optimisé des séquences générées sur chaque unité testable.

Nous voulons prouver que la génération automatique de stimuli sur des circuits de grande taille est possible avec des outils classiques. Nous prendrons comme exemple le test d'un circuit actuellement développé au laboratoire.

Nous allons au cours de cette étude répondre aux questions suivantes :

1. Comment identifier automatiquement les points observables et les points contrôlables du circuit ainsi que les protocoles d'accès à ces points ?

2. Comment découper un circuit en unités testables peu recouvrantes et indépendantes ?

3. Comment obtenir un jeu de vecteurs et un dictionnaire de fautes à partir d'un ensemble de séquences et d'un ensemble de dictionnaires, sans re-simuler le circuit complet ?

Chapitre 2

État de l'art

Dans la première partie de ce chapitre nous faisons un tour d'horizon des différentes méthodes utilisées dans l'industrie pour le test des circuits complexes. Pour cela, nous avons recensé pour quelques grands projets :

1. les structures de test implémentées,
2. la méthode de génération des vecteurs,
3. la méthode d'obtention du taux de couverture et du dictionnaire de fautes.

Les techniques pour faire face à la complexité passent très souvent par le recours au découpage du circuit. C'est pourquoi dans une deuxième partie, nous nous intéressons plus particulièrement aux travaux existants visant à partitionner un circuit en vue du test.

2.1 La génération des tests dans l'industrie

Dans le domaine du test, l'initiative vient souvent de l'industrie comme le montre le *scan-path* inventé par NEC et IBM [FKY89], ou l'effort de standardisation du *Boundary-Scan* [IEE90] mené par Philips avec le *JTAG* (*Join Test Action Group*). Nous allons étudier ici les techniques de test réellement mises en œuvre par les industriels confrontés au problème du test des circuits de grande complexité.

2.1.1 IBM : le 370

IBM reste une référence dans le monde du test. C'est en effet en 1964, pour le test de l'IBM System/360 que Carter et al. introduisent le concept de *scan in/scan out* [WP83] qui préfigure le *scan-path*. En 1977, Eichelberger et Williams introduisent le *Level Sensitive Scan Design* [EW77], un *scan-path* pour horlogerie bi-phase, qui reste une des bases du test aujourd'hui.

Pour la conception en vue du test du processeur central de l'IBM system/370 [Sta90], un circuit CMOS d'un peu plus d'un million de transistors, le souci majeur des concepteurs a été de pouvoir réutiliser les tests développés pour le circuit seul lorsque celui-ci est au sein d'une carte. C'est pourquoi l'architecture *Boundary-Scan* et une génération pseudo-aléatoire interne au circuit ont été adoptées.

Le processeur intègre trois familles de structures de test :

1. tous les points mémoires à l'exception des RAM sont dans un *scan-path*,
2. une correspondance directe entre les entrées des RAM et les entrées primaires du circuit ou des éléments du *scan-path* d'une part, et entre les sorties des RAM et les sorties primaires ou des éléments du *scan-path* d'autre part, est assurée lors de la mise en mode test. Cela permet un accès direct aux entrées et aux sorties de la RAM.
3. Le circuit est équipé d'une architecture *Boundary-Scan* standard possédant l'instruction optionnelle INTEST. Cette instruction permet d'activer un *scan-path* de périphérie qui prend alors le contrôle des entrées du circuit et observe ses sorties.

Les concepteurs ont décidé d'adopter une approche pseudo-aléatoire pour la génération des vecteurs de test. Ils n'ont donc pas eu besoin d'un outil de génération automatique. Cette approche est décrite dans la figure 2.1.

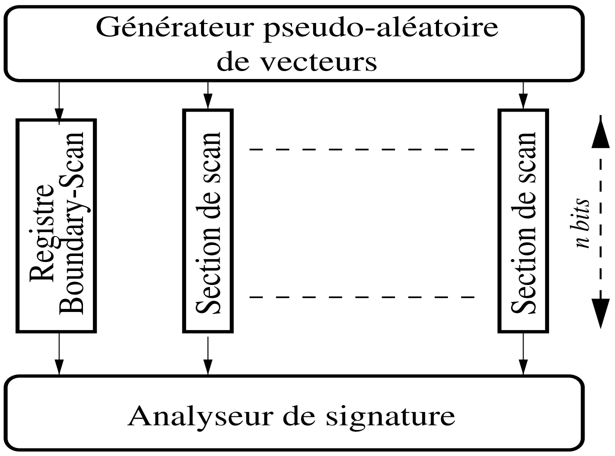


FIG. 2.1 –: Approche d'auto-test pseudo-aléatoire implémentée par IBM

En mode auto-test, le *scan-path* est divisé en plusieurs sections de longueurs identiques. Un registre à décalage monté en polynôme générateur (LFSR) est branché sur les entrées de chaque section, ainsi que sur le registre *Boundary-Scan*. De même, les sorties des sections et la sortie du registre *Boundary-Scan* sont branchées sur un analyseur de signature. Ainsi des valeurs pseudo-aléatoires sont générées dans les sections et sur l'interface du cœur, en un nombre de cycles correspondant à la longueur des sections de *scan-path*. Après un cycle fonctionnel le résultat du vecteur est compressé sous la forme d'une signature pendant que le vecteur pseudo-aléatoire suivant est introduit dans les sections de *scan-path* et le registre *Boundary-Scan*. A la fin du test, la signature finale est ressortie pour être comparée à celle calculée par la simulation. Grâce à cette technique d'auto-test associé à l'utilisation du *Boundary-Scan* les vecteurs servant à le tester sont générés au sein même du circuit, et le même test est utilisable sans transformation lorsque le circuit est placé au sein d'une carte.

Dans cette approche il faut s'assurer qu'un nombre suffisant de vecteurs est produit pour atteindre un taux de couverture acceptable grâce à une simulation de fautes. Elle est ici effectuée de façon globale avec tout le circuit. Le modèle de faute est celui des collages.

Cette technique de test à base de génération aléatoire permet de se passer d'un outil de génération de vecteurs de test, mais de l'aveu même d'IBM, quelques fautes peuvent ne pas être couvertes. Dans ce cas il faut avoir recours à quelques vecteurs déterministes pour compléter la séquence de test.

2.1.2 Motorola : Le 68040 et le 68060

Le 68040 de Motorola [SMG90] est un microprocesseur 32 bits d'une complexité de 1,2 millions de transistors. Il possède entre autres une unité de calcul entier, une unité de calcul flottant, et des mémoires caches de 4 Ko sur les adresses et les données.

D'un point de vue architectural, le 68040 est un mélange de chemin de données, de ROM/PLA, de RAM, de logique irrégulière et de machines à états.

Aucun registre du chemin de données n'est dans le *scan-path*. Celui-ci est testé à l'aide de vecteurs fonctionnels écrits à l'aide du langage assembleur 68040.

Tous les registres de la logique de contrôle sont dans le *scan-path* qui est scindé en plusieurs chaînes. De plus, tous les signaux qui contrôlent directement le chemin de données sont échantillonnés dans des registres appartenant au *scan-path* en mode test. Les vecteurs de test sont obtenus à l'aide d'un générateur automatique.

Les mémoires et les PLA présents dans le circuit sont entièrement encapsulés dans des *scan-path* de périphérie et des vecteurs dédiés sont écrits à la main en respectant un algorithme classique de test de mémoire.

Enfin il y a un *Boundary-Scan*, mais il est minimal et n'est utilisé que dans le cadre du test sur carte. Il ne permet pas, entre autres, de piloter le test des mémoires ou le mode *scan-path*.

Nous pouvons donc noter que Motorola utilise une approche mixte de vecteurs déterministes générés et de vecteurs écrits à la main. Pour ce qui concerne l'évaluation du taux de couverture, seules les parties appartenant à la logique de contrôle bénéficient d'un dictionnaire de fautes qui est issu de la phase de génération automatique de vecteurs de test. De par l'utilisation d'algorithmes de test reconnus, le taux de couverture de fautes sur les mémoires et les PLA est considéré comme maximum. Enfin en ce qui concerne le chemin de donnée, la notion de qualité du test est laissée à la discrétion de ceux qui écrivent les séquences fonctionnelles. Cette « évaluation » n'est plus possible avec des architectures plus compliquées.

Dans la génération suivante de microprocesseurs, le 68060 [CPC94], une architecture de type Harvard de 2,5 millions de transistors, les techniques de test ont évolué. Nous avons relevé des différences notables avec les structures de test du 68040, à savoir :

- un *scan-path* présent partout, dans les parties de contrôle comme dans les chemins de données,
- des mémoires et des PLA présents, toujours encapsulés dans des *scan-path* périphériques,
- et un *Boundary-Scan*, servant aussi d'interface pour le *scan-path* et la logique de contrôle des auto-tests.

Le test des mémoires et des PLA est toujours effectué avec des vecteurs écrits à la main et appliqués à travers le *scan-path*, qui assurent un taux de couverture maximum par construction.

La nouveauté par rapport au test du 68040 est l'utilisation de cellules « boite noire ». Ce sont des cellules qui sont transparentes en mode normal, et deviennent des bascules de *scan-path* lors du test. Placées judicieusement en complément des registres fonctionnels déjà présents dans le *scan-path*, elle ont permis de découper le circuit en 6 blocs combinatoires indépendants pour simplifier et accélérer la génération automatique de vecteurs de test. En effet, en mode test, chaque bloc ainsi constitué possède un *scan-path* complet et son interface est entièrement accessible. Les frontières des 6 blocs ont été définies manuellement, en s'appuyant sur le découpage hiérarchique du circuit. Le taux de couverture global des vecteurs générés est simplement la moyenne des 6 taux obtenus lors de la génération.

Nous voyons donc que Motorola a abandonné l'utilisation de vecteurs fonctionnels au profit d'un découpage structurel du 68060, nécessitant une vraie réflexion pour fixer les frontières du découpage. Cependant les blocs indépendants restent trop grands et Motorola rapporte que la génération automatique de vecteurs de test sur le bloc contenant l'unité de calcul flottant a demandé plus de deux semaines de calcul, bien que cela semble être dû aux capacités limitées de leur outil de génération. Enfin, pour ce qui concerne le test des chemins critiques à fréquence nominale, cette technique ne permet pas de les tester efficacement lorsque ceux-ci passent à travers une frontière entre deux blocs indépendants lors du test.

2.1.3 IBM & Motorola : le PowerPC 603

En pratique le choix des méthodes de test dépend de deux facteurs : les habitudes des concepteurs et la disponibilité d'outils de test adaptés. Une illustration en est le test du processeur PowerPC 603 [VTMPR95], réalisé conjointement par les équipes d'IBM et de Motorola. Le PowerPC 603 est un microprocesseur superscalaire 64 bits de 1,6 millions de transistors. Le choix des structures de test est un compromis entre les habitudes et les outils d'IBM et de Motorola [HVTL94]:

- toute la logique à l'exception des mémoires est équipée en *scan-path* complet sur le modèle du *Level Sensitive Scan Design* d'IBM [EW77],
- les bancs de registres sont testés à l'aide de vecteurs fonctionnels écrits en assembleur PowerPC,
- les deux mémoires caches de 8 Ko ont un mécanisme d'auto-test ;
- et le *Boundary-Scan* est très avancé, avec 39 instructions propriétaires permettant le test et la mise au point du PowerPC.

C'est visiblement la disponibilité d'outils de génération automatique puissants qui a guidé le choix d'un *scan-path* complet sans découpage. Ceux-ci ont permis d'atteindre un taux de couverture supérieur à 95 % sur les parties combinatoires. Comme précédemment, les mémoires sont considérées comme testées à 100 % grâce à l'utilisation d'algorithmes connus. Par contre, il n'y a pas eu de simulation de fautes du circuit complet, ce qui empêche d'avoir un taux de couverture global. En effet la circuiterie *Boundary-Scan* elle-même, ainsi que quelques blocs de logique autour des mémoires, n'ont été testés qu'avec des vecteurs fonctionnels. Leur efficacité a été seulement évaluée, et non mesurée, car les concepteurs ne possédaient pas de simulateur de fautes capable de traiter tout le circuit.

La méthode utilisée pour tester le PowerPC paraît très efficace pour les parties logiques et pour les mémoires auto-testées, bien qu'elle nécessite de puissants outils de génération de vecteurs de test. Par contre elle demande encore une intervention manuelle pour le test des bancs de registres, et met en lumière la nécessité de traiter séparément la logique *Boundary-Scan*.

2.1.4 Hewlett-Packard : le PA7100LC

Le HP PA7100LC [JDA93] est un processeur RISC de 900.000 transistors possédant une unité de calcul entier, une unité de calcul flottant, un contrôleur d'entrées/sorties et une mémoire d'un kilo-octet. Les structures de test se composent d'un *scan-path* complet sauf pour le chemin de données, d'un auto-test pour la mémoire, et d'un *Boundary-Scan* minimum.

La majorité des vecteurs de test sont des vecteurs fonctionnels. Ceux-ci sont issus d'une simulation du circuit dans un environnement système. Les valeurs des entrées/sorties du circuit sont alors extraites et transformées en vecteurs de test.

Ces tests sont complétés par des séquences générées automatiquement pour les parties de contrôle. Pour simplifier cette génération, la logique de contrôle du *scan-path* et la logique du *Boundary-Scan* sont préalablement remplacées par des « boites noires » émulant le mode test. Cela permet de décharger l'outil de génération automatique de vecteurs. Enfin les vecteurs correspondant au contrôle de l'auto-test de la mémoire viennent compléter la séquence.

Il n'y a pas de simulation de fautes au niveau du circuit. En dehors des parties de contrôle, il n'y a donc pas de mesure objective du taux de couverture obtenu. C'est une question d'intime conviction pour ce qui concerne les 800.000 vecteurs fonctionnels utilisés pour le test du PA7100LC.

La technique de test utilisée chez Hewlett-Packard ne se différencie pas beaucoup de celle utilisée par Motorola pour le 68040 (voir la partie 2.1.2). L'utilisation de vecteurs fonctionnels est justifiée par la complexité du circuit (900.000 transistors) qui est le plus petit de notre étude, mais ne semble pas réaliste pour des circuits plus gros à cause de l'absence de calcul de taux de couverture. La seule nouveauté ici, est le remplacement de certains blocs complexes par des équivalents pour pallier les carences de l'outil de génération automatique de vecteurs. Il faut cependant être prudent avec ce genre de manipulations qui risquent de modifier le comportement du circuit.

2.1.5 Texas Instrument : le TMS320C80

Le TMS320C80 [HSBP95] est un circuit multimédia vidéo de 3,9 millions de transistors. Il est composé de quatre processeurs de traitement du signal programmables, d'un processeur RISC et de mémoires partagées. L'ensemble forme une machine parallèle MIMD. Grâce à un mode *Boundary-Scan* interne, chacun des processeurs est accessible indépendamment des autres en mode test. Comme quatre des processeurs sont identiques, il est possible d'utiliser la même séquence de test pour tester chacun d'entre eux. Il n'y a pas de problème de test à travers les frontières puisque les processeurs sont prévus pour communiquer à travers des bus en mode fonctionnel.

Chaque processeur étant équipé d'un *scan-path* complet, les vecteurs ont été générés par un outil automatique. Des modifications de la description fournie à l'outil de génération ont néanmoins été nécessaires :

- remplacement de la logique de contrôle du test par une « boite noire » émulant le mode test,
- élimination des latchs de synchronisation à écriture systématique en forçant leur entrée d'horloge à 0,
- remplacement des bus « tristate » par leur équivalent en portes logiques réalisant un décodage mutuellement exclusif,
- et transormation de tous les signaux bidirectionnels de l'interface en noeud interne.

Il n'y a pas de simulation de fautes puisque le générateur fournit un taux de couverture et un dictionnaire : le taux de couverture global est donc calculé par la moyenne pondérée des taux obtenus pour chacun des processeurs. Par contre les vecteurs sont ensuite convertis et assemblés pour être applicables depuis l'interface du circuit, et ils sont ensuite vérifiés fonctionnellement sur un émulateur matériel. Cela permet de s'assurer que le générateur n'a pas provoqué de court-circuit sur les vrais bus « tristate ».

L'utilisation d'un *Boundary-Scan* interne autour de chaque processeur dans ce circuit permet d'effectuer un partitionnement très efficace. Cependant cette technique n'est adaptée qu'à ce genre de circuit qui ressemble plus à une carte regroupant des composants indépendants. Il est par contre intéressant de voir que la phase de modification de la description pour la génération est validée par une simulation logique.

2.1.6 Sun et Texas-Instruments : le SuperSPARC

Le SuperSPARC, conçu par Sun mais fabriqué et testé par Texas-Instruments, est un processeur super-scalaire de 3,1 millions de transistors [PY93]. Un cache de données et un cache d'instructions, de 36 Ko chacun, représentent à eux seuls plus des deux tiers des transistors du circuit. Ces mémoires ne sont pas équipées d'auto-test, mais sont accessibles directement depuis l'interface, lors du test.

Le reste du circuit, à l'exception des contrôleurs de test et de la circuiterie *Boundary-Scan*, est entièrement équipé d'un *scan-path*. Toute la logique est partitionnée en quatre zones indépendantes d'environ 50.000 portes dont quelques milliers de bascules. Ce choix est justifié par les capacités de l'outil de génération automatique de vecteurs de test. Ce découpage est rendu possible par l'insertion, à la main, de cellules « boites noires » pour isoler en mode test les quatre zones les unes des autres.

Comme pour le test du TMS320C80, la description du circuit a été modifiée avant d'être envoyée à l'outil de génération automatique :

– les parties du circuit décrites au niveau transistors ont été remplacées par des équivalents en portes logiques,

– les bus « tristate » ont été remplacés par des équivalents logiques mutuellement exclusifs,

– toutes les bascules ont été remplacées par le modèle standard demandé par l'outil de génération automatique de vecteurs de test,

– et les latchs de synchronisation à écriture systématique ont été remplacés par des portes logique simples.

Des vecteurs fonctionnels ont été rajoutés pour le test de la logique de contrôle des *scan-path* et pour le *Boundary-Scan*. Nous pouvons noter que le *Boundary-Scan* est très complet et permet de gérer le *scan-path* ainsi qu'un mode d'auto-test activable à chaque mise en route du processeur.

Il n'y a pas eu de simulation de fautes globale du circuit en raison de sa complexité trop importante. Le taux de couverture est donc donné en trois chiffres : 99% pour les mémoires, 95% pour les parties ne possédant pas de *scan-path* et plus de 95% pour les parties en possédant.

L'insertion de multiplexeurs et de bascules pour créer des zones indépendantes demande un effort de conception très important. Les concepteurs du SuperSparc recommandent eux-même d'abandonner cette technique pour la conception des prochains circuits. Ce partitionnement manuel a visiblement été le point dur de la conception en vue du test.

2.1.7 La génération des tests chez Intel

Alors que la grande majorité des fabricants de circuits ont adopté le *scan-path*, Intel ne l'a toujours pas utilisé dans la conception de ses microprocesseurs 486 [GZ90] [GIKN89] et Pentium [NG96]. Par contre l'auto-test pour les mémoires est adopté depuis la conception du 386 [Gel87]. Enfin un *Boundary-Scan* est présent depuis la génération des Pentiums.

La génération des vecteurs de test se fait à l'aide de vecteurs fonctionnels dédiés à chaque partie du circuit. Pour connaître la qualité de ces tests et obtenir un dictionnaire de fautes, les concepteurs utilisent la simulation de fautes sur le circuit complet. Jusqu'au microprocesseur 486 cette simulation de fautes était effectuée grâce à un émulateur matériel, mais la complexité du Pentium Pro et le coût des émulateurs correspondants a provoqué le retour à une méthode de simulation logicielle [KAM+96].

Les ingénieurs d'INTEL ont mis au point un outil de simulation de fautes pour un circuit complet fonctionnant avec deux simulateurs : un simulateur logique haut niveau capable de stocker les valeurs logiques présentes sur les registres du circuit, et un simulateur de fautes au niveau portes.

Le principe est de remplacer la simulation d'un bloc du circuit par un appel au simulateur de fautes. Seules les fautes de ce bloc sont alors simulées, et le simulateur de haut niveau se contente de les propager par simulation logique jusqu'aux sorties du circuit. Les étapes d'un cycle de simulation sont :

1. le circuit sans faute est simulé. Pour un des blocs du circuit, la simulation logique haut-niveau est remplacée par un appel au simulateur de fautes. Celui-ci va simuler l'ensemble des fautes issues du bloc lui-même. A la fin du cycle, il transmet au simulateur logique l'ensemble des fautes observables sur les sorties du bloc avec leur effet.

2. l'etat des registres et des sorties du circuit est sauvegardé. Il représente l'état du circuit sans faute au vecteur courant.

3. pour chaque faute qui n'est pas encore détectée :

   (a) les registres du circuit sont remis dans l'état dans lequel ils étaient au vecteur précédent, avec la faute concernée active ;

   (b) toutes les sorties du bloc sont positionnées aux valeurs logiques induites par la faute ;

   (c) le cycle de simulation logique est effectué;

   (d) l'état des registres est sauvegardé s'il est différent de l'état du circuit sans faute ;

(e) la faute est marquée comme détectée si une différence est observée avec les sorties du circuit sans faute.
(f) les registres sont remis dans l'état du circuit sans faute.

4. la simulation logique du circuit sans faute est refaite pour remettre tous les signaux du circuit à leurs valeurs normales.
5. l'horloge est avancée d'un cycle.

La simulation de fautes peut elle-même être distribuée sur un réseau de machines. Grâce à cette technique le circuit complet est simulé en des temps raisonnables, mais seul le bloc qui est la cible des vecteurs est évalué du point de vue des fautes qu'il contient.

Les performances obtenues – de l'ordre de celles obtenues avec un émulateur matériel – sont justifiées par le fait que l'activité produite par une faute issue du bloc dans le circuit complet est limitée, et les simulations logiques utilisées pour détecter les fautes sont donc rapides. Surtout, au delà d'un certaine complexité du circuit complet, l'émulateur n'est plus capable d'effectuer la simulation de fautes dans des délais raisonnables.

Pour simplifier l'observation des fautes dans le circuit, un *scan-path* d'observation permet de sortir sur l'interface du circuit un nombre important de points internes. Mais ce *scan-path* n'est en aucun cas prévu pour contrôler ces points.

Toutes les mémoires ont un auto-test basé sur une génération pseudo-aléatoire et une compression de la signature. Ces vecteurs d'auto-test ont été simulés à l'aide du simulateur de fautes propriétaire décrit plus haut.

Enfin le *Boundary-Scan* permet, outre ses fonctionnalités normales, de contrôler les auto-test et le *scan-path* d'observation.

La méthode de test de chez INTEL découle de choix initiaux complètement différents des autres méthodes, à savoir l'absence de *scan-path* et l'utilisation systématique de vecteurs fonctionnels. Elle s'appuie sur une méthode de simulation de fautes unique dans le monde de l'industrie et qui ne semble pas facilement généralisable. Elle demande un connaissance fonctionnelle du circuit pour écrire les vecteurs à partir de l'étude des dictionnaires de fautes issus des simulation précédente. Enfin elle ne s'applique qu'aux circuits synchrones.

2.1.8 Le Macro Test de Philips

Philips a développé une méthode de test particulière qui prend le contre-pied de l'approche utilisée par INTEL. En effet le Macro-Test [BDSS90] prend le parti de mettre autour de chaque bloc fonctionnel du circuit un *scan-path* de périphérie qui l'isolera en mode de test. Cet effort est explicitement fait par le concepteur du bloc qui doit fournir, outre le bloc lui-même, les vecteurs permettant de le tester à l'aide de ce *scan-path* de périphérie. Cela lui permet de choisir le modèle de faute associé au bloc, ainsi que les outils de génération automatique de vecteurs de test et de simulation de fautes qui lui sont adaptés. Il peut aussi choisir d'écrire les vecteurs à la main ou bien d'implémenter un auto-test. Dans l'esprit du Macro-test, c'est le concepteur qui est responsable du test de son bloc.

Lorsque le bloc est intégré au circuit, l'accès à son interface est assuré par des chaînes de *scan-path* de quatre types :

Scan de données : ils permettent de transporter les données pendant le test.

Scan d'auto-test : ils permettent de générer des valeurs sur les blocs auto-testés et de compresser les résultats sous forme de signatures.

Scan d'instructions : ils permettent de charger une configuration dans les contrôleurs ou les registres et de les conserver pendant la durée du test.

Scan d'interface : ils permettent d'observer et de contrôler les bus d'interface entre les blocs.

Un « contrôleur de test de bloc » complète le dispositif sur le circuit. Grâce à lui chaque bloc peut être testé avec sa séquence dédiée depuis l'interface du circuit. Un outil d'assemblage de test permet de créer automatiquement la séquence finale regroupant le test de tous les blocs du circuit.

Bien que la phase de définition des blocs soit coûteuse, Philips pense qu'elle est compensée par la rapidité d'obtention du test final. En effet, les tests complets assurant 100% de taux de couverture sur un circuit de 150.000 transistors ont été obtenus en moins de deux heures, certes au prix d'une augmentation de 10% de la surface du circuit.

La méthode du Macro-Test est conceptuellement la plus simple et la plus efficace pour mettre en œuvre des tests en partitionnant le circuit. Ses deux principaux défauts sont l'augmentation significative de la surface des circuits, et l'ajout systématique de portes logiques, pénalisant les performances, sur les frontières des blocs. De plus, la réutilisation de blocs qui n'ont pas été conçus suivant l'approche Macro-test entraîne un effort de supplémentaire de conception pour les adapter à cette méthode.

2.1.9 Conclusion

Les industriels confrontés à la complexité croissante de leurs réalisations, essaient d'atteindre deux buts : des tests plus efficaces et des tests obtenus plus rapidement.

Pour mesurer l'efficacité des tests, il faut disposer d'un simulateur de fautes capable de traiter le circuit dans sa globalité. Or, mis à part INTEL, aucun industriel ne possède l'outil permettant une simulation de fautes globale et nécessaire pour évaluer le taux de couverture de vecteurs écrits à la main. C'est pourquoi nous observons l'abandon progressif des vecteurs fonctionnels au profit de vecteurs structurels et d'auto-test.

Pour obtenir des tests plus rapidement, il faut être capable de les générer automatiquement, mais les outils de génération automatique de vecteurs de test souffrent des mêmes limites que les simulateurs de fautes. C'est pourquoi les concepteurs essaient de découper leurs circuits en sous-parties plus petites pour les adapter aux outils. Ce découpage sert aussi à adapter la méthode de génération au bloc concerné (mémoire, logique de contrôle, ...).

Le problème du découpage est lié au choix des frontières qui le caractérise. Pour certains, ces frontières sont définies par les concepteurs qui utilisent donc des structures spéciales. Ces structures sont, cependant, souvent liées à une technologie, comme dans le cas du Macro-test. Cette perspective n'est pas possible dans le cas général de circuits formés par l'assemblage de blocs issus de bibliothèques et de fabricants différents, pour lesquels le concepteur ne maîtrise pas les structures de test.

Pour d'autres, les points d'accès naturels du circuit sont exploités pour guider le découpage, qui n'est pas alors optimal et forme des sous-parties encore trop complexes. En effet, la partie découpée doit être d'une complexité compatible avec les outils de génération automatique de vecteurs de test. Or, la volonté d'obtenir des sous-parties indépendantes est souvent incompatible avec ce principe.

Enfin, ce découpage doit être automatique pour faire face aux volumes importants des circuits actuels. Dans la partie suivante nous allons nous intéresser aux techniques visant à découper automatiquement un circuit en vue du test.

2.2 Découpage automatique des circuits en vue du test

Le découpage d'un circuit en vue du test consiste à obtenir des sous-parties plus petites sur lesquelles il est possible de générer des vecteurs. Ces sous-parties peuvent être disjointes ou non. Dans le premier cas nous parlerons de partitionnement du circuit. Dans le cas d'un découpage en sous-parties qui peuvent être recouvrantes, nous parlerons de découpage « recouvrant ».

Dans cette partie nous nous limiterons aux études visant à découper un circuit en vue d'un test déterministe. Il existe des techniques automatiques de découpage en vue d'un test pseudo-aléatoire [DM93]. Elles consistent à déterminer l'ensemble optimal des noeuds d'un circuit où vont être placées les sorties d'un générateur aléatoire de vecteurs. Elles ne sont en pratique pas utilisables dans des circuits de grande complexité.

2.2.1 Prospect

PROSPECT, « *a PROduction System for Partitioning and Evaluating Chip Testability* » [LCE86], développé dans les laboratoires du « *Hugues Aircraft Company* », est un système à base de règles qui permet de reconnaître des structures de test, de vérifier la conformité d'un circuit avec des règles de conception en vue du test, et enfin de partitionner le circuit pour la génération des vecteurs.

Pour faire face à l'évolution des techniques et des structures de test, l'auteur a choisi de reconnaître ces structures avec un système à base de règles, qu'il est facile d'actualiser. Ces règles sont écrites sous la forme :

```
SI <conditions sur le circuit> ALORS <actions sur le circuit>
```

Les `conditions` décrivent des configurations de type de cellule, de nom de connecteur et de connectique. Les `actions` peuvent être des modifications, des ajouts ou bien des suppressions dans la description du circuit. Il est ainsi possible de reconnaître diverses configurations et de les remplacer par un équivalent plus simple. Par exemple PROSPECT est capable de remplacer un multiplexeur suivi d'une bascule par une bascule de *scan-path* avec les étapes suivantes :

1. reconnaissance du multiplexeur suivi de la bascule à partir de la règle décrivant cette configuration,

2. ajout d'un élément générique « bascule de *scan-path* »,

3. modification du circuit pour connecter cette nouvelle bascule,

4. suppression de la bascule simple et du multiplexeur.

C'est donc une approche par reconnaissance de formes qui permet d'identifier les structures de test.

Lorsque tous les éléments reconnus ont été remplacés par des éléments génériques, PROSPECT vérifie l'interconnexion de ces éléments. Cela permet entre autre d'identifier les *scan-path*, mais aussi de vérifier quelques règles de conception en vue du test.

En préalable à la phase de partitionnement, PROSPECT transforme toutes les entrées données des éléments du *scan-path* en sorties primaires du circuit, et toutes les sorties des éléments du *scan-path* en entrées primaires. PROSPECT choisit alors une porte qui n'est pas encore incluse dans une sous-partie. Puis toutes les portes connectées aux entrées de cette porte et toutes les portes connectées à ses sorties sont rajoutées à cette nouvelle sous-partie. Celle-ci est ainsi construite de façon itérative en remontant en amont et en se propageant en aval à travers la description structurelle du circuit jusqu'à rencontrer des entrées ou des sorties primaires. La méthode est répétée jusqu'à ce que toute la logique qui n'a pas été identifiée comme structure de test soit intégrée à une sous-partie. De cette façon la description structurelle est découpée en sous-parties indépendantes puisqu'aucune porte n'appartient à deux sous-parties.

L'approche par règles et reconnaissance de formes permet l'adaptation de cette méthode à de nombreuses configurations, mais nécessite néanmoins la connaissance exacte de la topologie des éléments de test utilisés. Or avec l'utilisation de la synthèse logique ou la réutilisation de composants existants, les concepteurs n'ont pas toujours la maîtrise ni la connaissance précise de l'implémentation de leur *scan-path*. La reconnaissance automatique de structures plus complexes comme l'auto-test ou le *Boundary-Scan* paraît en outre difficile.

La construction de sous-parties de test indépendantes par PROSPECT permet une utilisation optimale des outils de génération de vecteurs de test puisqu'aucune porte ne sera testée plus d'une fois pour la même faute. Malheureusement dans les circuits actuels une sortie dépend souvent d'un très grand nombre d'entrées. Si les registres inclus dans le *scan-path* ne sont ni en assez grand nombre ni placés judicieusement, il y a de fortes chances d'obtenir avec cette méthode des sous-parties de très grande complexité qui vont dépasser les capacités des outils classiques de génération de vecteurs de test.

2.2.2 Crete

Pour mieux maîtriser la complexité des sous-parties découpées, et le test en général, il faut être capable de faire les bons choix au moment de la conception. C'est pourquoi beaucoup d'outils de test sont en fait des outils d'aide à la conception en vue du test.

CRETE [GSB91], de l'*University of Southern California*, est un système qui a pour but de redéfinir une hiérarchie pour un circuit, afin qu'elle soit favorable au test. Pour cela le circuit sans structure de test est mis à plat, pour former une description structurelle contenant deux types de portes : des portes combinatoires et des registres. Cette représentaion du circuit va pouvoir ensuite être modifiée pour y insérer des structures de test. Enfin CRETE va générer les vecteurs pour chaque bloc et composer la séquence finale.

Pour former des sous-parties disjointes, CRETE utilise des règles de regroupement. Il y a deux familles de règles. Les premières permettent de former des blocs de logique à partir des portes combinatoires. Les autres sont utilisées pour regrouper les registres.

Règles de formation des blocs de logique : Un bloc est tout d'abord formé d'une porte combinatoire. Deux blocs combinatoires sont regroupés si l'un des critères suivant est respecté :

1. l'un des blocs est directement connecté à l'autre,
2. un signal commun est en entrée des deux blocs,
3. ou un bloc possède en entrée la sortie Q d'un registre et l'autre possède en entrée la sortie $\overline{Q}$.

Ces règles sont appliquées de façon itérative jusqu'à ce qu'aucun regroupement ne soit possible. Ensuite, CRETE regroupe les registres à l'aide de la deuxième famille de règles.

Règles de regroupement des registres : Deux registres sont regroupés si tous les critères suivants sont respectés :

1. ils possédent le même bloc de logique en entrée et ils possèdent le même bloc de logique en sortie,
2. ils sont de même type (reset/set) et sont contrôlés par la même horloge,
3. et ils sont contrôlés par les mêmes signaux de contrôles s'il y en a.

CRETE identifie ensuite les sous-parties formées de blocs structurellement identiques. Il identifie aussi les structures combinatoires tel les multiplexeurs qui permettent de transporter de l'information sans l'altérer. Il les exploite lorsqu'ils sont à la frontière d'un bloc combinatoire et d'un registre afin de réduire la complexité des sous-parties. Enfin CRETE est capable d'isoler des structures spéciales comme les RAMs et les ROMs.

Cette représentation canonique en blocs de logique combinatoires indépendants et en registres va servir de base pour l'insertion de structures de test : transformation de registre en élément de *scan-path*, ajout d'auto-test, ...

Enfin, CRETE gère la génération automatique de vecteurs sur les sous-parties par des outils externes et assemble les vecteurs pour former la séquence finale.

CRETE est un véritable outil d'insertion de structures de test. A ce titre il n'est pas capable de reconnaître des structures de test déjà incluses dans le circuit, ce qui en fait un outil mal adapté dans le cadre de la réutilisation de composants. De plus, CRETE ne tient pas compte des capacités des outils de génération pour ses sous-parties qui peuvent être surdimensionnées.

2.2.3 Contest

CONTEST [GMDz90] [MGzD90] de l'université d'Hannovre, est un outil de génération automatique de vecteurs de test pour blocs de logique combinatoire qui utilise une technique de découpage purement recouvrant pour accélérer la génération.

La technique de découpage utilisée autorise la constitution de blocs recouvrants appelés « cônes ». Un cône est obtenu à partir d'une sortie primaire et englobe toutes les portes qui ont une influence sur cette sortie. Ainsi une porte peut appartenir à plusieurs cônes si la valeur de sa sortie influe sur plus d'une sortie primaire.

Pour générer les vecteurs, CONTEST commence par découper le circuit en autant de cônes qu'il a de sorties. Un dictionnaire de fautes global décrit pour chaque faute du circuit son état courant. Ensuite, pour chaque cône :

1. la description structurelle du cône est extraite,
2. un dictionnaire local est généré à partir du dictionnaire global en n'incluant que les fautes appartenant au cône,
3. des vecteurs sont générés sur le cône à partir du dictionnaire local : ils ont pour cible les fautes non encore détectées au niveau du circuit,
4. le dictionnaire global est mis à jour à partir du dictionnaire local.

Grâce à cette méthode de génération, les auteurs de CONTEST montrent que les fautes redondantes indétectables sont identifiées en moyenne deux fois plus vite qu'avec une méthode globale.

Cette technique de découpage en cônes recouvrants est intéressante car si toutes les sorties ne dépendent pas de toutes les entrées, elle permet d'obtenir des sous-parties plus petites que ne le font les techniques de partitionnement. Par contre elle n'est pas adaptée à une génération en parallèle des vecteurs. En effet la génération de vecteurs sur un cône a besoin du dictionnaire de fautes issu des vecteurs déjà générés. C'est indispensable pour ne pas générer plusieurs vecteurs pour la même faute si celle-ci est incluse dans plusieurs cônes. La génération ne peut donc se faire que séquentiellement, cône par cône. Enfin, CONTEST n'est pas adapté à un circuit complet contenant des structures d'auto-test et des structures *Boundary-Scan*.

2.2.4 Partitionnement fonctionnel

Toutes les techniques de découpage présentées précédemment se fondent sur une vue structurelle du circuit, où celui-ci est représenté par un réseau de portes logiques orientées. D. Crestani [Cre91] [CAG+92], du laboratoire d'informatique, de robotique et de micro-électronique de Montpellier, propose le découpage d'un circuit décrit sous la forme d'interconnexions de modules fonctionnels. Cette méthode est dédiée aux parties combinatoires d'un circuit séquentiel synchrone.

Ce découpage se base sur des mesures fonctionnelles de testabilité. Ici les frontières des sous-parties ne sont pas forcément les points observables et contrôlables du circuit, mais peuvent être n'importe quel signal entre deux modules fonctionnels.

Les boucles de reconvergence fonctionnelle du circuit sont identifiées. Elles servent à bâtir des « diamants ». Un diamant est constitué d'un noeud divergent et de l'ensemble des modules fonctionnels appartenant aux chemins de reconvergence du noeud.

Les mesures de testabilité fonctionnelles sont pré-calculées sur la description fonctionnelle. Elles sont de trois types :

- L'*observabilité* : elle reflète de la difficulté de faire transiter une valeur depuis le noeud concerné jusqu'à un point observable.
- La *contrôlabilité* : elle reflète de la difficulté de faire transiter une valeur depuis un point contrôlable jusqu'au noeud concerné.
- Le *coefficient de corrélation* : il reflète de la difficulté de dé-corréler une information se présentant en entrée d'une boucle de reconvergence jusqu'à une sortie primaire.

Avant de lancer le découpage automatique, l'utilisateur fixe trois contraintes :

1. la taille maximum de la partition, à fixer en fonction des capacités de l'outil de génération automatique de vecteurs de test,

2. le degré minimum de *contrôlabilité* que doit posséder une entrée de la partition,

3. le degré minimum d'*observabilité* que doit posséder une sortie de la partition.

Une partition est d'abord constituée à partir d'un diamant. Puis elle est étendue par le rajout de modules fonctionnels voisins pour que ses frontières respectent les contraintes de *contrôlabilité* et d'*observabilité*, et jusqu'au maximum autorisé par la contrainte de taille. Un module fonctionnel ne peut pas être scindé. Par contre, il peut appartenir à plusieurs sous-parties qui seront alors recouvrantes. Ce processus est répété pour chaque diamant ne faisant pas encore partie d'une partition, puis les modules isolés restants sont aussi étendus pour former les dernières sous-parties. En cas d'échec de la procédure de découpage il est demandé à l'utilisateur de redéfinir ses trois contraintes.

Pour chaque partition des vecteurs sont générés. Ensuite l'outil tente de propager chaque vecteur depuis les points contrôlables jusqu'aux entrées de la partition d'une part, et depuis les sorties de la partition jusqu'aux points observables d'autre part. Cette phase de propagation est guidée par les mesures de *coefficient de corrélation* et la connaissance des diamants de reconvergence. En cas d'échec un nouveau vecteur est demandé à l'outil de génération automatique. Il se peut donc qu'une faible partie des vecteurs destinés à tester une partition soit effectivement applicables à cette partition.

Cette méthode de découpage est très puissante puisqu'elle permet réellement de maîtriser la taille des sous-parties. Par contre, elle demande pour être applicable de disposer d'une description du circuit en modules fonctionnels, ce qui n'est pas toujours possible, surtout dans le cadre de la réutilisation de composants. Enfin, elle ne concerne que les parties combinatoires d'un circuit et n'est donc pas adaptée à des structures comme les mémoires et le *Boundary-Scan*.

2.3 Conclusion

Nous avons vu que le découpage d'un circuit en vue du test est une étape obligée pour faire face à la complexité croissante des circuits. Actuellement les industriels font un découpage manuel de leurs circuits. Des outils automatiques proposent des solutions visant à découper un circuit en sous-parties indépendantes ou recouvrantes, ou bien proposent l'insertion de structures de test en vue de ce découpage automatique.

Les outils proposant d'insérer des structures de test ne sont pas adaptés aux circuits hétérogènes de grande complexité où la testabilité est souvent décidée très tôt, et en fonction de la nature des blocs. De plus, lors de la réutilisation de composants déjà équipés en structures de test, il n'est pas possible d'en insérer d'autres à posteriori. Pour les circuits tels que nous les avons définis au chapitre 1 il est impératif de savoir reconnaître et exploiter des structures de test qui sont déjà présentes. Malheureusement les techniques de reconnaissance automatique de structures de test utilisant une librairie de composants ou un ensemble de règles ne sont adaptées à l'hétérogénéïté des circuits complexes qu'au prix de mise à jour fréquentes et d'utilisation de composants standards ou propriétaires.

Les outils proposant un découpage en sous-parties indépendantes comme PROSPECT ou CRETE ne permettent pas d'obtenir des complexités compatibles avec les outils classiques de génération automatique de vecteurs de test. A l'opposé, les techniques de découpage en cônes recouvrants comme dans CONTEST permettent une meilleure maîtrise de la complexité des sous-parties, mais ne sont pas adaptées à une génération distribuée des vecteurs sur un réseau de stations de travail. Le nombre de vecteurs générés risque en effet d'être très important, une porte appartenant à plusieurs sous-parties étant testée plusieurs fois. Enfin, les techniques de découpage basées sur une description du circuit en module fonctionnel ne sont pas applicables en l'absence de cette description.

Chapitre 3

Une méthode de test par découpage recouvrant

Nous présentons ici notre méthode de test des circuits intégrés de grande taille. Cette méthode est destinée à la génération des tests sur des circuits obtenus par l'assemblage de composants issus de bibliothèques et de fabricants différents. Nous avons vu au chapitre précédent qu'une approche visant à découper le circuit en sous-parties de tailles raisonnables et de composition adaptée aux outils classiques de génération automatique de vecteurs de test est une solution réaliste au problème du test. Nous avons donc choisi un découpage automatique du circuit en vue de la génération automatique des vecteurs de test.

Les points observables et contrôlables du circuit vont servir de frontières à notre découpage. Cela nous permet d'amener les vecteurs de test sur les entrées de la sous-partie découpée, et de récupérer les résultats sur ses sorties, depuis les entrées et les sorties primaires du circuit. Dans la suite de l'exposé, nous appelerons ces sous-parties des « unités testables ».

Nous nous plaçons dans le cadre d'un circuit possédant déjà des structures de test permettant de rendre observables et contrôlables des points internes. La connaissance de ces points est donc un préalable au découpage dans notre méthode de test. Nous proposons pour cela une identification automatique de ces points.

Nous avons choisi d'effectuer un découpage recouvrant, c'est à dire qu'une même porte peut appartenir à plusieurs unités testables. Ce choix facilite le découpage et permet de mieux maîtriser la taille des unités testables pour qu'elles restent adaptées aux outils de génération automatique de vecteurs de test. Par contre, puisqu'une porte peut appartenir à plusieurs unités testables, elle sera la cible de plusieurs vecteurs destinés à tester ses fautes. Plus le découpage sera recouvrant, plus la durée du test sera augmentée, sans gain du point de vue du taux de couverture obtenu. Pour atténuer ce défaut, il faut optimiser le découpage de façon à fabriquer des unités testables peu recouvrantes. Enfin, lors de l'assemblage des vecteurs de test correspondant à différentes unités testables, nous essayons de tester en parallèle deux unités testables recouvrantes de façon à réduire au maximum la longueur de la séquence finale.

3.1 Prérequis

Pour que notre méthode soit applicable, elle demande certains prérequis que nous allons détailler. Ces exigences concernent le niveau de description du circuit et sa testabilité.

3.1.1 Niveau de description du circuit

Notre méthode s'applique aux circuits intégrés numériques. Elle suppose la disponibilité d'une description structurelle d'un ensemble de portes logiques. Cette description se présente sous la forme d'un réseau orienté de portes, chaque porte possédant un comportement associé. L'ensemble des comportements forme ce que nous appellons par la suite la « bibliothèque » du circuit. Les comportements des composants appartenant à cette bibliothèque doivent être décrits au niveau booléen.

Il est important que le réseau soit à grain fin, chaque porte représentant l'équivalent de quelques transistors. En effet la signification du taux de couverture dépend de la densité de signaux d'interconnexions au sein de la description du circuit.

Lors de la réutilisation d'un composant préexistant ou bien lors de l'utilisation de générateurs paramétrables, le concepteur ne dispose pas forcément d'une description structurelle au niveau portes. Le plus souvent il possède seulement une description comportementale – pour la simulation et la synthèse – et quelquefois une description physique – pour le placement et le routage. Pour obtenir néammoins une description structurelle au niveau porte, deux approches sont possibles :

1. **A partir de la description physique**, l'interconnexion des transistors est extraite, puis un outil d'abstraction fonctionnelle [Lau94] reconstruit des portes élémentaires à partir de ces transistors pour former une vue structurelle fine. Cette approche est assez lourde à mettre en œuvre, mais permet d'obtenir une description reflétant bien la réalité physique du composant.

2. **A partir de la description comportementale**, quand celle-ci est un réseau booléen. Cette description est alors virtuellement remplacée par une

description structurelle au niveau porte, chaque expression du réseau booléen correspondant à une porte. Cette approche ne permet pas d'obtenir, après génération des vecteurs, un taux de couverture fiable puisque ce réseau de porte ne correspond pas forcément à la réalisation physique. Par contre, elle permet de mettre en œuvre notre méthode de test très tôt dans le cycle de conception dans le cas d'une méthodogie « Top-Down ».

3.1.2 Testabilité du circuit

Le circuit peut posséder des structures de test. Comme nous l'avons précisé dans notre chapitre de problématique, le recours de plus en plus fréquent à la réutilisation de composants ou à des générateurs paramétrables ne permet plus d'insérer des structures de test à posteriori. Celles-ci sont inhérentes au macrobloc. Notre méthode ne prévoit pas de rajouter des structures de test dans un circuit, mais exploite celles déjà présentes. En effet, ce sont les points observables et les points contrôlables présents dans le circuit qui vont servir de frontières au découpage du circuit.

Pour que notre méthode donne des résultats satisfaisants, il faut donc qu'un minimum de points observables et contrôlables soient présents dans le circuit. Pour un circuit complexe ne contenant aucune structure de test, l'application de notre méthode de découpage engendrera une unité testable par sortie primaire, chacune englobant la quasi-totalité du circuit, à cause des multiples rebouclages inhérents aux circuits possédant des registres.

3.1.3 Mise en œuvre de la testabilité

En règle générale, les détails de l'implémentation des structures de test ne sont pas connus. En effet celles-ci sont souvent synthétisées, ou sont incluses dans le bloc auquel elles s'appliquent. Il est en particulier difficile de dresser à priori la liste des points observables et contrôlables du circuit. C'est pourquoi nous avons choisi d'identifier de façon automatique les structures de test présentes dans le circuit.

À la différence des outils s'appuyant sur une reconnaissance de formes, les structures de test vont être identifiées à partir de leur comportement. Notre méthode suppose donc que les signaux permettant d'activer les structures de test depuis l'interface du circuit soient connus, ainsi que les modalités de leur utilisation. Nous appellerons par la suite ces informations « protocoles de mise en œuvre des structures de test ».

3.2 Présentation générale de la méthode

La figure 3.1 présente notre méthode. Comme nous l'avons vu au paragraphe précédent, elle s'applique à une description structurelle au niveau portes du circuit s'appuyant sur une bibliothèque de comportements. Les protocoles de mise en œuvre des structures de test doivent être connus. Plusieurs étapes sont nécessaires pour obtenir les vecteurs de test et un dictionnaire de fautes :

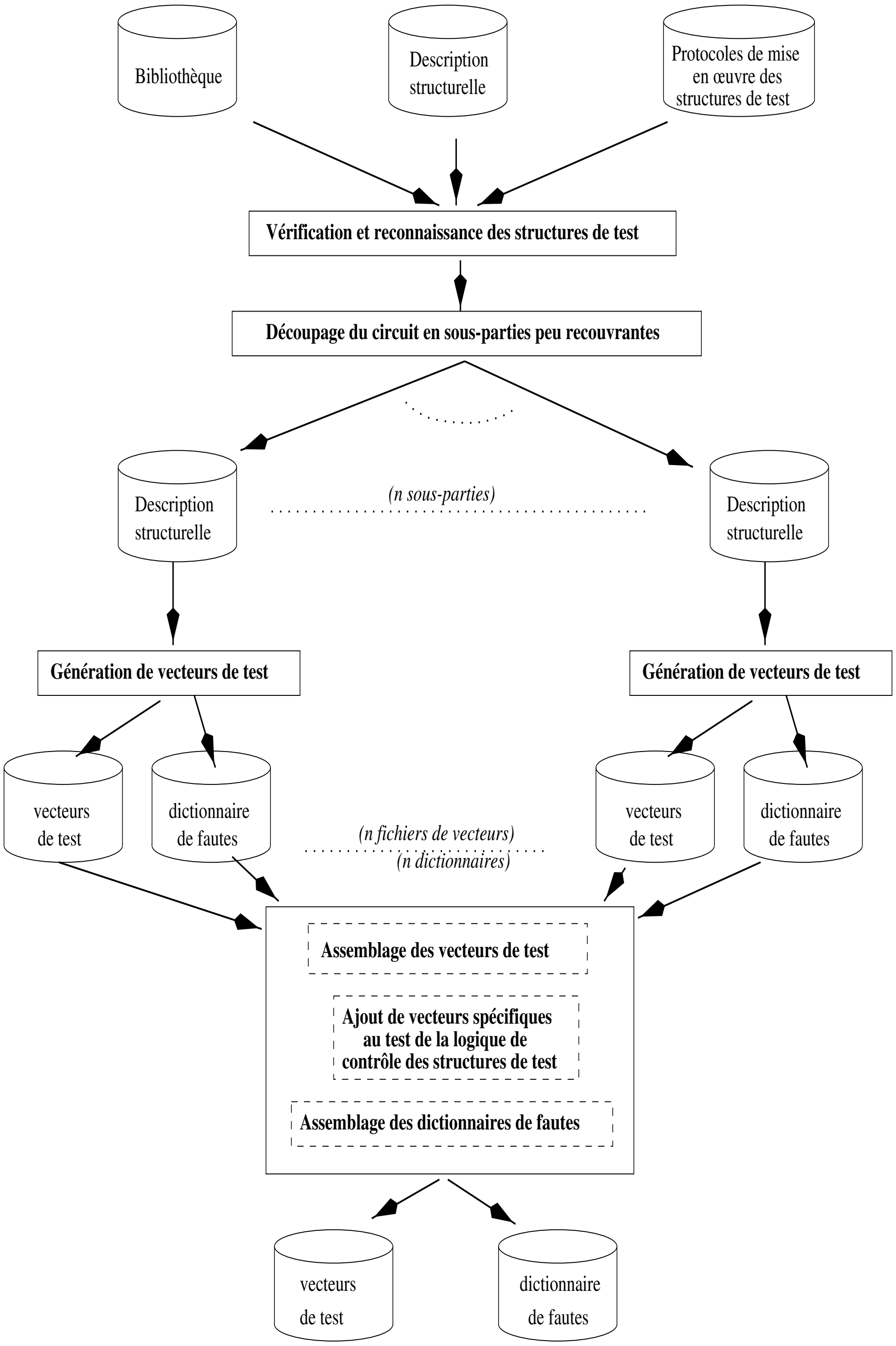


FIG. 3.1 –: Présentation générale de la méthode de découpage

1. Les structures de test présentes dans le circuit (*scan-path*, *Boundary-Scan*, auto-test) sont identifiées et vérifiées automatiquement. Cette reconnaissance se fait à partir des protocoles de mise en œuvre de ces structures, à l'aide de techniques de simulation logique et symbolique. Nous pouvons en déduire l'ensemble des points observables et contrôlables du circuit et

isoler les logiques de contrôle de ces structures de test pour un traitement particulier.

2. Le circuit, à l'exception des logiques de contrôle des structures de test précédemment identifiées, est découpé en unités testables de taille raisonnable aux vues des performances des outils de génération automatique. Ces unités testables ont toutes pour sorties des points observables du circuit, et pour entrées des points contrôlables du circuit. Cette technique de découpage a pour effet de fabriquer des parties recouvrantes, c'est à dire qu'une porte logique de la description structurelle initiale peux appartenir à plusieurs des unités testables découpées. Un algorithme permet de minimiser ce caractère recouvrant.

3. Des stimuli sont obtenus pour chaque unité testable à l'aide d'outils de génération automatique de vecteurs de test. Ces outils fournissent aussi un dictionnaire de fautes associé aux stimuli qui donne pour chaque faute son statut : active ou détectée. Si nous disposons d'un réseau de stations de travail, la génération des vecteurs sur les différentes unités testables peut se faire en parallèle.

4. Les différentes séquences de vecteurs, issues de chaque unité testable, sont transformées pour être applicables depuis l'interface du circuit. Ces séquences sont ensuite assemblées de façon optimale en privilégiant le test en parallèle d'unités testables indépendantes ou très peu recouvrantes. Enfin des vecteurs spécifiques aux structures de test sont rajoutés pour former la séquence finale de test du circuit. Parallèlement, les différents dictionnaires de fautes sont fusionnés pour que nous disposions d'un unique dictionnaire recensant le statut de chaque faute du circuit.

Cette méthode pose trois problèmes majeurs : la reconnaissance automatique des structures de test, le découpage en unités testables peu recouvrantes et l'assemblage des vecteurs et des dictionnaires. Avant de passer à la suite de l'exposé, nous allons présenter la façon dont nous modélisons un circuit. Les notions présentées ici serviront au cours des trois prochains chapitres, consacrés chacun à une des étapes de notre méthode.

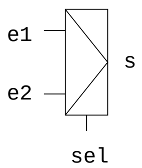
3.3 Description du circuit comme un réseau booléen

La description sur laquelle nous travaillons est un graphe orienté représentant le réseau booléen à plat du circuit. Il est issu de la représentation structurelle en portes du circuit et des comportements de chaque élément de la bibliothèque. Ces comportements doivent être décrits au niveau booléen. Nous nous appuyons ici sur le sous-ensemble VHDL utilisé par la chaine ALLIANCE. Il correspond au langage VHDL [IEE87] amputé des clauses procédurales et temporelles : seules subsistent les clauses structurelles permettant de décrire une structure hiérarchique, et les clauses de types assignations concurrentes.

Bien que le prototype logiciel que nous avons développé s'appuie sur le sous-ensemble de la chaine ALLIANCE, le modèle de représentation du circuit comme un réseau booléen est très général : il peut être reconstruit sans aucune difficulté de principe à partir de n'importe quelle description du circuit comme un réseau de portes logiques.

3.3.1 Représentation d'une porte logique

Chaque variable de la description comportementale d'une porte ou d'un module est représentée par un nœud du graphe. À chaque nœud est associé une expression booléenne. Un nœud reçoit un ensemble d'arcs orientés provenant des nœuds représentant les variables définies par l'expression booléenne qui lui est associée. L'exemple d'une porte dont la description comportementale possède des variables internes est présenté figure 3.2.




```
ENTITY porte IS
PORT(
  e1, e2, sel : in BIT,
  s : out BIT );
END porte;
ARCHITECTURE data_flow OF porte IS
SIGNAL notsel, choix1, choix2 : BIT ;
BEGIN
   choix1 <= e1 and sel;
   choix2 <= e2 and notsel;
    notsel <= not(sel);
    s <= choix1 or choix2;
END;
```

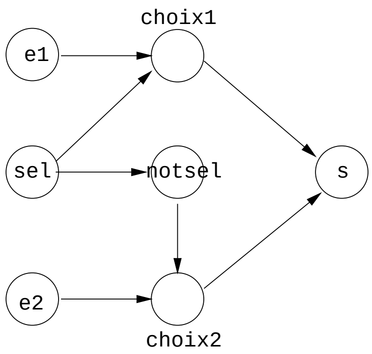


FIG. 3.2 –: Une porte, sa description comportementale et le réseau booléen associé

Au nœud `notsel` est associé l'expression `not(sel)`. Il reçoit donc un arc provenant du nœud `sel`. De plus, puisqu'il est utilisé dans l'expression rattachée au nœud `choix2`, il émet un arc vers ce nœud.

3.3.2 Représentation d'un registre ou d'un bus

Dans le cas des signaux gardés (registres ou bus), une ou plusieurs affectations conditionnelles sont associées à chaque signal. Chaque affectation conditionnelle comporte un signal de garde et un signal définissant la valeur du signal gardé. Pour chaque affectation conditionnelle un « *bi-nœud* », qui permet de matérialiser le couple valeur et condition de garde, est donc créé dans le graphe.

Un *bi-nœud* est l'association de deux nœuds virtuels : le premier représente l'expression booléenne qui correspond à la condition d'écriture. Le deuxième représente l'expression booléenne de la donnée à écrire.

Représentation d'un registre

Dans le sous-ensemble VHDL de la chaine ALLIANCE, un registre est toujours représenté par un signal gardé. Un *bi-nœud* est donc associé à chaque condition de garde. En simulation logique, lorsque le nœud virtuel représentant la condition d'écriture vaut '1', alors le point mémoire est écrit avec la valeur portée par le nœud virtuel représentant la donnée.

La figure 3.3 donne l'exemple d'un réseau booléen représentant un registre qui posséde deux horloges. CKT permet l'écriture de SIN dans le registre, et CKF permet l'écriture de DIN. Un signal SET permet de forcer le registre à '1'. Le nœud représentant le point mémoire possède donc trois bi-nœuds en entrée. L'expression booléenne associée au nœud virtuel représentant la condition d'écriture correspondant à SET est par exemple `(not SET)`, puisque la condition de garde est `(SET = '0')` .

```
ENTITY registre IS
PORT(
  set, din, sin, mode, ck : in BIT;
  s : out BIT );
END registre;
ARCHITECTURE data_flow OF registre IS
SIGNAL reg : REG_BIT REGISTER;
BEGIN
  sync : BLOCK (not ck and not ck'stable)
  BEGIN
       reg <= guarded (din and mode) or (sin and (not mode));
  END BLOCK sync;
  async : BLOCK (set = '0')
  BEGIN
       reg <= guarded '1';
  END BLOCK async;

  s <= reg;
END;
```

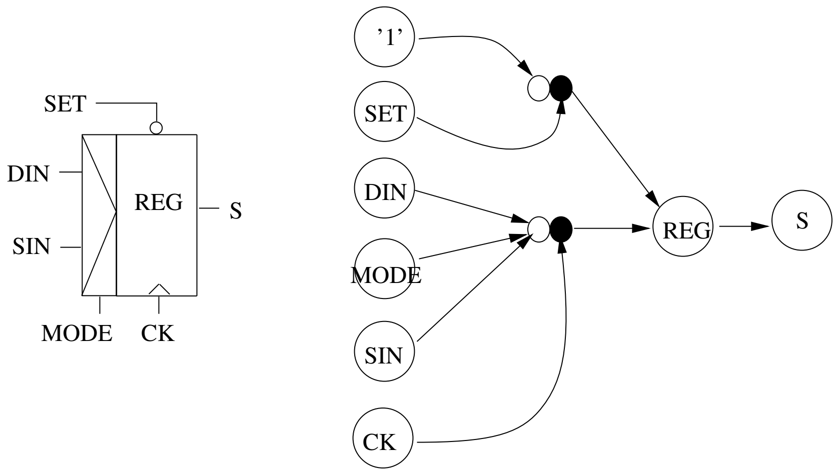


FIG. 3.3 –: Représentation d'un registre

3.3.3 Représentation d'un « tristate »

Les *bi-nœuds* sont aussi présents dans la représentation des portes « tristate ». Les arcs proviennent d'une part de la donnée à écrire, et d'autre part de la commande de mise en état de haute impédance. La figure 3.4 montre le réseau booléen d'une porte « tristate » possédant deux émetteurs.

```
ENTITY tristate IS
PORT(
  en1, d1, en2, d2 : in BIT;
  s : out MUX_BIT BUS );
END tristate;
ARCHITECTURE data_flow OF tristate IS
BEGIN
  drv1 : BLOCK (en1 = ’1’)
  BEGIN
       s <= guarded d1;
  END BLOCK drv1;

  drv2 : BLOCK (en2 = ’1’)
  BEGIN
       s <= guarded d2;
  END BLOCK drv2;
END;
```

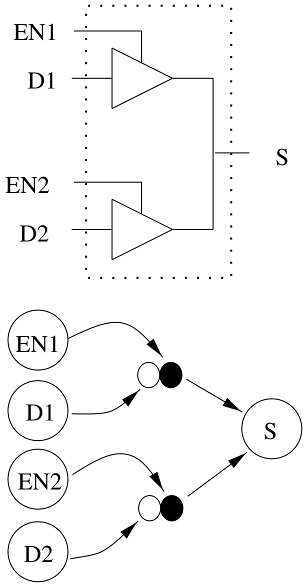


FIG. 3.4 –: Représentation d’un bus « tristate »

3.3.4 Représentation d’un réseau de portes

La description structurelle d’un circuit décrit l’interconnexion de portes à l’aide de signaux. Chaque signal est, lui aussi, représenté par un nœud du graphe. Un exemple de représentation d’un circuit combinatoire est donné dans la figure 3.5. Nous considérons ici que chaque porte n’est décrite que par une seule expression. Il existe ainsi dans ce réseau booléen autant de nœuds que de signaux dans la représentation structurelle.

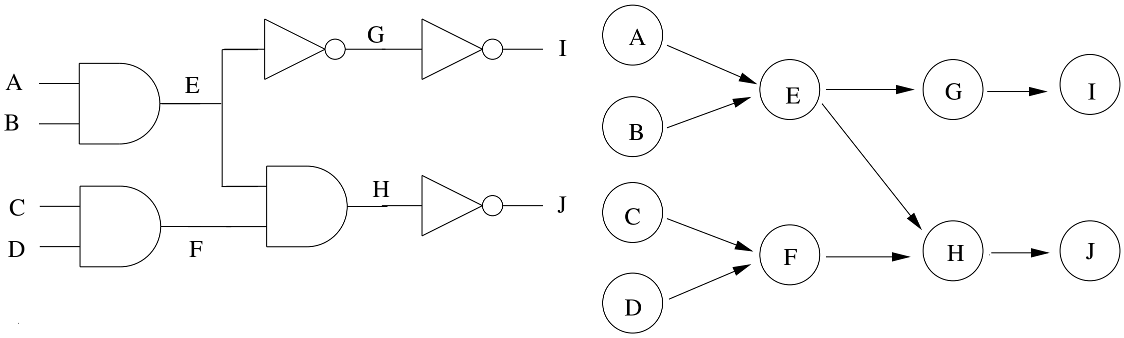


FIG. 3.5 –: Représentation d’un circuit combinatoire formé de plusieurs portes

La figure 3.6 donne l’exemple d’un circuit mélant deux portes logiques, un registre et une porte « tristate ». Les nœuds issus de la représentation structurelle du circuit et ceux issus de la description comportementale d’un des éléments de la bibliothèque sont indicernables.

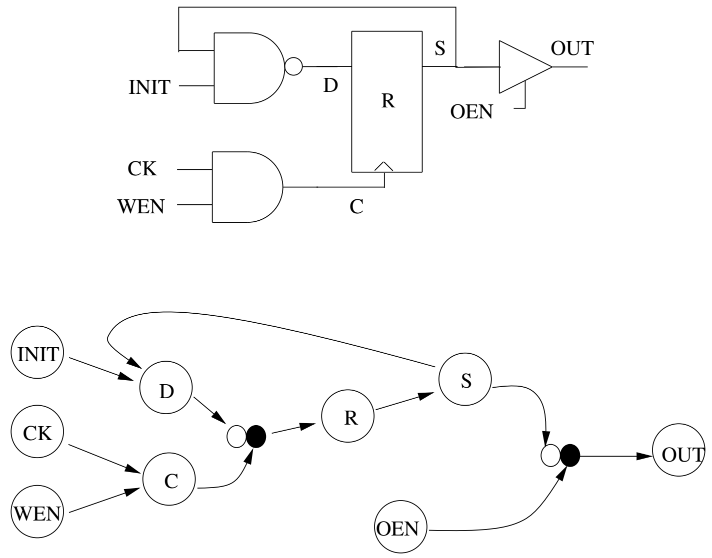


FIG. 3.6 –: Représentation en graphe orienté d'un circuit plus complexe

3.3.5 Marqueurs

Nous associons à chaque nœud du graphe un ensemble de marqueurs. Certains sont définis à la construction du graphe. Ainsi, pour différencier dans le graphe les nœuds correspondant réellement à des signaux de la description structurelle originelle de ceux représentant une variable interne à la description comportementale d'un élément de la bibliothèque, deux marqueurs sont utilisés : « structurel » et « comportemental ». Dans l'exemple de la figure 3.2 page 49, les nœuds `notsel`, `choix1` et `choix2` sont marqués « comportementaux », tandis que les autres sont marqués « structurels ».

D'autres marqueurs vont être positionnés sur le graphe lors de la phase de reconnaissance des structures de test, ainsi que lors du processus de découpage du circuit.

3.4 Conclusion

Nous avons présenté les grandes étapes de notre méthode de test par découpage structurel, et nous avons mis en lumière les trois principales difficultés, qui seront traitées dans les chapitres suivants. Nous avons détaillé le contexte dans lequel notre méthode s'applique, et nous avons proposé une manière de représenter un circuit sous forme de réseau booléen à plat s'appuyant sur les descriptions comportementales des composants de la bibliothèque.

Chapitre 4

Reconnaissance automatique de signaux observables et contrôlables

La plupart des outils de reconnaissance automatique de structures de test utilisent des techniques de reconnaissance de forme. L'ensemble des structures de test sont stockées dans une base de données soit sous forme de règles [Aba89], soit en décrivant explicitement les schémas à reconnaître [LCE86]. L'outil de reconnaissance parcourt la description du circuit en recherchant des éléments identiques à ceux présents dans sa base de données. Cette technique demande cependant que chaque structure de test utilisée dans le circuit soit caractérisée dans cette base de données. Ce n'est pas toujours possible avec la réutilisation de composants. La connaissance des structures de test incluses dans ce type de « macro-bloc » se limite alors aux moyens de les mettre en œuvre depuis son interface. Les détails de leur implémentation ne sont pas explicitement décrits. Il est alors difficile de mettre à jour la base de données d'un outil de reconnaissance automatique de structures de test fonctionnant par reconnaissance de formes.

Nous souhaitons donc identifier automatiquement les signaux observables et contrôlables d'un circuit à partir de données fonctionnelles externes que nous appelons « protocoles de mise en œuvre des structures de test ». Cette identification a pour but de guider le découpage du circuit en vue de la génération automatique de vecteurs de test.

Nous proposons que l'identification automatique des signaux observables et contrôlables s'appuie sur des techniques de simulation symbolique. Pour chaque structure de test, une première phase de simulation logique propage des valeurs logiques dans le circuit afin de le mettre dans le mode d'exploitation de cette structure, puis une simulation symbolique va propager des attributs depuis l'interface du circuit. Enfin, un examen des valeurs des attributs symboliques au sein du circuit permet l'identification des signaux observables et contrôlables de cette structure.

4.1 Techniques de simulation

Nous allons expliquer ici l'utilisation conjointe de la simulation logique et de la simulation symbolique pour l'identification des structures de test. Ces techniques ont été initialement développées par M. Hirech dans le cadre d'un outil de vérification de règles de testabilité [HFGR94].

4.1.1 La simplification du réseau booléen

Bien que dans le domaine de la vérification de réseaux booléens une simulation à deux états logiques ('0' et '1') soit suffisante [Vuo97], dans le domaine du test il est important de modéliser un état 'X'. L'état 'X' représente un état non déterminé du signal. Lors de l'initialisation d'un circuit en vue de l'exploitation d'un mode de test, très peu de connecteurs du circuit sont en général concernés, les autres étant laissés dans cet état indéterminé. Il faut alors faire la différence entre les signaux et les registres réellement initialisés, dont la valeur logique associée est '0' ou '1', et les autres, qui gardent une valeur logique 'X'.

La simulation logique permet de forcer des valeurs logiques à l'intérieur du circuit, pour le mettre dans un état particulier en rapport avec une des structures de test implémentées. Pour chacune d'entre elles les séquences à appliquer pour les mettre en œuvre sont supposées connues. L'application de ces séquences permet de simplifier la description du circuit en éliminant les chemins non fonctionnels pendant ce mode de test. Concrètement, cela revient à invalider temporairement certains arcs dans le réseau booléen à plat du circuit. Nous disons alors que le réseau du circuit est « sous contraintes » du mode de test.

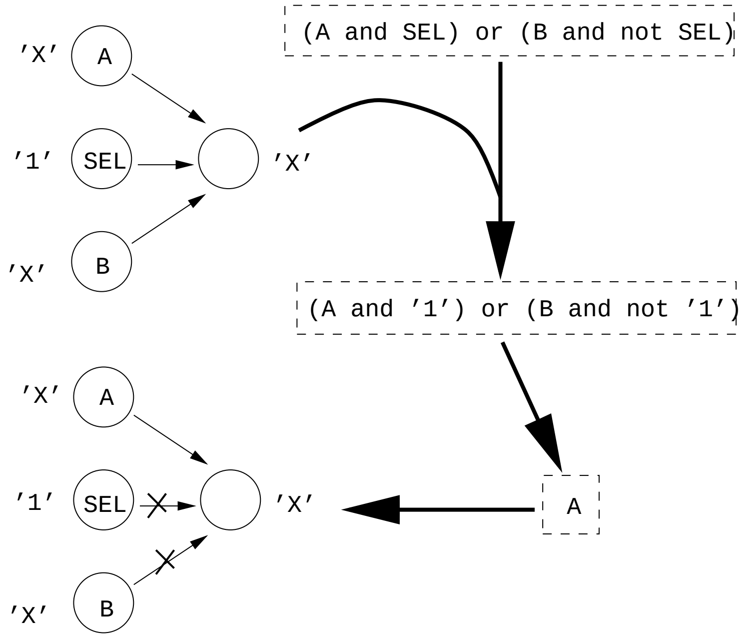


FIG. 4.1 –: Invalidation d'arcs sur un nœud à partir de l'expression booléenne associée

Pour mettre le réseau sous contraintes à l'issue de la phase de propagation des valeurs logiques, il faut procéder aux étapes suivantes pour chaque nœud du réseau booléen :

1. Les valeurs logiques présentes sur les variables d'entrées de l'expression booléenne sont collectées.
2. Toutes les variables de l'expression dont la valeur est différente de 'X' sont remplacées par leur valeur.
3. L'expression est simplifiée (grâce aux graphes de décision binaires).
4. Les arcs entrant qui correspondent aux variables et qui n'apparaissent plus dans le support de l'expression simplifiée sont invalidés.

Un exemple d'invalidation de nœuds est décrit dans la figure 4.1. L'arc émis par B est invalidé bien que le nœud correspondant ait une valeur logique non définie.

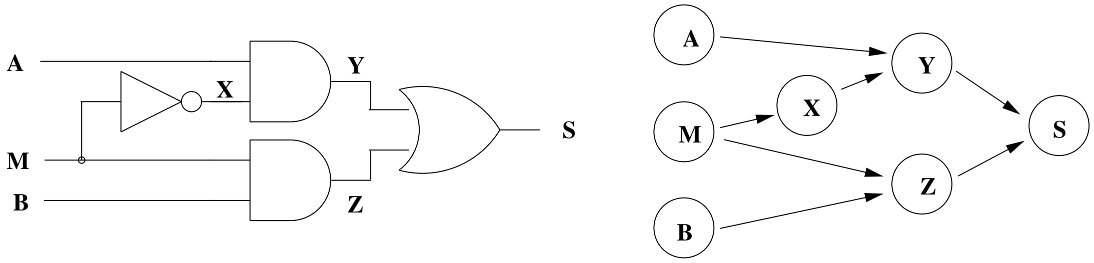


FIG. 4.2 –: Un multiplexeur en portes et sa représentation en réseau booléen à plat

Dans l'exemple de la figure 4.2, le signal M permet de choisir entre A et B pour affecter la sortie S. Si une valeur logique '0' est présente sur M alors que A et B restent à 'X', seul le chemin de A vers S reste actif, éliminant le chemin de B vers S. La figure 4.3.a décrit l'état du réseau après propagation des valeurs logiques. La figure 4.3.b décrit l'état du réseau après invalidation des arcs inutiles.

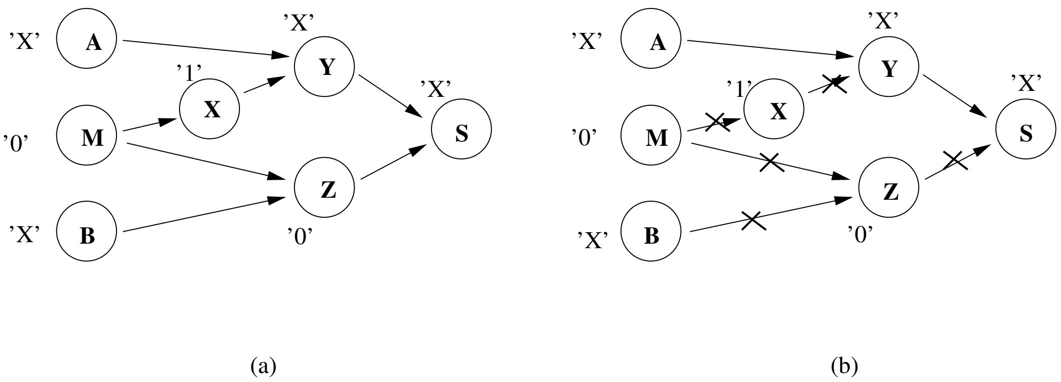


FIG. 4.3 –: Le réseau booléen du multiplexeur, de la figure précédente, sous contraintes

En ce qui concerne les *bi-nœuds*, si le nœud virtuel correspondant à la condition d'écriture est fixé à '0', alors l'ensemble des arcs émis par le *bi-nœud* sont invalidés.

La figure 4.4 représente le réseau booléen sous contraintes du registre de la page 51 dont le signal MODE est collé à '1' et le signal SET est collé à '1'.

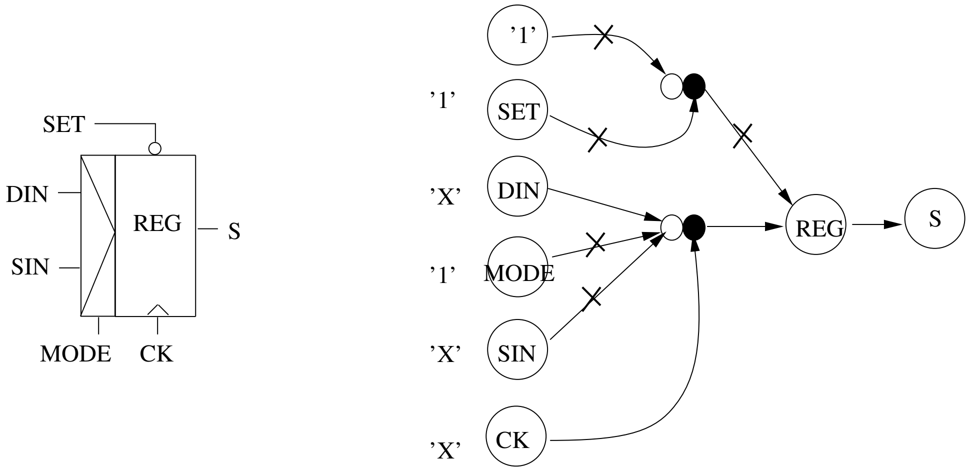


FIG. 4.4 –: Le graphe d'un registre sous contraintes

Ces simplifications vont permettre à la simulation symbolique de prendre en compte seulement les chemins du circuit réellement actifs durant l'exploitation de la structure de test concernée.

4.1.2 La simulation symbolique

La simulation symbolique d'un circuit consiste à évaluer son comportement en utilisant des valeurs symboliques codant un ensemble de conditions de fonctionnement du circuit [Bry90].

La simulation symbolique est une extension de la simulation logique. Elle associe à chaque signal une *valeur symbolique*. La propagation des valeurs symboliques à travers les expressions booléennes du circuit est assurée par des *fonctions de transfert*.

Les valeurs symboliques

Une valeur symbolique est un n-uplet d'*attributs symboliques*. Chaque attribut prend une valeur parmi un ensemble de valeurs spécifiques à cet attribut, dont la valeur nulle. Du point de vue sémantique, les attributs symboliques représentent un ensemble de propriétés statiques du signal auquel ils sont rattachés, ainsi que les caractéristiques des chemins fonctionnels qui atteignent ce signal, soit depuis les entrées primaires du circuit, soit depuis le dernier point mémoire rencontré. La sémantique associée à chaque attribut dépend du type de circuit utilisé. Le tableau 4.1 donne un exemple d'attributs symboliques utilisés dans le cadre des circuits synchrones monophase.

L'attribut **AC** caractérise les horloges. Les valeurs **CK**, **NCK** et **XCK** représentent respectivement l'horloge, l'horloge inversée et enfin une horloge dont

Attribut	**Liste de valeurs**
AC	0, CK, NCK, XCK
AS	0, S.DCK, NS.DCK, XS.DCK

TAB. 4.1 –: *Attributs symboliques et leurs valeurs*

AS	**Fonction IP**	**Fonction AP**
0	0	0
S.DCK	NS.DCK	XS.DCK
NS.DCK	S.DCK	XS.DCK
XS.DCK	XS.DCK	XS.DCK

TAB. 4.2 –: *Fonctions* **IP** *et* **AP** *de l'attribut* **AS**

l'information de polarité a été perdue (suite à la traversée d'un ou-exclusif par exemple). La valeur **0**, présente pour tous les attributs, est la valeur nulle de cet attribut.

L'attribut **AS** caractérise les signaux séries du *scan-path* synchronisé sur un front descendant de l'horloge. Les valeurs préfixées **S**, **NS** et **XS** représentent une polarité respectivement identique, inversée et inconnue par rapport à l'entrée série du *scan-path*.

L'ensemble des attributs et de leurs valeurs associées peut être augmenté de façon à prendre en compte de nombreux styles de conception : *scan-path* multiples, horlogerie multi-phase [Hir93].

Les fonctions de transfert

Il existe deux catégories de fonctions de transfert. Les premières permettent de propager les valeurs symboliques à travers les nœuds du réseau booléen. Les secondes concernent les bi-nœuds.

Les fonctions de transfert symboliques sont appliquées attribut par attribut. Il existe trois fonctions de transfert par attribut. La fonction **IP** représente le passage d'un attribut par une inversion de polarité. La fonction **AP** représente la perte d'information de polarité pour un attribut. Enfin la fonction **NINV** permet de combiner deux valeurs d'un même attribut et est utilisée pour les expressions qui possèdent plus d'une variable. Les tableaux 4.2 et 4.3 décrivent ces trois fonctions pour l'attribut **AS** défini plus haut.

Pour appliquer ces fonctions à chaque expression booléenne du réseau simplifié, Nous procédons de la manière suivante :

As	**0**	**S.DCK**	**NS.DCK**	**XS.DCK**
0	0	S.DCK	NS.DCK	XS.DCK
S.DCK	S.DCK	S.DCK	XS.DCK	XS.DCK
NS.DCK	NS.DCK	XS.DCK	NS.DCK	XS.DCK
XS.DCK	XS.DCK	XS.DCK	XS.DCK	XS.DCK

TAB. 4.3 –: *Fonctions* **NINV** *de l'attribut* **AS**

– La polarité de chaque variable du support de l'expression booléenne par rapport à sa sortie est calculée. Elle peut être paire, impaire ou indéfinie.

– Pour chaque valeur symbolique rattachée aux variables du support de l'expression booléenne, nous appliquons, attribut par attribut, la fonction correspondant à sa polarité dans l'expression : la fonction **IP** pour une polarité impaire, la fonction **AP** pour une polarité indéfinie. Dans le cas d'une polarité paire, aucune fonction n'est appliquée.

– Nous appliquons à l'ensemble des valeurs symboliques ainsi obtenues, attribut par attribut et deux à deux, la fonction **NINV**.

.

La figure 4.5 illustre quelques exemples de caractérisation d'expressions booléenne. La figure 4.6 nous montre l'application des fonctions de transfert pour l'attribut **AS** sur un exemple. La notation **{NS.DCK}** signifie que l'attribut **AS** a pour valeur **NS.DCK**.

```
s <= a and (not b);

s <= a xor b;

s <= (a and sel) or
     (b and (not sel));
```

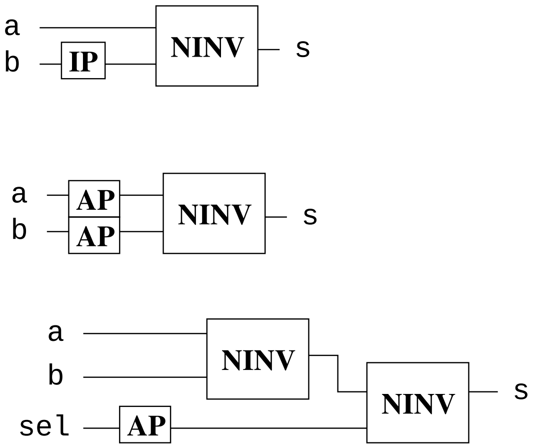


FIG. 4.5 –: Caractérisation par les fonctions de transfert de quelques expressions booléennes

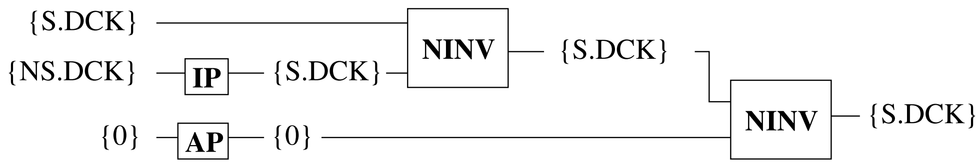


FIG. 4.6 –: Exemple de propagation de l'attribut **AS** à travers des fonctions de transfert

Concernant les bi-nœuds bus « tristate », les fonctions de transfert utilisées sont les mêmes que dans le cas des expressions booléennes associées aux portes combinatoires. Les bus « tristate » sont en effet considérés équivalents à des multiplexeurs du point de vue de la simulation symbolique, la simulation logique ayant invalidé les émetteurs inactifs.

Pour les registres, une fonction de transfert **FTS** combine les attributs symboliques présents sur chacun des nœuds virtuels d'un bi-nœud. Cette fonction prend aussi en compte la nature du registre, si le bi-nœud est rendu actif par un front ou un état du nœud virtuel correspondant à la condition d'écriture. Il existe des registres de quatre natures différentes. Les bi-nœuds correspondants sont marqués lors de la construction du graphe à l'aide d'un des marqueurs suivants : REGISTRE_FRONT_DESCENDANT, REGISTRE_FRONT_MONTANT, REGISTRE_ETAT_HAUT et REGISTRE_ETAT_BAS. Par exemple, un attribut **AC** de valeur **CK** présent sur le nœud virtuel de condition d'un bi-nœud REGISTRE_FRONT_DESCENDANT et un attribut **AS** de valeur **S.DCK** présent sur le nœud virtuel de donnée vont générer un attribut **AS** de valeur **S.DCK** et un attribut **AC** de valeur nulle en sortie du bi-nœud.

Chaque attribut possède une fonction **FTS** décrite de façon procédurale et prenant en compte l'ensemble des attributs des deux nœuds virtuels d'un bi-nœud. Nous donnons un exemple de la fonction **FTS** pour l'attribut **AS** défini plus haut dans la figure 4.7. **VC** est la valeur symbolique attachée au nœud virtuel de la condition. **VD** est la valeur symbolique attachée au nœud virtuel de la donnée. La fonction `attribut(valeur VS,attribut A)` donne la valeur de l'attribut `A` de la valeur symbolique `VS`. La fonction `nature(BN)` renvoie la nature du registre représenté par le bi-nœud `BN`.

La propagation des attributs symboliques se fait depuis les entrées primaires du circuit vers les points mémoires et les sorties. Des *valeurs symboliques simples* sont associées aux entrées primaires du circuit. Une valeur symbolique simple ne possède qu'un seul attribut symbolique non nul. La propagation se fait en une seule passe et à travers les chemins actifs du circuit, la phase de simulation logique ayant invalidé les arcs non actifs du graphe. Une exploration des valeurs symboliques à des points stratégiques du circuit permet d'en extraire certaines

```
SI (attribut(VD,AS) == S.DCK) ALORS
   SI (nature(BN) == REGISTRE_FRONT_DESCENDANT) ALORS
      SI (attribut(VC,AC) == CK) ALORS
         RETOURNE S.DCK
      SINON
         RETOURNE 0
      FIN SI
   SINON SI (nature(BN) == REGISTRE_FRONT_MONTANT) ALORS
            SI (attribut(VC,AC) == NCK) ALORS
               RETOURNE S.DCK
            SINON
               RETOURNE 0
            FIN SI
   SINON RETOURNE 0
   FIN SI
FIN SI
```

FIG. 4.7 –: Fonction de transfert **FTS** de l'attribut **AS**

propriétés, selon la sémantique qui est associée aux attributs définis. Dans notre exemple, ce sont des propriétés de *scan-path* et d'inversions qui vont être extraites.

4.2 Application à la reconnaissance automatique des signaux observables et contrôlables dans un circuit

Un signal du circuit est dit contrôlable lorsqu'il est possible d'imposer une valeur à ce signal depuis l'interface du circuit, indépendamment de sa fonctionnalité. De même, un signal du circuit est dit observable lorsqu'il est possible de lire sa valeur depuis l'interface du circuit. Ainsi les entrées primaires d'un circuit sont naturellement contrôlables et les sorties primaires sont naturellement observables.

Nous avons vu au chapitre 1 que la technique du *scan-path* est très largement utilisée pour augmenter le nombre de signaux contrôlables et observables d'un circuit.

Il n'existe pas de définition structurelle formelle du *scan-path*. Par contre il est possible d'en donner une définition fonctionnelle. Cette structure de test possède un mode d'utilisation particulier appelé mode décalage, qui transforme un sous-ensemble des points mémoires du circuit en un registre à décalage. Il est ainsi possible de contrôler et d'observer des signaux internes depuis l'interface du circuit par l'intermédiaire de ces registres.

Lorsque le circuit est en mode fonctionnel, les valeurs présentes dans le registre à décalage permettent de contrôler un sous-ensemble de signaux du circuit. Nous appelons ces signaux les points contrôlables du *scan-path*. Le mode fonctionnel permet aussi d'échantilloner un autre sous-ensemble des signaux du circuit en recueillant leurs valeurs au sein du registre à décalage. Ces signaux sont les points observables du *scan-path*.

La connaissance de ce *scan-path* au niveau du circuit se résume généralement :

- aux connecteurs externes qui correspondent à l'entrée série et à la sortie série du registre à décalage,
- aux connecteurs externes qui permettent de contrôler le fonctionnement de ce registre,
- à la séquence logique à appliquer pour mettre le *scan-path* en mode décalage,
- et à la séquence logique qui permet de mettre le circuit en mode fonctionnel.

La longueur de ce *scan-path*, c'est à dire le nombre de points mémoires qui forment le registre à décalage, n'est pas forcément connue.

Les informations que nous souhaitons extraire pour procéder au découpage automatique du circuit sont les suivantes :

1. la liste ordonnée des points mémoires qui forment le *scan-path*.
2. La liste des signaux contrôlables de ce *scan-path*. Chacun est associé à un point mémoire, le nombre d'inversions depuis l'entrée série jusqu'à celui-ci étant connu.
3. La liste des signaux observables de ce *scan-path*. Chacun est associé à un point mémoire, le nombre d'inversions depuis celui-ci jusqu'à la sortie série étant connu.
4. L'ensemble des portes logiques qui appartiennent à ce *scan-path*, c'est à dire le chemin lui-même ainsi que les portes qui font partie de la logique de contrôle de ce chemin.

L'exploitation des deux séquences logiques correspondant au mode décalage et au mode fonctionnel à travers les simulations logiques et symboliques va permettre de collecter ces informations.

4.2.1 Identification des registres observables et contrôlables

Le circuit est tout d'abord mis en mode décalage du *scan-path* par simulation logique. Dans le cas d'un *scan-path* simple, cela revient souvent à appliquer une valeur logique sur un signal de mode, mais cette méthode supporte des initialisations plus complexes comme dans le cas des *scan-path* dont la mise en œuvre passe par la circuiterie *Boundary-Scan* [IEE90].

Avant de procéder à la simulation symbolique, des valeurs symboliques simples sont placées sur les entrées du circuit. Une valeur symbolique simple ne possède qu'un seul attribut dont la valeur n'est pas nulle.

L'inspection des valeurs des attributs symboliques présents sur les points mémoires du circuit à l'issue de la simulation symbolique permet alors d'identifier les registres observables et contrôlables correspondant au *scan-path*. Enfin, l'exploration des nœuds depuis l'entrée série du *scan-path* jusqu'à la sortie permet d'ordonner ces points.

Le circuit de la figure 4.8 est un circuit séquentiel synchrone fonctionnant sur le front descendant de l'horloge CK. Il possède un *scan-path* partiel, ce qui signifie que seul un sous-ensemble de ses points mémoires sont dans un chemin de test qui commence sur l'entrée SCANIN et se termine sur la sortie SCANOUT.

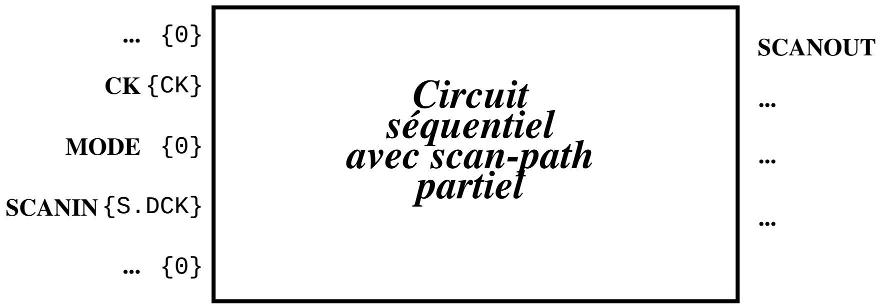


FIG. 4.8 –: Une portion de circuit séquentiel possédant des valeurs symboliques simples sur ses entrées primaires

Utilisant le jeu d'attributs symboliques défini page 60, une valeur symbolique simple **{CK}** est placée sur l'entrée CK. Une valeur symbolique simple **{S.DCK}** est placée sur l'entrée SCANIN. Des valeurs symboliques nulles sont placées sur les autres entrées primaires. L'entrée MODE possède une valeur logique déterminée qui permet d'activer le chemin de test du circuit.

A la fin de l'unique cycle de simulation symbolique, des valeurs symboliques sont présentes sur tous les nœuds du graphe.

Les registres possèdant une valeur à polarité définie pour l'attribut **AS** font partie du chemin de test. Il existe en effet un chemin avec un nombre pair d'inversions depuis l'entrée série du *scan-path* jusqu'à un registre qui possède une valeur **{S.DCK}**. De même, il existe un chemin avec un nombre impair d'inversions depuis l'entrée série du *scan-path* jusqu'à un registre qui possède une valeur **{NS.DCK}**. Les registres possèdant une valeur **{XS.DCK}** ou **{0}** pour l'attribut **AS** ne font pas partie du chemin de test.

Pour ordonner la liste des registres de *scan-path*, nous allons chercher dans le graphe sous contraintes les chemins partant de l'entrée série et arrivant à la sortie série pour lesquels l'ensemble des nœuds traversés possède une valeur avec une polarité définie pour l'attribut **AS**. Un chemin est alors défini par l'enchaînement des nœuds correspondants aux registres traversés avec leurs polarités depuis l'entrée série associée.

Si aucun chemin n'est trouvé, c'est que le mode décalage n'est pas correctement défini. Cette méthode est applicable même si le chemin de test entre deux registres traverse de la logique combinatoire.

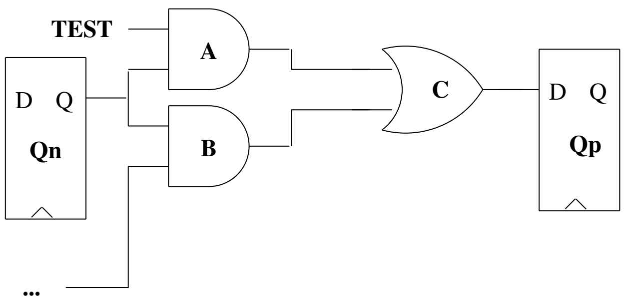


FIG. 4.9 –: Une fraction de *scan-path* issue d'un circuit synthétisé

Les figures 4.9 et 4.10 présentent une partie de *scan-path* obtenue par synthèse logique. Lorsque TEST est forcé à '1', le point mémoire Q_p prend la valeur du point mémoire Q_n bien que les trois portes A,B et C ne forment pas un vrai multiplexeur.

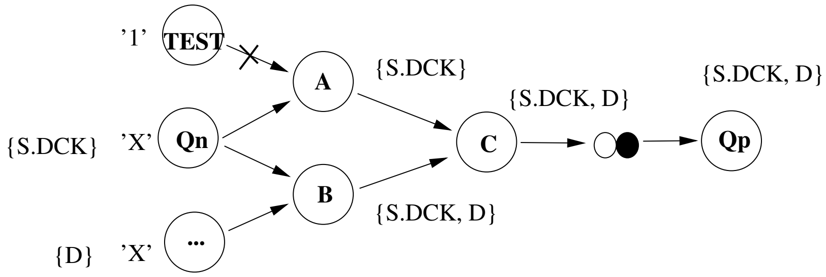


FIG. 4.10 –: La représentation en graphe sous contraintes correspondant à la figure précédente

Si plusieurs chemins non équivalents sont trouvés, alors le mode décalage n'est pas correctement défini. Sinon, l'ensemble des nœuds traversés par les chemins sont alors marqués comme appartenant à la structure de test.

A l'issue de cette recherche, nous possédons la liste ordonnée des nœuds représentant les points mémoires du registre de *scan-path*. À chacun de ces nœuds est associée une polarité par rapport à l'entrée série du registre. L'étape suivante va consister à trouver l'ensemble des signaux observés et contrôlés par ces points mémoires.

4.2.2 Identification des signaux observables et contrôlables

Une exploration des nœuds en amont et en aval de chaque registre dans le graphe du circuit sous contraintes du mode fonctionnel permet d'identifier les éventuels nœuds observables et contrôlables rattachés à chaque registre.

Le circuit est tout d'abord mis sous contraintes du mode fonctionnel, à l'aide d'une phase de simulation logique.

Puis le graphe contraint est parcouru en aval. Tous les nœuds ne possèdant qu'un prédécesseur et qu'un successeur sont traversés. Si le successeur possède plusieurs prédécesseurs, alors le nœud courant est un signal contrôlable. Si le successeur possède plusieurs successeurs, alors c'est lui qui représente le signal contrôlable. Si aucun nœud remplissant ces critères n'est trouvé, c'est que le point mémoire ne contrôle pas de signal dans le circuit lorsque celui-ci est en mode fonctionnel.

Enfin le graphe contraint est parcouru en amont depuis la partie bi-nœud correspondant à la donnée écrite dans le point mémoire. Tous les nœuds ne possédant qu'un prédécesseur actif sont traversés. Le dernier nœud traversé représente le signal observable. Si aucun nœud remplissant ces critères n'est trouvé, c'est que le point mémoire n'observe aucun signal dans le circuit en mode fonctionnel.

La figure 4.11 illustre ces deux parcours – amont et aval – depuis un point mémoire.

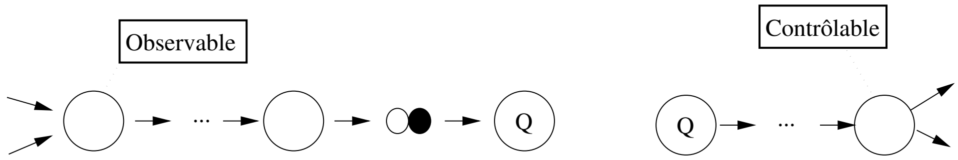


FIG. 4.11 –: Parcours amont et aval depuis un point mémoire pour marquer les nœuds observables et contrôlables

Les nœuds correspondant aux signaux observables et contrôlables sont ainsi marqués CONTRÔLABLES ou OBSERVABLES. Le nombre d'inversions depuis le point mémoire pour les signaux contrôlables et jusqu'au point mémoire pour les signaux observables est enregistré.

4.2.3 Identification de la logique de contrôle

La logique de contrôle d'une structure de test est composée de l'ensemble des portes ayant participé à la mise en place d'un des modes de cette structure.

Nous allons marquer CONTRÔLE l'ensemble des nœuds ayant pris une valeur logique définie, c'est à dire différente de 'X', au cours des deux simulations logiques destinées à mettre le circuit sous contraintes du mode décalage et sous contraintes du mode fonctionnel. Cet ensemble est appelé logique de contrôle de la structure de test.

Nous avons besoin d'identifier cette logique pour qu'à l'issue de la phase de découpage il soit possible de vérifier que chacun des nœuds du graphe est directement testé car il appartient à une unité testable, ou bien qu'il est implicitement testé car il appartient à la logique de contrôle d'un mode de test.

4.2.4 Exemple

La figure 4.12 représente le graphe du circuit de la figure 4.8 page 65. Il possède un *scan-path* composé de deux bascules. Seuls les nœuds qui appartiennent au *scan-path* sont explicités. Les nœuds N_{11} à N_n représentent la logique fonctionnelle du circuit. Certains de ces nœuds correspondent donc à des registres qui ne sont pas dans le *scan-path*.

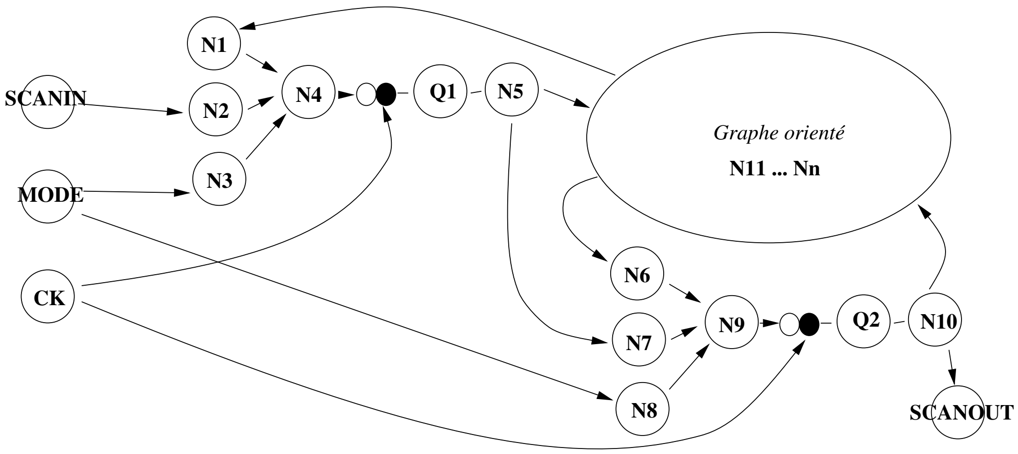


FIG. 4.12 –: Le graphe d'un circuit possédant un *scan-path* composé de 2 bascules

Dans un premier temps, le circuit est mis en mode décalage. Cela se fait par l'application d'un '1' sur l'entrée MODE. La simulation symbolique et l'inspection des valeurs permettent d'identifier les nœuds Q_1 et Q_2 comme formant le registre à décalage. Les nœuds SCANIN, N_2, N_4, Q_1, N_5, N_7, N_9, Q_2, N_{10} et SCANOUT sont marqués comme appartenant au chemin de *scan-path*.

Dans un second temps, le circuit est mis en mode fonctionnel. Cela se fait par l'application d'un '0' sur l'entrée MODE. À partir du nœud Q_1, le nœud N_5 en aval répond aux critères pour être un point contrôlable, et est donc marqué CONTRÔLABLE. Le nœud N_1 en amont répond aux critères pour être un point observable et est donc marqué OBSERVABLE. De même, les nœuds N_{10} et N_6 sont marqués respectivement CONTRÔLABLE et OBSERVABLE grâce à un parcours aval et un parcours amont depuis Q_2.

Enfin, au cours des deux simulations logiques les nœuds MODE, N_3 et N_8 sont marqués car ils ont pris une valeur logique '0' et '1' : c'est la logique de contrôle du *scan-path*.

4.3 Conclusion

Notre méthode fonctionnelle de reconnaissance à l'aide de la simulation s'adapte à de nombreuses formes de *scan-path*. Puisqu'elle ne s'appuie sur aucune bibliothèque prédéfinie, elle permet d'identifier toutes les formes de *scan-path* répondant à la définition fonctionnelle donnée plus haut. De plus cette reconnaissance a aussi pour effet de vérifier que le *scan-path* est bien connecté ou que les séquences procurées sont bonnes, puisque dans le cas contraire aucun chemin n'est trouvé.

Si le circuit possède plusieurs *scan-path*, la procédure est répétée séquentiellement pour chacun d'entre eux. L'ensemble des marqueurs rajoutés par la reconnaissance d'un *scan-path* reste associé à celui-ci. Un nœud peut donc par exemple être marqué OBSERVABLE du point de vue d'un *scan-path*, et CONTRÔLABLE du point de vue d'un autre.

Enfin nous verrons que ces techniques peuvent aussi être appliquées à la reconnaissance de l'architecture *Boundary-Scan* et aux formes d'auto-test à base de registre à décalage.

Chapitre 5

Découpage en unités testables faiblement recouvrantes

Nous allons dans cette partie nous intéresser au problème du découpage d'une description structurelle à grain fin éventuellement hiérarchique en unités testables faiblement recouvrantes. Cette description structurelle prend la forme du graphe orienté décrit au chapitre 3. Sur ce graphe sont disposés des marqueurs permettant de savoir si un nœud est observable ou contrôlable.

Les frontières du découpage du graphe sont ces nœuds observables et contrôlables. Le graphe est découpé en deux temps. Tout d'abord il est virtuellement découpé en unités testables élémentaires ne possédant qu'une seule sortie, un nœud observable du point de vue de la structure de test courante, et plusieurs entrées contrôlables. Puis un mécanisme permet de regrouper plusieurs unités testables élémentaires pour fabriquer des unités testables à plusieurs sorties, en minimisant le recouvrement entre deux unités testables.

5.1 Découpage en unités testables élémentaires

Une *unité testable* (notée UT) est une sous-partie du circuit sur laquelle des vecteurs déterministes peuvent être appliqués depuis l'interface. Le résultat doit aussi pouvoir être lu sur l'interface du circuit. Les sorties d'une unité testable doivent donc être observables et ses entrées contrôlables. Enfin la description structurelle d'une unité testable peut être sauvegardée dans un fichier simulable de façon à générer (automatiquement ou manuellement) les vecteurs de test.

Une *unité testable élémentaire* (notée UTE) est une unité testable ne comportant qu'une seule sortie. Il y a donc autant d'unités testables élémentaires dans un circuit qu'il y a de nœuds observables dans son graphe.

Le découpage du graphe en unités testables élémentaires va servir de base au découpage en unités testables. Il est très similaire à celui utilisé par l'outil CONTEST de l'université de Hannovre [GMDz90], mais il n'est pas limité ici aux seuls circuits combinatoires.

5.1.1 Algorithme de découpage

La première étape consiste à découper le graphe en unités testables élémentaires.

Ce découpage s'effectue grâce à un parcours en arrière dans le graphe du circuit mis sous contraintes. Cette mise sous contraintes permet de n'explorer que les chemins effectivement actifs pendant l'utilisation de la structure de test. Le mode utilisé pour mettre le graphe sous contraintes est celui qui a permis l'identification des nœuds observables et contrôlables. Ainsi dans le cas d'un *scan-path*, c'est le mode fonctionnel qui est utilisé.

Le parcours commence sur un nœud marqué OBSERVABLE. L'ensemble des chemins actifs en amont est parcouru jusqu'à rencontrer un nœud marqué CONTRÔLABLE. Au passage tous les nœuds traversés sont marqués avec une référence au nœud observable courant. Si un bi-nœud est rencontré, l'ensemble des chemins actifs correspondant aux deux nœuds virtuels sont parcourus. Les registres non « scannables » ou les bus « tristate » présents sur le parcours font partie de l'unité testable élémentaire.

La fonction récursive MARQUE_UTE, qui est appliquée successivement à chaque nœud observable, est présentée dans la figure 5.1. N est le nœud courant et NO est le nœud observable à partir duquel a commencé le parcours. Un marqueur temporaire appelé FAIT permet à l'algorithme récursif de ne pas boucler à l'intérieur des cycles présents dans le réseau (l'existence de registres non « scannables » rend possible l'existence de cycles dans le réseau sous contraintes).

Le découpage en unités testables élémentaires d'un circuit combinatoire est présenté dans la figure 5.2[1]. Les sorties **I** et **J** sont marquées OBSERVABLES, et les nœuds **A**, **B**, **C** et **D** sont marqués CONTRÔLABLES. Le nœud **A** est observable à travers **I** et **J**, alors que le nœud **C** n'est observable qu'à travers **J**.

1. Cet exemple correspond au circuit combinatoire de la figure 3.5 page 52

```
MARQUE_UTE (nœud courant N, nœud observable NO)
      SI N est marqué 'FAIT'
         FIN MARQUE_UTE
      Marquer N avec le marqueur 'FAIT'
      Ajouter NO à la liste des nœuds observables de N
      SI N est marquée 'CONTRÔLABLE'
         FIN MARQUE_UTE
      SI N est un nœud simple
         POUR chaque nœud NI_i en entrée de N
                 Si l'arc NI_i → N est actif
                    MARQUE_UTE (NI_i,NO)
      SINON (N est un point mémoire ou un tristate)
         POUR chaque bi-nœud BNI_i en entrée de N
                 BNID_i est le nœud virtuel 'donnée'
                 Si l'arc BNID_i → N est actif
                    MARQUE_UTE (BNID_i,NO)
                 BNIC_i est le nœud virtuel 'commande'
                 Si l'arc BNIC_i → N est actif
                    MARQUE_UTE (BNIC_i,NO)
FIN MARQUE_UTE
```

FIG. 5.1 –: Algorithme de découpage du graphe en UTE

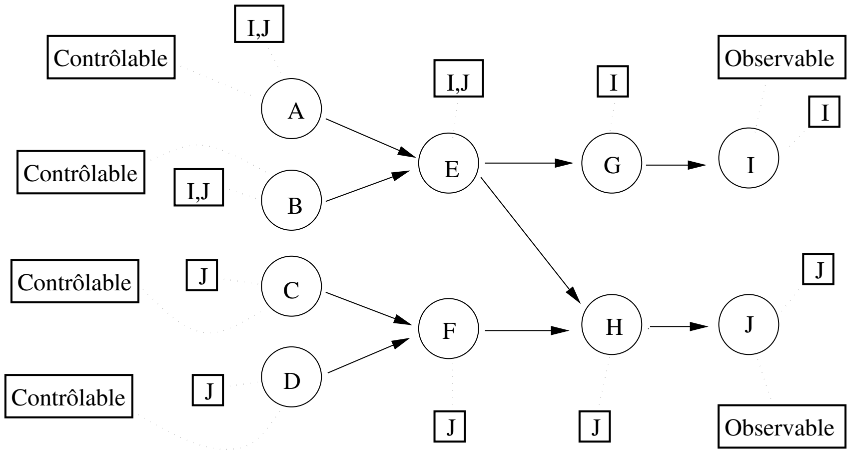


FIG. 5.2 –: Découpage en unités testables élémentaires d'un circuit combinatoire

La figure 5.3 illustre le passage à travers un registre non « scannable », c'est à dire ne possédant aucun nœud marqué CONTRÔLABLE en aval, et aucun nœud marqué OBSERVABLE en amont. La procédure de marquage va propager le nœud observable **S** jusqu'aux nœuds contrôlables **I** et **CK** puisqu'il n'y a aucun nœud marqué CONTRÔLABLE entre le registre **Q** et les entrées primaires.

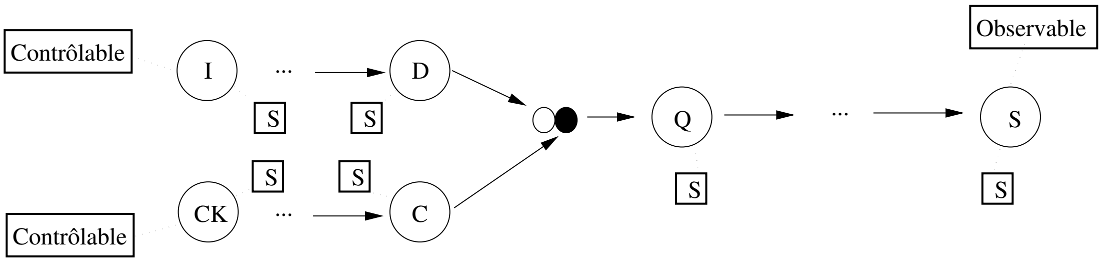


FIG. 5.3 –: Découpage en unités testables élémentaires à travers un registre non « scannable »

5.1.2 Description simplifiée d'une unité testable élémentaire

Une unité testable élémentaire peut être caractérisée par son unique sortie, un nœud observable. Elle est constituée de l'ensemble des nœuds du graphe possédant ce nœud observable dans leurs marqueurs. Dans le cas de la figure 5.2, l'unité testable élémentaire rattachée à I est composée des nœuds A,B,E,G et I et l'unité testable élémentaire rattachée à J est composée des nœuds A,B,C,D,E,F,H et J.

Une unité testable peut aussi être décrite par la liste des nœuds contrôlables qu'elle contient. Cette description simplifiée permet de réduire considérablement l'espace mémoire requis pour stocker les unités testables.

Les *descriptions simplifiées* des unités testables élémentaires de la figure 5.2 sont donc notées :

$UTE_I = \{A, B\}$
$UTE_J = \{A, B, C, D\}$

5.2 Regroupement d'unités testables élémentaires

Le graphe du circuit étant découpé en unités testables élémentaires, des vecteurs peuvent être générés sur chacune de ces unités séparément. Nous avons pourtant choisi de regrouper ces unités testables élémentaires en unités testables plus importantes. Il y a pour cela deux raisons : le caractère recouvrant des unités testables élémentaires d'une part, et leur trop grand nombre d'autre part.

5.2.1 Fort recouvrement des unités testables élémentaires

Deux unités testables élémentaires peuvent avoir un grand nombre de portes en commun : le circuit de la figure 5.4 possède deux unités testables élémentaires, UTE_{out} et $UTE_{\overline{OUT}}$. Ces deux UTE sont recouvrantes : l'ensemble des nœuds appartenant à UTE_{OUT} est entièrement inclus dans $UTE_{\overline{OUT}}$. Si des vecteurs sont générés séparément pour UTE_{out} et $UTE_{\overline{OUT}}$, les vecteurs couvrant les fautes d'$UTE_{\overline{OUT}}$ vont donc aussi couvrir toutes les fautes d'UTE_{out}. Un double effort va alors être fait pour générer des vecteurs, et une séquence deux fois plus longue va être appliquée, sans aucune amélioration du taux de couverture.

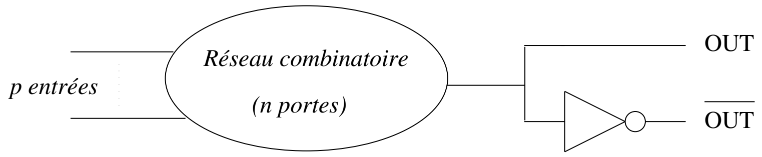


FIG. 5.4 –: Un circuit possédant deux unités testables élémentaires fortement recouvrantes

Une façon d'évaluer l'effort supplémentaire de test engendré par un découpage est de calculer le *facteur d'augmentation* qui lui est lié. Le facteur d'augmentation est défini comme étant la somme du nombre de portes appartenant à chaque unité testable divisé par le nombre total de portes du circuit. Si le circuit n'est découpé qu'en une seule unité testable, ce facteur vaut donc 1. Chaque porte appartenant à plus d'une unité testable fait augmenter ce facteur. Pour le circuit de la figure 5.4, le facteur d'augmentation dû à un découpage en unité testables élémentaires

Circuit	**% scan**	**UTE**			**FA**
		Nombre d'UTE	Taille Moy. (en portes)	Taille Max. (en portes)	
s13207	50	457	228	2513	12,7
	80	619	63	1576	4,85
	100	714	36	1207	3,26
s15850	50	413	1227	4647	50,58
	80	568	321	2121	18,48
	100	670	154	1104	10.60
s35932	50	1056	875	8327	54,62
	80	1478	214	3666	19,33
	100	1760	16	87	1.78

TAB. 5.1 –: *Découpage en unités testables élémentaires de trois circuits ISCAS séquentiels possédant un scan-path*

est :

$$FA = \frac{TAILLE(UTE_{OUT})+TAILLE(UTE_{\overline{OUT}})}{TAILLE(CIRCUIT)} = \frac{n+(n+1)}{(n+1)} \simeq 2$$

Les résultats présentés dans le tableau 5.1 montrent la valeur de ce facteur pour trois des plus gros circuits ISCAS séquentiels. Pour chaque circuit, nous présentons trois versions : une avec un *scan-path* partiel comprenant 50% des registres du circuit, la seconde avec 80% des registres dans le *scan-path*, et la dernière avec un *scan-path* complet.

Nous voyons ici que générer des vecteurs séparément pour toutes les unités testables élémentaires issues du découpage du circuit s15850 équipé d'un *scan-path* complet entraînerait un effort de génération de vecteurs sur un circuit plus de dix fois plus grand, et jusqu'à cinquante fois plus grand lorsque seulement 50% des registres sont inclus dans le *scan-path*.

C'est bien sûr lorsqu'un circuit possède un *scan-path* complet que le facteur d'augmentation dû à un découpage en UTE est le plus faible, puisque le nombre de points observables et contrôlables est le plus élevé.

Il est donc indispensable de procéder à un regroupement des unités testables élémentaires les plus recouvrantes pour faire baisser ce facteur d'augmentation.

5.2.2 Complexité et nombre des unités testables élémentaires

Les complexités moyennes des unités testables élémentaires du tableau 5.1 ne dépassent pas quelques centaines de portes pour les plus grosses. Pour le découpage en UTE du circuit s35932 avec *scan-path* complet par exemple, nous obtenons 1760 unités testables élémentaires d'une complexité moyenne de 16 portes. Nous avons vu au chapitre 2 que les capacités des outils automatiques de génération de vecteurs de test ne permettent pas de générer des vecteurs sur des circuits trop complexes. Cependant, il est avantageux d'utiliser les outils automatiques au maximum de leurs capacités. De plus le nombre important de manipulations de fichiers impliquées par la grande quantité d'unité testables élémentaires va considérablement ralentir le processus complet de génération. Nous risquons en effet de passer plus de temps à gérer les fichiers qu'à générer des vecteurs. Il est donc avantageux de ce point de vue de réduire le nombre d'unités testables.

Les capacités et les performances des outils de génération automatique de vecteurs de test sont fortement liées à la complexité du circuit traité, en nombre de portes, et dépendent beaucoup du niveau de séquentialité pour les circuits séquentiels. Le niveau de séquentialité représente le nombre de cycles minimum nécessaires au contrôle et à l'observation des registres non « scannables ». En pratique, tant que l'outil de génération fonctionne en mémoire centrale de la station de travail, les performances sont plus ou moins linéaires avec le nombre de portes, à niveau de séquentialité constant. Il est donc plus intéressant de générer des vecteurs sur des unités testables les plus grandes possibles tant qu'elles respectent les limites de l'outil de génération vis à vis de la station de travail.

Lors du regroupement des unités testables élémentaires en unités testables plus grandes, nous allons donc chercher un compromis entre le nombre et la taille des unités testables.

5.2.3 Formation des unités testables

Les unités testables sont formées par assemblage itératif partant de l'ensemble des unités testables élémentaires. Pour décider si deux unités testables doivent être rassemblées en une seule, nous devons posséder une mesure du nombre de nœuds communs à ces deux unités. Nous allons définir pour cela un *taux de recouvrement* de deux unités testables, s'appuyant sur les descriptions simplifiées d'unités testables définies précédemment.

Taille de contrôlabilité d'une unité testable

Comme nous l'avons expliqué plus haut, une unité testable n'est décrite que par la liste des nœuds contrôlables qu'elle contient. Nous considérons que le nombre d'entrées contrôlables est un bon évaluateur de la complexité d'une UTE. Cette hypothèse va nous permettre de mesurer les taux de recouvrement de manière approchée, mais beaucoup plus rapidement qu'en prenant en compte l'ensemble des nœuds d'une unité testable.

Nous définissons la taille de contrôlabilité (notée TC) d'une unité testable comme étant le nombre de nœuds contrôlables qu'elle contient. Dans le cas de la figure 5.2 de la page 74, nous obtenons :

$TC(UTE_I) = 2$
$TC(UTE_J) = 4$

Union de deux descriptions simplifiées d'unité testable

L'union de deux unités testables est formée de l'ensemble des nœuds appartenant à l'une ou l'autre, et est elle-même une unité testable. La description simplifiée de cette nouvelle unité testable est formée de l'ensemble des nœuds contrôlables lui appartenant. Nous pouvons donc directement obtenir la description simplifiée issue de l'union de deux unités testables à partir de leur descriptions simplifiées. Ainsi, l'unité testable UT issue de l'union des deux unités testables élémentaires de la figure 5.2 possède la description simplifiée suivante :

$$UT = UTE_I \cup UTE_J = \{A, B\} \cup \{A, B, C, D\} = \{A, B, C, D\}$$

Pour chaque union d'unités testables, la liste des unités testables élémentaires qui la forment est tenue à jour. La liste des UTE associées à l'unité testable décrite ci-dessus est $\{UTE_I, UTE_J\}$.

Intersection de deux descriptions simplifiées d'unité testable

L'intersection de deux descriptions simplifiées d'unités testables est formée de l'ensemble des nœuds appartenant aux deux descriptions. En effet, si aucun nœud n'est commun à deux unités testables, alors leurs descriptions simplifiées respectives n'ont, elles aussi, aucun nœud en commun. Par contre, si deux unités testables ont un nœud en commun, l'algorithme de découpage en UTE présenté page 73 implique alors que l'ensemble des nœuds en amont seront communs. Les points d'arrêt de ce parcours amont, qui sont des nœuds contrôlables, seront donc communs aux deux unités testables. Ainsi, nous pouvons évaluer le recouvrement de deux unités testables en ne se basant que sur leurs descriptions simplifiées.

Nous définissons l'intersection de deux descriptions simplifiées d'unités testables comme étant l'ensemble des nœuds appartenant aux deux descriptions simplifiées. Pour l'exemple ci-dessus, nous obtenons :

$$UT = UTE_I \cap UTE_J = \{A, B\} \cap \{A, B, C, D\} = \{A, B\}$$

Taux de recouvrement de deux unités testables

Nous définissons le taux de recouvrement de deux unités testables comme le rapport de la taille de contrôlabilité de leur intersection par la taille de contrôlabilité de la plus petite, normalisé sur une échelle de 0 à 100. Nous notons :

$$TR(UT_i, UT_j) = \frac{TC(UT_i \cap UT_j)}{MIN(TC(UT_i), TC(UT_j))} * 100$$

Si deux unités testables ne possèdent aucun nœud en commun, alors leur intersection est nulle et leur taux de recouvrement aussi.

Si une unité testable est complètement incluse dans une autre, alors la taille de la description de l'intersection est égale à la taille de la description de la plus petite des deux, et leur taux de recouvrement est égal à '100'.

Toutes les valeurs intermédiaires dénotent un recouvrement partiel entre deux unités testables. Une valeur proche de '100' indique un fort recouvrement. Il vaut mieux alors générer des vecteurs sur l'union des deux unités testables. Une valeur proche de '0' indique un faible recouvrement. Il est possible alors de générer des vecteurs séparément sur chaque unité testable.

Regroupement des unités testables

Notre algorithme permet en une seule passe de regrouper des unités testables dont le taux de recouvrement dépasse un certain seuil réglable TR_{MIN}.

Le point de départ est une liste d'unités testables élémentaires. L'objectif est d'obtenir une nouvelle liste d'unités testables possédant un facteur d'augmentation faible, mais avec des unités testables de taille raisonnable.

Nous nous trouvons dans le cas classique des problèmes NP-complets : une énumération de l'ensemble des solutions permettrait de selectionner la meilleure, mais le temps de calcul est alors rédhibitoire car exponentiel. Pour ces problèmes il existe cependant une manière polynômiale d'évaluer la qualité d'une solution [MG]. Il est alors possible de trouver une solution satisfaisante, sans garantie que ce soit la meilleure. Notre évaluateur ici est le facteur d'augmentation correspondant à un regroupement.

```
UT_1, ... , UT_n sont les n unités testables
POUR courante := 1 A n :
        POUR candidate = (courante + 1) à n
                calculer TR = TR(UT_courante,UT_candidate)
                SI (TR >TR_MIN)
                   UT_candidate = UT_courante ∪ UT_candidate
                   effacer UT_courante
                   ALLER A la courante suivante
                FIN SI
        FIN POUR
FIN POUR
```

FIG. 5.5 –: Algorithme de regroupement des UT avec un seuil TR_{MIN}

Notre algorithme ne permet pas donc pas d'obtenir le regroupement idéal, c'est à dire assurant le meilleur facteur d'augmentation pour un seuil donné. Nous avons préféré une approche « gloutonne » : une unité testable n'est pas associée avec celle pour laquelle le taux de recouvrement est le plus fort, mais avec la première pour laquelle ce taux est supérieur au seuil TR_{MIN}. Cela nous permet de ne pas avoir à calculer systématiquement les $\frac{n(n+1)}{2}$ taux de recouvrement correspondant aux n unités testables élémentaires, et de faire en une seule passe des regroupements multiples. L'algorithme utilisé est présenté dans la figure 5.5.

Nous avons opté pour une approche privilégiant la rapidité d'exécution. La précision n'est pas ici très importante. En effet les notions de taux de recouvrement et de facteur d'augmentation ne sont elle-mêmes que des approximations comme nous l'avons vu dans la partie 5.2.2.

Ce choix de la simplicité est particulièrement adapté aux volumes importants visés par notre méthode de test.

Sauvegarde des unités testables

A l'issue du regroupement, il ne reste que p unités testables, chacune étant décrite par la liste des nœuds contrôlables qui lui appartiennent, $\{liste\_NC\}$, et par la liste des unités testables élémentaires qui la composent, $\{liste\_UTE\}$.

Pour chaque unité testable il faut construire une nouvelle description structurelle simulable indépendante. Puis cette description structurelle est sauvegardée dans un fichier. C'est à l'issue de ces sauvegardes que le facteur d'augmentation dû au découpage est calculé, en comparant le nombre de portes sauvegardées dans les différentes descriptions structurelles au nombre total de portes.

Exemple

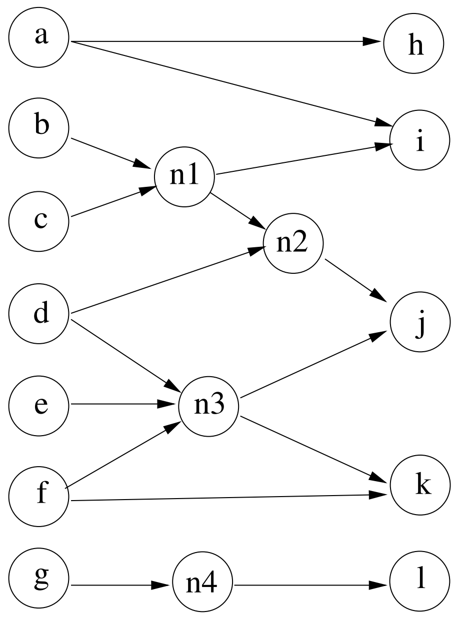


FIG. 5.6 –: Graphe orienté d'un circuit combinatoire

La figure 5.6 montre le graphe d'un circuit combinatoire possédant sept entrées contrôlables a, b, c, d, e, f et g, et cinq sorties observables h, i, j, k et l. Les cinq descriptions d'unités testables élémentaires sont les suivantes :

$UTE_h = \{a\}$
$UTE_i = \{a, b, c\}$
$UTE_j = \{b, c, d, e, f\}$
$UTE_k = \{d, e, f\}$
$UTE_l = \{g\}$

Prenons un seuil TR_{MIN} de 80. Les trois unités testables résultant de ce regroupement sont :

$UT_2 = \{a, b, c\}$ et la liste des UTE associée est $\{UTE_h, UTE_i\}$
$UT_4 = \{b, c, d, e, f\}$ et la liste des UTE associée est $\{UTE_j, UTE_k\}$
$UT_5 = \{g\}$ et la liste des UTE associée est $\{UTE_l\}$

Nous sauvegardons dans la description structurelle représentant UT_2 les portes qui correspondent aux nœuds h, i et n_1, dans la description structurelle représentant UT_4 les portes qui correspondent aux nœuds j, k, n_1, n_2 et n_3, et dans la description structurelle représentant UT_5 les portes qui correspondent aux nœuds l et n_4. Nous avons sauvegardé 10 portes alors que le circuit n'en contient que 9, puisque n_1 appartient à UT_2 et UT_4. Le facteur d'augmentation est donc de

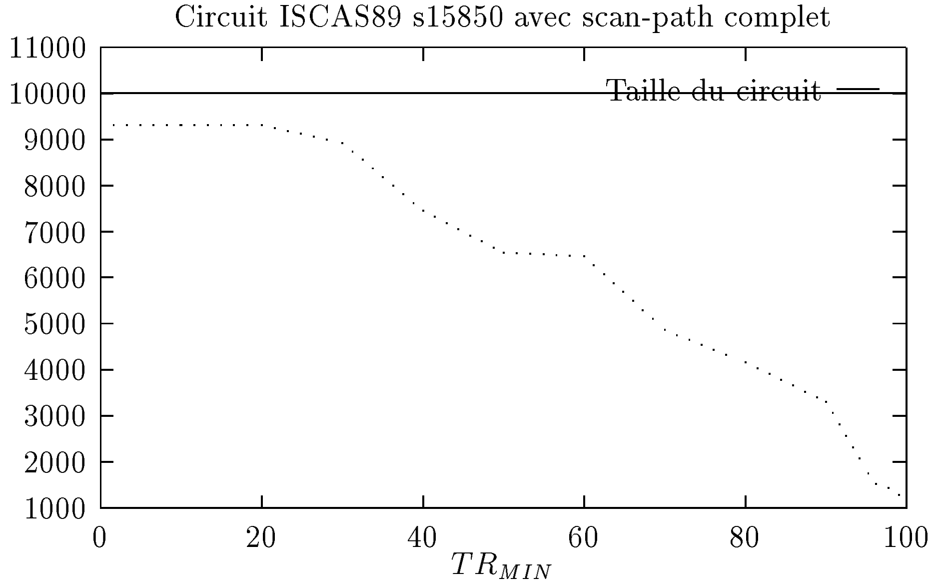


FIG. 5.7 –: Taille (en portes) de la plus grande UT pour différents seuils TR_{MIN}

$\frac{10}{9} = 1.11$. Enfin, la plus grande unité testable, UT_4, contient 5 portes.

Bien que notre exemple porte sur un circuit combinatoire, il n'y a aucune différence de méthode lorsque des UTE contiennent des registres.

Efficacité du découpage

Le résultat du découpage va dépendre de la valeur du seuil TR_{MIN}. Si celui-ci vaut 0, nous n'obtenons que quelques unités testables, de grande taille, et complètement indépendantes. Ce découpage est équivalent à celui effectué par l'outil PROSPECT [LCE86]. Si TR_{MIN} est fixé à 100, il n'y aura aucun regroupement, et chaque unité testable ne sera constituée que d'une unité testable élémentaire, à la manière du découpage en cônes de l'outil CONTEST [GMDz90]. Les courbes 5.7, 5.8 et 5.9 montrent comment varie le nombre d'unités testables, le facteur d'augmentation et la taille de la plus grande unité testable pour différentes valeurs du seuil TR_{MIN} sur le circuit ISCAS s15850.

Bien qu'il n'y ait pas de lien formel entre notre définition du taux de recouvrement minimal TR_{MIN} et la notion de facteur d'augmentation, la courbe 5.9 montre que ce facteur est une fonction croissante du seuil TR_{MIN}. La courbe 5.7 montre que la taille maximum est une fonction décroissante du seuil TR_{MIN}. Enfin, la courbe 5.8 montre l'influence de TR_{MIN} sur le nombre d'unités testables. Ainsi, le paramètre TR_{MIN} associé à notre algorithme remplit parfaitement son rôle de réglage de compromis entre le facteur d'augmentation et la taille et le nombre des unités testables. Un TR_{MIN} proche de 100 donne un mauvais facteur

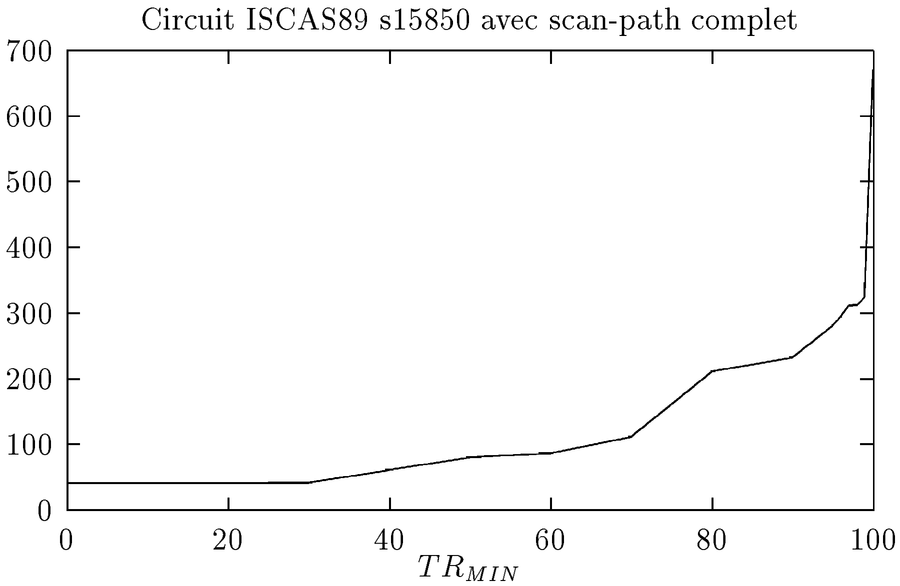


FIG. 5.8 –: Nombre d'unités testables pour différents seuils TR_{MIN}

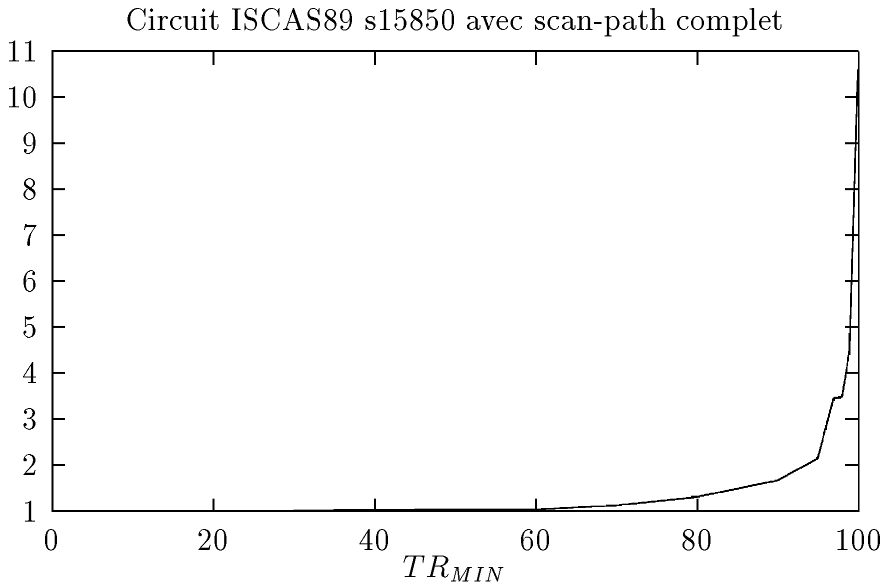


FIG. 5.9 –: Facteur d'augmentation pour différents seuils TR_{MIN}

d'augmentation, mais les unités testables sont petites. Un TR_{MIN} proche de 0 permet d'obtenir un bon facteur d'augmentation, mais la plus grande unité testable est presque aussi grande que le circuit lui-même.

Le choix d'un seuil TR_{MIN} par l'utilisateur dépend des capacités de l'outil de génération de vecteurs, et de la possibilité de paralléliser cette génération sur un réseau de machines.

5.3 Conclusion

Nous avons proposé un algorithme pour découper les parties non testées d'un circuit en unités testables faiblement recouvrantes. Ce découpage se base sur les notions d'unités testables élémentaires et de taux de recouvrement. Notre algorithme permet de regrouper ces unités testables élémentaires en une seule passe. Un paramètre réglable permet d'obtenir un compromis entre la taille et le nombre des unités testables d'une part, et leur caractère recouvrant d'autre part.

Les unités testables ainsi obtenues sont destinées à être exploitées par un outil externe de génération de vecteurs de test. Selon la nature du circuit, ces unités testables peuvent contenir des registres, et dans ce cas l'outil de génération doit être capable de traiter les circuits séquentiels. Enfin il est possible de distribuer la génération sur un réseau de machines puisque les unités testables sont indépendantes.

À l'issue de la génération de vecteurs de test, nous disposons d'une séquence de test et d'un dictionnaire de fautes par unité testable. Nous montrons au chapitre 7 comment ces informations sont assemblées pour ne disposer que d'une séquence de test et d'un dictionnaire de fautes pour tout le circuit.

Chapitre 6

Prise en compte de blocs auto-testés

Lorsqu'une partie d'un circuit dispose d'un mécanisme d'auto-test, il n'est pas nécessaire de générer des vecteurs, ceux-ci étant déjà fournis par la structure de test elle-même. Dans le cadre d'un découpage automatique en vue de la génération des tests, il faut donc veiller à ne pas inclure ces parties. De même, une macro-cellule possèdant des vecteurs destinés à la tester ne doit pas être incluse dans des unités testables.

L'auto-test est utilisé pour résoudre le problème du test d'un bloc du circuit. Cependant le bloc auto-testé interagit avec le reste du circuit. Nous nous intéressons particulièrement à cette interaction et à son influence sur notre méthode de découpage.

À la différence du *Boundary-Scan*, l'auto-test n'est pas normalisé. Il ne bénéficie pas non plus d'une définition fonctionnelle générale comme dans le cas du *scan-path*. Nous nous appuyons sur deux exemples de formes d'auto-test réellement utilisées pour illustrer l'interaction d'un dispositif d'auto-test avec notre méthode de découpage.

6.1 Présentation générale de l'auto-test

L'auto-test consiste à inclure dans un bloc les moyens de le tester. Nous pouvons distinguer quatre grandes parties dans un bloc auto-testé : la partie fonctionnelle du bloc, la logique de génération des vecteurs, la logique d'analyse des résultats et la logique de contrôle de l'auto-test.

L'utilisation d'une structure d'auto-test se passe en plusieurs étapes :

1. **Une séquence d'initialisation**. Elle permet de mettre dans un état connu la logique de génération des vecteurs et la logique d'analyse des résultats. Elle bascule un dispositif de multiplexage présent sur les entrées du bloc

fonctionnel de manière à ce que la logique d'auto-test en prenne le contrôle. Dans certains cas cette séquence permet d'initialiser le bloc fonctionnel lui-même.

2. **Une séquence de test**. De longueur variable, c'est la séquence durant laquelle les vecteurs générés sont effectivement appliqués aux entrées du bloc fonctionnel, et ses sorties enregistrées au sein du bloc d'analyse des résultats.

3. **Une séquence de collecte**. A l'issue de la séquence de test, les résultats sont extraits de la logique d'analyse vers la logique de contrôle pour être transmis au dehors, puis le bloc est remis dans son environnement fonctionnel.

De nombreuses formes d'auto-test implémentent ces étapes, mais il n'existe pas d'architecture normalisée de l'auto-test. Cependant, comme nous l'avons vu au chapitre 1, les mémoires intégrées dans les circuits sont de plus en plus souvent auto-testées. De même les techniques d'auto-test pseudo-aléatoires sont utilisées pour des blocs de logique irrégulière. Nous allons détailler dans la suite deux implémentations de ces architectures d'auto-test après une présentation des techniques de génération de vecteurs de test et des techniques d'analyse de résultats.

6.1.1 Génération intégrée de vecteurs de test

La génération intégrée de vecteurs consiste à bâtir une architecture capable de produire des vecteurs à destination du bloc à tester. Il existe plusieurs approches dont nous citons ici les deux principales. Dans la première approche, des vecteurs déterministes sont utilisés, et ils sont explicitement implantés dans le silicium. Cette implantation est réalisée à l'aide d'une machine à états, d'une ROM ou d'automates cellulaires. Cette approche permet d'avoir une longueur de test optimisée pour un taux de couverture maximum, mais elle est coûteuse en surface. En pratique la génération déterministe intégrée de vecteurs de test est limitée au blocs importants de logique régulière comme les mémoires.

La deuxième approche consiste à générer des vecteurs de façon pseudo-aléatoire. Pour cela, les propriétés des registres à décalage à rebouclage linéaire (« Linear Feedback Shift Register », ou LFSR), qui sont basées sur la théorie des polynômes primitifs, sont utilisées. L'architecture des LFSR, très simple, permet de générer de façon désordonnée $2^n - 1$ combinaisons d'un mot de n bits, à l'exception de la valeur 0. En pratique, seule une partie de ces $2^n - 1$ combinaisons est réellement appliquée. La figure 6.1 présente un LFSR de 4 bits.

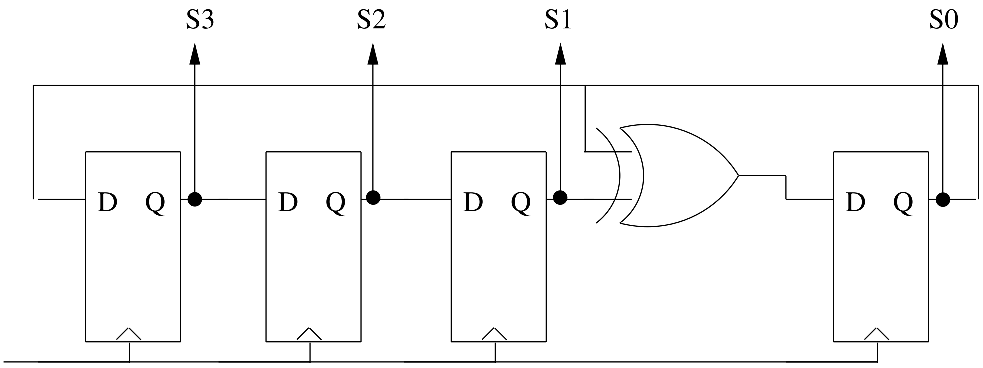


FIG. 6.1 –: Un registre à décalage à rebouclage linéaire (LFSR) de 4 bits

Si ce registre est initialisé avec la valeur 1111, il prend successivement les valeurs 1110, 0111, 1010, 0101, 1011, 1100, 0110, 0011, 1000, 0100, 0010, 0001, 1001, 1101 et de nouveau 1111. Ainsi l'ensemble des combinaisons de 4 bits à l'exception de 0000 est généré de façon pseudo-aléatoire. L'initialisation du registre se fait généralement en incluant celui-ci dans un *scan-path*.

La difficulté est ici de déterminer la valeur d'initialisation du générateur et le nombre de cycles permettant d'obtenir le taux de couverture visé. Cela demande des efforts en terme de simulation de fautes car il n'existe aucune méthode analytique permettant de fixer ces valeurs. Il est aussi possible d'améliorer le taux de couverture en pondérant la génération pseudo-aléatoire, mais ces techniques restent compliquées à mettre en œuvre. En pratique la génération pseudo-aléatoire de vecteurs de test est utilisée pour tester des blocs de logique irrégulière.

6.1.2 Analyse intégrée des résultats

L'analyse intégrée des résultats consiste à vérifier la validité des valeurs obtenues en sortie du bloc auto-testé lors de son test. L'approche qui consiste à stocker explicitement l'ensemble des valeurs attendues n'est réaliste que pour un petit nombre de valeurs. Le plus souvent l'ensemble des valeurs est compacté sur le circuit et le résultat de ce compactage est comparé à l'issue du test avec une valeur de référence calculée par simulation sur le circuit sans faute. Il existe différentes techniques d'analyse par compactage. Les techniques les plus simples sont le comptage de transitions ou le comptage des « uns ». Elles présentent cependant un taux élevé de masquage d'erreurs, c'est à dire de cas où le résultat de l'analyse pour un circuit défectueux est le même que pour un circuit sans faute.

Le meilleur compromis entre simplicité et taux de masquage d'erreurs est atteint à l'aide des propriétés de la division polynômiale. Les registres à décalage à multiples entrées (« Multiple Input Shift Register », ou MISR) ainsi réalisés sont formés sur la base d'un LFSR auquel a été rajouté un étage d'entrées parallèles raccordé aux sorties du circuit sous test. La signature, c'est à dire le contenu du registre à l'issue du test, représente alors le reste de la division polynômiale de la réponse du circuit par le polynôme générateur implémenté dans le LFSR. La probabilité d'obtenir un masquage d'erreurs est la plus faible si un polynôme primitif est utilisé. Un MISR de 4 bits est présenté dans la figure 6.2.

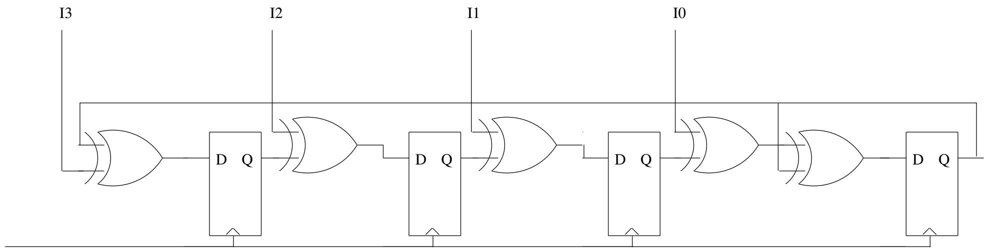


FIG. 6.2 –: Un analyseur de signature à base de LFSR : le MISR

Nous retrouvons ici l'architecture du LFSR de 4 bits à laquelle a été rajoutée quatre ou-exclusifs permettant l'analyse simultanée de 4 signaux. Si ce MISR est initialisé à 0110 et qu'il reçoit une séquence 0011, 1010, 1111 et 0111, alors sa valeur finale sera 0010. Si au troisième cycle le MISR reçoit 1101 et non 1111 à la suite d'une faute dans le circuit, alors la signature à l'issue du test est 0111, et la faute est donc détectée. En pratique la signature est extraite en intégrant le MISR dans un *scan-path*.

6.2 L'auto-test déterministe des mémoires

Il existe de nombreux algorithmes de test de mémoire [dG91]. Ces algorithmes assurent par construction un taux de couverture maximal pour un ensemble de modèles de fautes données. Il n'est pas alors nécessaire d'effectuer une simulation de fautes pour vérifier l'efficacité des tests.

La grande majorité des algorithmes de test de mémoire est basée sur des séquences de type « MARCH ». Un élément de « MARCH » est une séquence d'opérations appliquée à une cellule de la mémoire avant de passer à la suivante. Un test de mémoire est une séquence finie d'éléments de « MARCH ». L'algorithme « MARCHY » [dG91] est composé de quatre éléments de « MARCH » :

1. Tout d'abord chaque cellule de la mémoire est initialisée avec une donnée D en parcourant les adresses dans un ordre quelconque.
2. Puis chaque cellule de la mémoire est relue tout en réécrivant une donnée $\overline{D}$ et en la relisant immédiatement. Les adresses sont parcourus dans un ordre décroissant.

3. Ensuite chaque cellule de la mémoire est relue tout en réécrivant une donnée D et en la relisant immédiatement. Les adresses sont parcourus dans un ordre croissant.

4. Enfin l'ensemble de la mémoire est relue en parcourant les adresses dans un ordre quelconque.

Durant l'exécution de cet algorithme chaque point mémoire subit huit opérations de lecture ou d'écriture. La durée du test est de huit fois le nombre de cycles permettant la lecture ou l'écriture de l'ensemble de la mémoire. Ce test couvre l'ensemble des fautes de collage et de circuit ouvert des cellules mémoires et des fils d'adresses, ainsi que l'ensemble des fautes de transition liées aux cellules mémoires, et certaines fautes de couplage [dG91]. Il n'y a pas de dictionnaire de faute décrivant explicitement les fautes associées à ce test, puisque le taux de couverture est assuré par construction.

Par sa simplicité, cet algorithme est particulièrement adapté à une réalisation matérielle [Aut91a]. La figure 6.3 représente l'architecture générale simplifiée d'une mémoire auto-testée par l'outil BISTRAM Compiler de COMPASS Design Automation. Nous modélisons l'interface de la mémoire par quatre bus : un bus de données en entrée (DIN), un bus de données en sortie (DOUT), un bus d'adresse (ADR) et un bus de contrôle (CTRL). Ces quatre bus se retrouvent sur l'interface du bloc auto-testé. Nous retrouvons à l'intérieur du bloc auto-testé la logique de génération de vecteurs, la logique d'analyse des résultats, la logique de contrôle de l'auto-test ainsi qu'un ensemble de multiplexeurs placés sur les ports d'entrées fonctionnels de la mémoire.

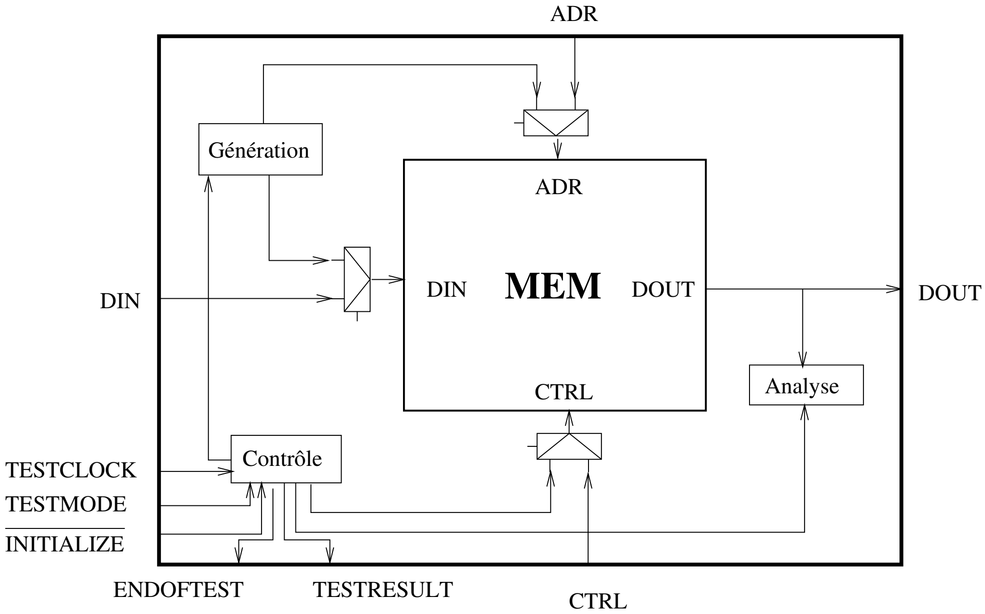


FIG. 6.3 –: Une mémoire auto-testée

Le bloc de génération est composé de deux parties : un compteur ou un registre de type « LFSR » qui permet de contrôler les adresses de la mémoire, et un générateur des valeurs D et $\overline{D}$ qui seront appliquées sur le bus de données.

Le bloc d'analyse est composé de deux comparateurs : l'un avec la valeur D et l'autre avec son inverse.

Le bloc de contrôle est composé d'une machine à états qui coordonne les fonctionnements du bloc de génération, du bloc d'analyse, du bus de contrôle de la mémoire et des multiplexeurs de façon à exécuter l'algorithme d'auto-test. Il permet d'enchaîner les opérations de lecture, d'écriture et de comparaison à D ou $\overline{D}$ composant un élément de « MARCH », puis de passer à l'adresse suivante.

L'interface de l'auto-test avec l'extérieur se fait grâce à cinq signaux :

- $TESTCLOCK$ est l'horloge qui rythme l'auto-test,
- $TESTMODE$ permet de choisir entre le mode d'auto-test et le mode fonctionnel,
- $\overline{INITIALIZE}$ permet de démarrer la séquence d'auto-test,
- $ENDOFTEST$ signale lorsque cette séquence est terminée,
- et $TESTRESULT$ donne le résultat du test.

La séquence d'initialisation consiste à passer en mode auto-test grâce au signal $TESTMODE$, et à initialiser la logique de contrôle et le compteur d'adresses grâce au signal $\overline{INITIALIZE}$. La séquence de test dure $8 * n$ cycles d'horloge $TESTCLOCK$, où n est la taille en mots de la mémoire, et se termine par l'activation du signal $ENDOFTEST$. La séquence de collecte consiste alors simplement à lire la valeur du signal $TESTRESULT$ pour déterminer le résultat du test.

6.3 Auto-test pseudo-aléatoire

La figure 6.4 décrit le principe d'un bloc de logique auto-testé.

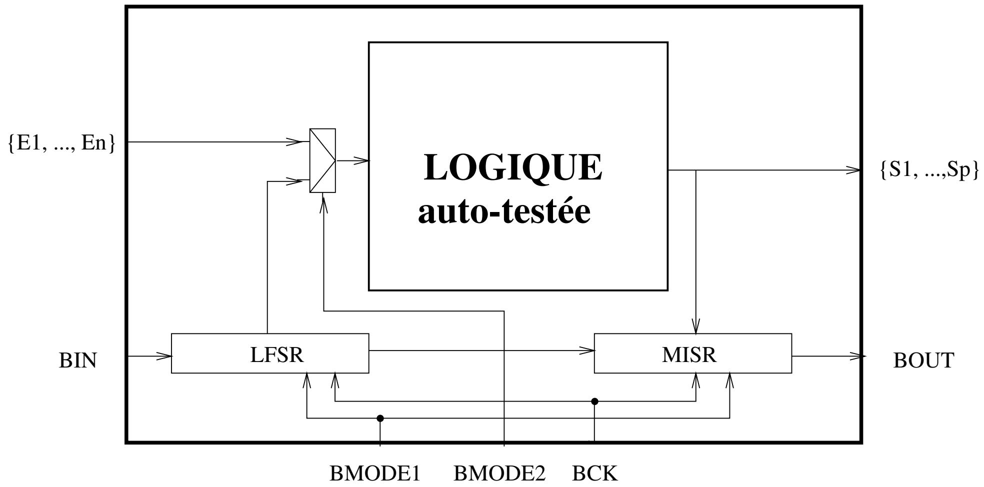


FIG. 6.4 –: Auto-test pseudo-aléatoire

Dans cette architecture, un LFSR de n bits est branché sur les n entrées fonctionnelles du bloc de logique via un multiplexeur commandé par le signal externe BMODE2. Simultanément, un MISR de p bits observe les p sorties fonctionnelles du bloc. Les deux registres LFSR et MISR sont initialisables à travers un mode de *scan-path* ayant comme entrée série BIN et comme sortie série BOUT. Le passage du mode *scan-path* au mode natif du LFSR et du MISR se fait grâce au signal BMODE1. Enfin, le décalage des registres est assuré par l'horloge dédiée BCK.

La séquence d'initialisation consiste à charger par *scan-path* les registres LFSR et MISR avec une valeur de départ. Puis le signal BMODE2 permet aux sorties du LFSR d'être directement branchées sur le bloc de logique. Le nombre de cycles d'horloge BCK de la séquence de test a été déterminé auparavant par une simulation de fautes de façon à obtenir un taux de couverture satisfaisant. Enfin la séquence de collecte consiste à lire par le *scan-path* le contenu du MISR. Cette signature est alors comparée à une valeur de référence obtenue par simulation logique du circuit sans faute.

Pour obtenir un dictionnaire de fautes associé au test du bloc, il faut effectuer une simulation de fautes de la séquence de test avec un modèle de fautes adapté à la nature du bloc, par exemple le modèle des collages uniques. Si cette simulation est trop coûteuse, ou si le test est exhaustif, il est possible d'associer directement au test un taux de couverture estimé, sans décrire explicitement un dictionnaire de fautes.

Ce type d'architecture est couramment utilisé pour le test intégré de bloc de logique irrégulière ou pour le test intégré de ROM. Dans l'outil BISTROM Compiler [Aut91b] de COMPASS Design Automation par exemple, un LFSR modifié capable de couvrir l'ensemble des 2^n valeurs contrôle les adresses de la ROM, et un MISR observe le bus de données. L'ensemble des adresses de la ROM étant vérifié, l'outil considère qu'elle est complétement testée, et ne fournit donc pas de dictionnaire de fautes.

6.4 Interaction avec la méthode de découpage

Nous avons vu au cours du chapitre 2 que notre méthode de découpage s'appuie sur les notions de signaux OBSERVABLES et CONTRÔLABLES. Ces signaux représentent les frontières des unités testables.

Bien qu'ils contrôlent le bloc à tester, les sorties du générateur intégré de vecteurs de test ne sont pas des signaux CONTRÔLABLES. Il n'est en effet pas possible de leur affecter n'importe quelle valeur, par exemple la valeur 0...0 pour un LFSR ou bien une valeur différente de D ou $\overline{D}$ pour l'architecture d'auto-test de la mémoire que nous avons présenté plus haut. Ces signaux sont donc appelés BIST_CONTRÔLÉS.

De même les signaux d'entrée de l'analyseur intégré, qui sont aussi les sorties du bloc testé des résultats, ne sont pas OBSERVABLES mais simplement BIST_OBSERVÉS.

Une unité auto-testée (UAT) ne possède que des sorties BIST_OBSERVÉES et des entrées BIST_CONTRÔLÉES. Elle est fabriquée de la même façon que les unités testables élémentaires (voir page 73) en parcourant en arrière le réseau booléen sous contraintes du mode d'exploitation de l'auto-test, depuis tous les nœuds BIST_OBSERVÉS jusqu'aux nœuds BIST_CONTRÔLÉS. Une unité auto-testée représente un ensemble de nœuds qui sont testés par une architecture d'auto-test. Il n'est donc pas nécessaire de générer des vecteurs de test pour une unité auto-testée.

Puisqu'aucun nœud d'une unité auto-testée n'est CONTRÔLABLE ou OBSERVABLE, il n'existe pas de points d'arrêt dans les blocs auto-testés du point de vue de l'algorithme de découpage en unités testables élémentaires. Lorsque ce bloc est inséré dans un circuit qui possède des signaux OBSERVABLES en aval du bloc et des signaux CONTRÔLABLES en amont, l'ensemble de l'unité auto-testée est alors intégrée dans une ou plusieurs unités testables à l'issue du découpage. La figure 6.5 illustre ce problème. Ici, trois blocs de logique sont connectés ensemble. Le bloc AMONT possède des entrées contrôlables, le bloc B est équipé d'un auto-test et le bloc AVAL possède des sorties observables. Ainsi le bloc B possède son propre test et n'a pas besoin d'être inclus dans une unité testable.

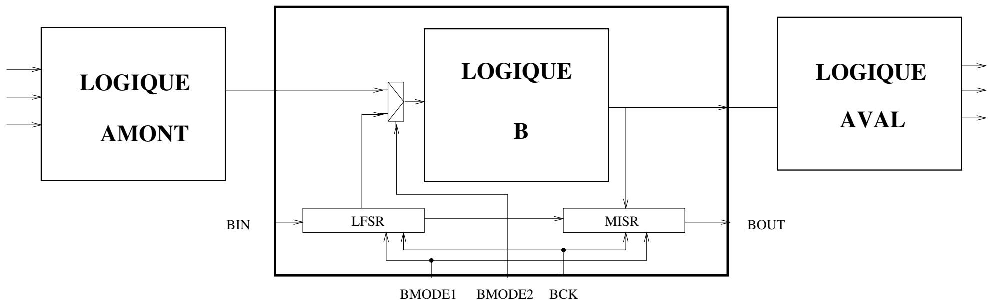


FIG. 6.5 –: L'influence de l'auto-test sur le découpage en unité testable

Pourtant, aucun nœud CONTRÔLABLE ne vient stopper le parcours arrière depuis les sorties du bloc AVAL à travers le bloc B. Ainsi notre méthode de découpage va inclure dans ses unités testables des portes issues d'un bloc déjà testé. Cela va à la fois compliquer la tâche de l'outil de génération, allonger la longueur du test et ne rien améliorer du point de vue de la couverture.

Pour éviter cela, il faut pouvoir observer les entrées des blocs auto-testés et contrôler leurs sorties. Cela suppose l'existence de mécanismes matériels qui permettent d'isoler les blocs auto-testés pendant le test du reste du circuit. Si l'ajout de matériel est impossible, comme c'est souvent le cas lors de l'utilisation de macro-cellules, l'effort de génération des vecteurs de test peut néammoins être réduit en utilisant un dictionnaire de fautes initial contenant toutes les fautes détectées par le mécanisme d'auto-test.

Un autre problème est posé par l'intégration du dictionnaire de fautes et du taux de couverture du bloc auto-testé ou de la macro-cellule avec le dictionnaire et le taux de couverture du circuit où il est présent. Pour assurer une homogéneïté de représentation avec les dictionnaires de fautes des unités testables, nous devons obtenir un dictionnaire de fautes associé à l'unité auto-testée qui se base sur la description de son réseau booléen orienté.

6.4.1 Isolement intégré de blocs auto-testés

L'isolement du bloc auto-testé lors du test du reste du circuit implique l'ajout de matériel au sein du circuit. Si la méthode de conception le permet, nous allons privilégier la modification du mécanisme d'auto-test du bloc pour qu'il permette le test des blocs voisins. Ainsi la figure 6.6 propose le schéma modifié d'un registre à décalage à rebouclage linéaire pour qu'il accepte un mode de chargement parallèle. Cette approche est inspirée du principe du BILBO (*Built-In Logic Block Observer*) qui permet le test pseudo-aléatoire et parallèle de plusieurs réseaux logiques combinatoires successifs [KMZ79] [WP83].

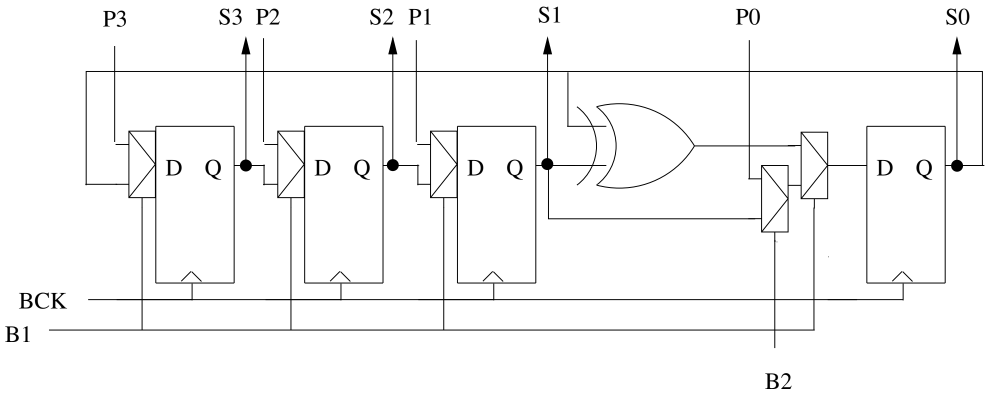


FIG. 6.6 –: Un LFSR scannable avec chargement parallèle

Le LFSR possède donc trois modes de fonctionnement : le signal B1 permet de choisir entre un mode de chargement parallèle et un mode de décalage, et dans cette configuration le signal B2 permet de choisir entre un décalage de type *scan-path* et la génération de vecteurs pseudo-aléatoires. Ce LFSR modifié permet donc à la fois de contrôler quatre bits en mode génération, et d'observer quatre bits en mode de chargement parallèle. La structure originale du BILBO possède un quatrième mode d'analyse intégrée des résultats, qui n'est pas utile ici. La figure 6.7 donne un exemple d'utilisation de ce LFSR modifié. Les entrées $\{E_1, ..., E_n\}$ du bloc auto-testé sont donc maintenant observables. Un multiplexeur en amont des sorties $\{S_1, ..., S_p\}$ permet en outre de contrôler les sorties avec les bascules du MISR.

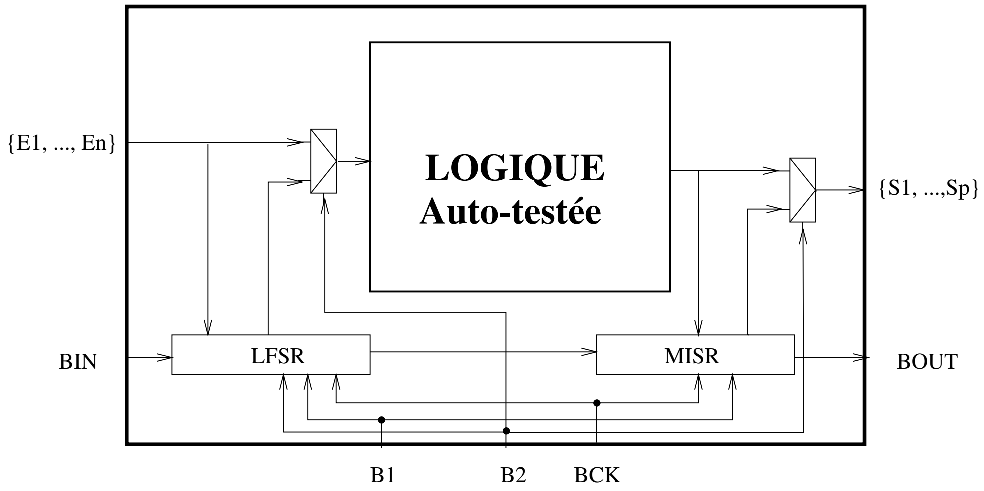


FIG. 6.7 –: Un bloc auto-testé contrôlant ses sorties et observant ses entrées

Ce bloc auto-testé s'insère parfaitement dans notre méthode de découpage :

- les sorties des bascules du MISR sont maintenant des entrées contrôlables d'unités testables,
- les entrées parallèles du LFSR sont maintenant des sorties observables d'unités testables,
- et aucune porte du bloc auto-testé n'est incluse dans une unité testable.

En effet, le parcours arrière qui permet de construire les unités testables élémentaires s'arrête sur les sorties contrôlables du MISR, et d'autres unités testables ont pour sorties les entrées observables du LFSR. En d'autres termes, l'intersection entre l'unité auto-testée et les unités testables est vide.

6.4.2 Isolement périphérique de blocs auto-testés

Si la modification du bloc auto-testé n'est pas possible, soit parce qu'il n'est disponible que sous la forme d'une macro-cellule, soit parce que son architecture n'est pas adaptée, l'ajout de matériel en périphérie du bloc est une solution envisageable. Un exemple de ce type d'architecture est présenté dans la figure 6.8.

Dans ce cas les entrées du bloc sont observées en amont de celui-ci à l'aide de registres d'observation (REGO) : ce sont des bascules appartenant à un *scan-path* dont les entrées fonctionnelles sont branchées aux entrées du bloc. Les sorties du bloc sont contrôlées en aval de celui-ci à l'aide de registres de contrôle (REGC) : ce sont des bascules appartenant à un *scan-path* dont les sorties parallèles sont multiplexées avec les sorties du bloc. Il est même possible selon le type de test visé d'utiliser une même bascule pour contrôler une sortie et observer une entrée.

FIG. 6.8 –: Contrôle et observation périphérique d'un bloc auto-testé

Le signal pilotant le multiplexeur, qui est placé en sortie du bloc, est issu de la logique de contrôle de test du circuit.

De la même façon que pour les blocs auto-testés qui possèdent des mécanismes internes d'isolation, les registres de contrôle permettent ici de stopper le parcours arrière de construction des unités testables élémentaires, et les registres d'observation sont des sorties d'autres unités testables élémentaires. L'intersection entre les unités testables et l'unité auto-testée est donc vide. Ainsi, ce bloc s'insère à moindre coût dans notre méthode de découpage.

Des techniques plus complexes d'isolement du bloc auto-testé s'inspirant du *Boundary-Scan* [IEE90] peuvent aussi être utilisées du moment qu'elles respectent les principes d'observation en amont du bloc, et de contrôle en aval. Par leur caractère extérieur, ces techniques sont applicables à n'importe quel type de bloc.

6.4.3 Génération de vecteurs de test avec un dictionnaire initial

Si aucune des solutions évoquées ci-dessus n'est réalisable, alors tout ou partie du bloc auto-testé est inclus dans des unités testables. Une manière de diminuer l'effort de génération consiste à écrire un dictionnaire de fautes initial associé à chaque unité testable ayant une intersection non vide avec l'unité auto-testée. Dans ce dictionnaire, toutes les fautes présentes sur un nœud appartenant à l'unité auto-testée sont marquées « détectées ». Ainsi l'outil de génération de vecteurs ne va pas tenter de trouver des séquences pour ces fautes, bien qu'il soit obligé de considérer toutes les portes fonctionnelles du bloc auto-testé pour générer ses vecteurs.

Si nous reprenons l'exemple de la page 93, l'unité testable est formée des blocs AMONT, B et AVAL. Dans le dictionnaire de fautes initial associé à cette unité testable, toutes les fautes appartenant au bloc B sont marquées comme détectées. L'outil de génération automatique de vecteurs de test n'essayera donc de générer des vecteurs que pour des fautes présentes sur les portes des blocs AMONT et AVAL.

6.4.4 Construction d'un dictionnaire de fautes homogène

Le dictionnaire de fautes de l'unité auto-testée est fabriqué à partir du réseau booléen orienté qui la représente, en marquant toutes les fautes détectées. Ce dictionnaire de fautes, artificiel, mais homogène avec les dictionnaires de fautes des unités testables, est destiné à la phase d'assemblage des tests. Il ne doit cependant pas occulter le véritable dictionnaire de fautes éventuellement fourni avec le bloc auto-testé, et qui contient les véritables informations de couverture. Celui-ci doit donc être conservé.

Nous devons connaître la composition des unités auto-testées pour élaborer ce dictionnaire de fautes. Les unités auto-testées sont construites à partir des nœuds BIST_OBSERVÉS et BIST_CONTRÔLÉS à l'aide de la procédure de marquage arrière présentée page 73. Nous devons donc identifier ces nœuds comme nous avons identifié les signaux observables et contrôlables.

Lorsque l'auto-test est bâti à l'aide de registres à décalage de type LFSR ou MISR, nous pouvons utiliser des techniques similaires à la reconnaissance automatique de signaux observables et contrôlables présentées au chapitre 4. Nous identifions les points mémoires de ces registres en nous basant sur leur fonctionnalité, et nous en déduisons les signaux BIST_OBSERVÉS et BIST_CONTRÔLÉS.

Pour les architectures qui ne sont pas basées sur des registres à décalage comme l'auto-test des mémoires, il n'existe aucune méthode générale pour identifier automatiquement les nœuds BIST_OBSERVÉS et BIST_CONTRÔLÉS car la fonctionnalité des blocs de génération et d'analyse n'est pas normalisée. Il faut alors que ces signaux soient explicitement décrits. C'est en pratique possible car un bloc auto-testé correspond généralement à un bloc fonctionnel dans un circuit, dont l'interface est alors connue.

6.5 Conclusion

La présence de blocs auto-testés ou de macro-cellules dans un circuit n'est pas incompatible avec notre méthode de découpage. Lorsque ces blocs possèdent des frontières contrôlables et observables, l'intégration se fait à moindre coût. Plusieurs solutions matérielles et logicielles sont envisageables dans le cas contraire.

Elles ont pour but de minimiser l'effort de test nécessaire à la prise en compte de ces blocs dans notre méthode de test par découpage en unités testables.

Chapitre 7

Assemblage de test

Nous avons obtenu pour chaque unité testable un fichier de vecteurs et un dictionnaire de fautes grâce à un outil de génération de vecteurs de test.

Ces vecteurs cependant ne sont pas applicables directement sur le circuit. En effet ils concernent l'interface des unités testables et non l'interface du circuit. Il faut donc traduire ces vecteurs. Puis il faut les assembler entre eux et rajouter les vecteurs qui activent les blocs auto-testés du circuit. Le but est de produire une séquence unique testant le circuit, soit à travers son interface, soit à travers un accès de type *Boundary-Scan*. La première partie de ce chapitre est consacrée à la production de cette séquence et la seconde à l'exploitation d'un éventuel *Boundary-Scan*. Dans la troisième partie du chapitre nous montrons comment calculer un taux de couverture global à partir des tests de chaque unité testable et de chaque bloc auto-testé.

7.1 Assemblage des vecteurs d'unités testables

Nous allons dans cette partie nous intéresser à l'assemblage optimisé des tests provenant de chaque unité testable. Dans un premier temps, nous montrons comment les vecteurs concernant une unité testable sont traduits pour être appliqués depuis l'interface du circuit. Puis nous proposons une stratégie pour tester en parallèle des unités testables disjointes, puis des unités testables peu recouvrantes. Ces opérations ont toutes pour but de minimiser la durée et le volume du test final.

Nous faisons l'hypothèse dans cette partie que l'ensemble des signaux observables et contrôlables appartient à un *scan-path* unique. Le cas de plusieurs chemins de *scan-path* indépendants est abordé à la fin de cette section.

Le problème de l'assemblage optimal de vecteurs de test appliqués à travers un *scan-path* a souvent été traité [JPP89] [OBT91] [LKL93]. Il a en général pour but de trouver un ordonnancement des points mémoires dans le chemin de *scan-path* qui permet une longueur de test minimale. Notre problème est différent puisqu'ici le chemin est déjà constitué.

7.1.1 Traduction de vecteurs

Pour chaque unité testable nous disposons d'une séquence de vecteurs de test. Cette séquence décrit les valeurs à appliquer aux entrées de l'unité testable ainsi que les valeurs attendues sur les sorties de l'unité testable. L'interface de l'unité testable est constituée des signaux observables et contrôlables du circuit. Ces signaux ne sont pas tous sur l'interface du circuit. Certains appartiennent à un registre de *scan-path*. Il faut donc traduire la séquence de test de l'unité testable pour qu'elle devienne applicable depuis l'interface du circuit. Nous disposons pour cela des informations suivantes :

- les séquences permettant de contrôler le registre de *scan-path*, telles qu'elles ont été décrites au chapitre 4,
- la description du registre de *scan-path* obtenue après la phase de reconnaissance,
- et la correspondance, obtenue lors du découpage, entre l'interface de l'unité testable et les signaux du circuit.

Ainsi, à chaque connecteur C_i de l'unité testable est associé un signal S_i du circuit. Ce signal est, soit un connecteur du circuit, soit un signal du *scan-path*. Dans ce cas nous connaisons le nombre d'inversions $INV(S_i)$ que subit une valeur entre l'entrée série du registre de *scan-path* et le signal S_i pour un signal contrôlable, ou entre S_i et la sortie série du registre de *scan-path* pour un signal observable. Nous connaissons aussi le rang $RANG(S_i)$ du signal S_i, c'est à dire la position du registre correspondant à S_i dans le registre de *scan-path*. Si la longueur du registre de *scan-path* est appelée $RANG_{max}$, alors $RANG(S_i)$ est compris entre 1 et $RANG_{max}$.

Si le signal S_i est un connecteur du circuit, nous considérons $INV(S_i) = 0$ et $RANG(S_i) = 0$.

Nous définissons $Rcont_{max}$ comme le rang du connecteur d'entrée de l'unité testable le plus éloigné de l'entrée série du registre de *scan-path*. De même, $Robs_{min}$ est le rang du connecteur de sortie de l'unité testable le plus éloigné de la sortie série du registre de *scan-path*, excepté les connecteurs qui correspondent à un connecteur du circuit.

$Rcont_{max}$ représente ainsi le nombre minimum de décalages qu'il faut effectuer sur le registre de *scan-path* pour contrôler l'unité testable. Nous notons cette valeur D_c.

Le nombre minimum de décalages qu'il faut effectuer sur le registre de *scan-path* pour observer l'unité testable est égal à $RANGmax - Robs_{min}$, c'est à dire la longueur du registre moins le rang du signal à observer le plus éloigné de la sortie série du registre de *scan-path*. Nous notons cette valeur D_o

La figure 7.1 illustre ces différentes valeurs sur l'exemple d'une unité testable contrôlée et observée par un *scan-path*.

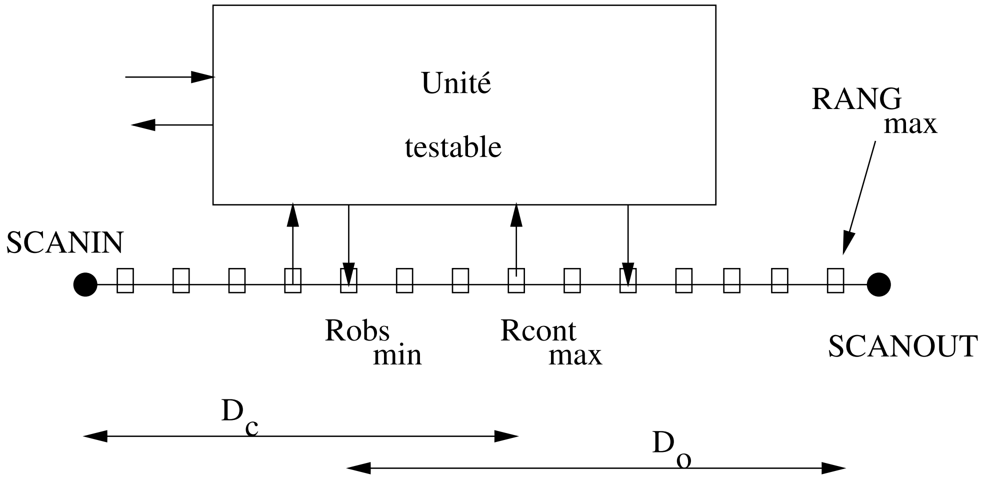


FIG. 7.1 –: Valeurs D_c et D_o pour une unité testable

Le nombre minimum de décalages à effectuer sur le registre de *scan-path* pour contrôler et observer l'unité testable à travers le *scan-path* est notée D_{co}. Nous avons :

$$D_{co} = MAX(D_c, D_o).$$

Pour chaque vecteur de la séquence de test de l'unité testable, les décalages du registre de *scan-path*, servant à contrôler les valeurs du vecteur, servent aussi à observer les réponses du vecteur précédent. Il faut donc effectuer D_{co} décalages. La figure 7.2 décrit précisément la séquence permettant d'appliquer un vecteur sur l'unité testable tout en vérifiant le résultat du précédent.

```
POUR j := 0 A D_co
        SI il existe un connecteur d'entrée de l'UT tel que
        RANG(C_i) = Rcont_max - j ALORS
            Appliquer sur l'entrée série du scan-path la valeur
            de C_i au vecteur courant, inversée INV(S_i) fois
        FIN SI

        Appliquer la séquence de décalage scan-path

        SI il existe un connecteur de sortie de l'UT tel que
        RANG(C_i) = Robs_min + j ALORS
            Vérifier sur la sortie série du scan-path la valeur
            de C_i au vecteur précédent, inversée INV(S_i) fois
        FIN SI
FIN POUR

POUR chaque connecteur d'entrée de l'UT tel que RANG(C_i) = 0
        Appliquer la valeur de C_i au vecteur courant
        sur le connecteur du circuit correspondant
FIN POUR

Appliquer la séquence de chargement parallèle du scan-path

POUR chaque connecteur de sortie de l'UT tel que RANG(C_i) = 0
        Vérifier la valeur de C_i au vecteur précédent
        sur le connecteur du circuit correspondant
FIN POUR
```

FIG. 7.2 –: Application d'un vecteur d'UT à travers un *scan-path*

Pour le premier vecteur de la séquence seuls D_c décalages sont requis puisqu'il n'y a pas de valeur à observer. De même seuls D_o décalages sont requis pour observer le dernier vecteur puisqu'il n'y a pas de valeur à contrôler. Le nombre total de décalages nécessaires à l'application des n vecteurs d'une unité testable est donc :

$$D_{total} = D_c + (n-1) \times D_{co} + D_o$$

Si un décalage représente $C_{décalage}$ cycles et que le chargement parallèle du registre de *scan-path* dure $C_{chargement}$ cycles, le nombre de cycles total engendré par l'application de la séquence sur l'unité testable depuis l'interface du circuit est donc :

$C_{total} = C_{décalage} \times D_{total} + n \times C_{chargement}$

En général, $C_{chargement} = 1$ et $C_{décalage} = 1$. De plus, si n est grand, nous pouvons négliger la différence entre D_c, D_o et D_{co}. Nous pouvons donc approximer le coût du test de n vecteurs d'une unité testable en nombre de cycles :

$C_{total} \simeq n \times (D_{co} + 1)$

7.1.2 Parallélisation du test

La méthode de traduction de vecteurs de test présentée précédemment peut être appliquée pour chaque unité testable. Cependant, notre méthode de découpage produit des unités testables peu recouvrantes. Il est donc possible de tester en parallèle certaines unités testables, ce qui a pour effet de faire baisser la longueur de la séquence finale.

Deux unités testables peuvent être testées en parallèle si elle ne possèdent aucune entrée en commun. En effet, il est alors possible de les contrôler toutes les deux à l'aide de la même séquence de décalage. Et, puisque par construction deux unités testables ne partagent aucune sortie, il est aussi possible de les observer toutes les deux à l'aide de la même séquence de décalage.

Nous avons vu au chapître 5 qu'une unité testable est caractérisée par ses entrées. Entre autre, la notion de taux de recouvrement est basée sur les entrées des unités testables. Nous allons donc réutiliser cette mesure pour décider si deux unités testables peuvent être testées simultanément.

Deux unités testables possédant un taux de recouvrement nul n'ont aucune entrée en commun et peuvent être testées en parallèle. Deux unités testables possédant un faible taux de recouvrement n'ont que quelques entrées en commun. Nous essayons alors de les tester simultanément.

Unités testables non recouvrantes

Nous disposons de deux unités testables possédant un taux de recouvrement nul et de deux séquences de vecteurs, l'une de longueur n et l'autre de longueur p. Nous supposons dans la suite que n est supérieur ou égal à p. Nous appelons UT_1 l'unité testable associée aux n vecteurs et UT_2 l'unité testable associée aux p vecteurs. Le coût engendré par le test séparé de ces deux unités testables est égal à la somme des coûts engendrés par le test de chaque unité testable :

$C_{total} = C^1_{total} + C^2_{total} = n \times (D^1_{co} + 1) + p \times (D^2_{co} + 1)$

Pour contrôler les deux unités testables avec le même vecteur, il faut faire un nombre de décalage $Rcont^{\cup}_{max}$ tel que :

$$Rcont^{\cup}_{max} = MAX(Rcont^{1}_{max}, Rcont^{2}_{max})$$

De même, le nombre de décalages nécessaires pour observer les deux unités testables est $RANG_{max} - Robs^{\cup}_{min}$ avec

$$Robs^{\cup}_{min} = MIN(Robs^{1}_{min}, Robs^{2}_{min})$$

Il est ainsi possible de calculer le nombre de décalages nécessaires au contrôle et à l'observation d'un vecteur sur deux unités testables non recouvrantes :

$$D^{\cup}_{co} = MAX(Rcont^{\cup}_{max}, RANG_{max} - Robs^{\cup}_{min}) = MAX(D^{1}_{co}, D^{2}_{co}).$$

Si les deux unités testables sont testées en parallèle, alors p vecteurs sont appliqués sur les deux unités testables. Puis les $(n-p)$ vecteurs sont appliqués à l'unité testable UT_1. Le coût engendré par le test en parallèle des deux unités testables non recouvrantes est donc :

$$C^{\cup}_{total} = p \times (D^{\cup}_{co} + 1) + (n-p) \times (D^{1}_{co} + 1) = p \times D^{\cup}_{co} + (n-p) \times D^{1}_{co} + n$$

Calculons le gain obtenu en testant deux unités testables en parallèle. Il est égal à :

$$Gain = C_{total} - C^{\cup}_{total}$$

Après simplification nous obtenons :

$$Gain = p \times (D^{1}_{co} + D^{2}_{co} - D^{\cup}_{co} + 1)$$

Puisque $D^{\cup}_{co} = MAX(D^{1}_{co}, D^{2}_{co})$, ce gain est toujours positif. Nous voyons donc que tester en parallèle deux unités testables permet de réduire la longueur du test d'environ $(p \times (D_{co} + 1))$ cycles, où p est le nombre de vecteurs de la séquence la plus courte et D_{co} le plus petit des nombres de décalages nécessaires au contrôle et à l'observation d'une des deux unités testables.

Pour choisir quelles unités testables vont être testées simultanément nous utilisons une approche « gloutonne » similaire à notre algorithme de regroupement des unités testables : une unité testable est testée en parallèle avec la première trouvée pour laquelle le taux de recouvrement est nul. Cela nous permet, entre autre, de faire des regroupements multiples en une seule passe. Cet algorithme est présenté dans la figure 7.3.

```
UT_1, ... , UT_n sont les n unités testables
POUR courante := 1 A n :
        POUR candidate = (courante + 1) à n
                calculer TR = TR(UT_courante,UT_candidate)
                SI (TR = 0) ALORS
                   Regrouper les vecteurs de UT_candidate et UT_courante
                   ALLER A la courante suivante
                FIN SI
        FIN POUR
FIN POUR
```

FIG. 7.3 –: Algorithme de regroupement des vecteurs des UT non recouvrantes

La procédure de regroupement des vecteurs des deux unités testables consiste à transformer les deux unités testables UT_1 et UT_2 et leurs vecteurs V_1, de longueur n et V_2, de longueur p, en deux nouvelles unités testables UT_3 et UT_4 (nous supposons $n > p$) :

1. $UT_3 = UT_1 \cup UT_2$
2. $UT_4 = UT_1$
3. V_3 = Union des n premiers vecteurs de UT_1 et UT_2
4. V_4 = les vecteurs restants de UT_1.

Cette méthode permet ensuite d'appliquer la procédure de traduction présentée page 102 à chacune des deux nouvelles unités testables pour former la séquence de test.

Unités testables peu recouvrantes

Si deux unités testables possèdent un taux de recouvrement faible, cela signifie qu'elles ont quelques entrées en commun. Nous pouvons alors essayer de les tester en parallèle. La seule condition requise est que les deux unités testables soient purement combinatoires. En effet, nous allons procéder à des permutations de vecteurs au sein des séquences de test, ce qui n'est pas possible avec des unités testables séquentielles.

Nous disposons de deux unités testables UT_1 et UT_2 et de deux séquences de vecteurs, l'une de longueur n et l'autre de longueur p. Nous supposons dans la suite que n est supérieur ou égal à p. Il existe q vecteurs pour lesquels les valeurs présentes sur les entrées communes sont les mêmes pour les deux séquences.

Le coût d'un test séparé des deux unité testables est :

$$C_{total} = C^1_{total} + C^2_{total} = n \times (D^1_{co} + 1) + p \times (D^2_{co} + 1)$$

Pour tester les deux unités testables en parallèle, nous devons appliquer les q vecteurs pour lesquels les valeurs sont communes sur l'union des deux unités testables, puis les $(n - q)$ vecteurs restants sur UT_1, et enfin les $(p - q)$ vecteurs restants sur UT_2. Le coût de ce test est donc :

$$C^{\cup}_{total} = q \times (D^{\cup}_{co} + 1) + (n - q) \times (D^1_{co} + 1) + (p - q) \times (D^2_{co} + 1)$$

De la même façon que plus haut, nous pouvons exprimer le gain à tester deux unités testables recouvrantes en parallèle :

$$Gain = q \times (D^1_{co} + D^2_{co} - D^{\cup}_{co} + 1)$$

Ce gain est toujours positif, quel que soit le taux de recouvrement entre les deux unités testables, même si q est petit. Cependant, la probabilité de trouver dans les deux séquences des valeurs identiques pour les entrées en commun devient très faible lorsque ce nombre d'entrées en commun est important. Nous n'appliquons donc ce regroupement qu'aux unités testables peu recouvrantes, c'est à dire possédant un faible taux de recouvrement.

La procédure utilisée pour décider le regroupement d'unités testables est basée sur le même principe que pour les unités testables non recouvrantes. La décision de regrouper est déclenchée par un seuil de recouvrement TR_{MAX}, et seulement si les deux unités testables sont combinatoires. L'algorithme proposé est présenté dans la figure 7.4.

```
UT_1, ... , UT_n sont les n unités testables
POUR courante := 1 A n :
        POUR candidate = (courante + 1) à n
                SI UT_courante et UT_candidate sont combinatoires ALORS
                    calculer TR = TR(UT_courante,UT_candidate)
                    SI (TR <= TR_MAX) ALORS
                        Regrouper les vecteurs de UT_candidate et UT_courante
                        ALLER A la courante suivante
                    FIN SI
                FIN SI
        FIN POUR
FIN POUR
```

FIG. 7.4 –: Algorithme de regroupement des vecteurs des UT recouvrantes avec un seuil TR_{MAX}

De la même façon que précédemment la procédure de regroupement crée trois nouvelles unités testables et leurs vecteurs associés. La première possède q vecteurs et est formée de l'union de UT_1 et UT_2. La deuxième est UT_1 avec $n - q$ vecteurs, et la troisième est UT_2 avec $p - q$ vecteurs. La procédure de traduction des vecteurs, définie plus haut, leur est ensuite appliquée.

7.1.3 Multiples *scan-path*

Pour réduire la durée d'application des tests dans un circuit, il est fréquent de segmenter le chemin de *scan-path* en plusieurs chemins indépendants plus courts.

Le coût de l'application de n vecteurs sur une unité testable contrôlée et observée à travers s chemins de *scan-path* dépend de la façon dont sont contrôlés ces *scan-path*. Si ceux-ci sont indépendants, c'est à dire qu'il est possible de les décaler simultanément, le coût d'application est :

$C_{total} = n \times MAX_{i \in s}(D^i_{co}) + n$

D^i_{co} représente le nombre minimum de décalages à effectuer sur le i^{e} chemin de *scan-path* pour contrôler et observer la partie de l'unité testable relative à ce *scan-path*.

Si par contre, les chemins de *scan-path* sont dépendants, par exemple parce qu'ils possèdent tous la même entrée série, alors le coût est de :

$C_{total} = n \times \sum_{i \in s}(D^i_{co}) + n$

C'est le cas par exemple lorsque les chemins de *scan-path* sont contrôlés par une interface *Boundary-Scan*.

Quelle que soit la façon de contrôler les *scan-path*, l'ensemble des manipulations sur les unités testables reste donc applicable avec plusieurs chemins de *scan-path*.

7.1.4 Efficacité de la parallélisation

Pour illustrer les gains obtenus par la parallélisation du test des unités testables, nous avons utilisé le circuit ISCAS s15850 équipé d'un *scan-path* complet. Le *scan-path* est ordonné de façon aléatoire, sans chercher à l'optimiser. Ce circuit possède 10306 portes, dont 534 bascules. Nous avons découpé ce circuit en 81 unités testables en utilisant un seuil TR_{min} de 50. Le facteur d'augmentation obtenu est de 1.03. Il s'agit donc un découpage peu recouvrant. Nous nous intéressons ici uniquement à l'assemblage des tests sur ce découpage.

Nous avons généré pour chaque unité testable des vecteurs de test. Nous avons obtenu un total de 1057 vecteurs combinatoires atteignant un taux de couverture de 98%. Si nous considérons que chaque unité testable utilise entièrement le *scan-path*, c'est à dire que D_{co} est maximum, le coût théorique du test est :

$$C_{total} = n \times (D_{co} + 1) = 1058 \times 535 = 566\ 030 \text{ cycles}$$

L'application effective des vecteurs, unité testable par unité testable, se fait à l'aide de 241 095 cycles. En effet, le D_{co} moyen est de 357 cycles, et non 534, et quelques unités testables n'utilisent pas le *scan-path* ($D_{co} = 0$). Dans le tableau 7.1, nous présentons le nombre de cycles obtenus à l'issue de la procédure de test parallèle d'unités testables pour différents seuils TR_{MAX} en comparaison au test non parallèle. Dans chacun des cas, le taux de couverture est strictement identique puisque les vecteurs originaux sont les mêmes.

Lorsque TR_{MAX} est égal à zéro, seules les unités testables indépendantes sont testées en parallèle. Nous obtenons pour cela environ 25 000 cycles de moins par rapport à un test non parallèle, soit un gain d'environ 10%. Le meilleur gain est obtenu avec un seuil TR_{MIN} de 5 où le test est 14% plus court que le test non parallèle. La procédure de test parallèle d'unités testables peu recouvrantes permet donc de compenser un peu le recouvrement.

Si par contre, nous choisissons un seuil trop élevé, les gains sont très faibles. Il devient en effet difficile de trouver des vecteurs communs à deux unités testables trop recouvrantes.

TR_{MAX}	Nombre de cycles	Gain
—	241 095	—
0	215 337	10%
5	207 803	14%
10	208 898	13%
20	214 462	11%
30	212 493	12%
40	241 093	0%
100	236 863	2%

TAB. 7.1 –: *Longueur de test pour différents seuils* TR_{MAX} *pour le circuit s15850s*

7.1.5 Interaction du découpage sur la longueur de test

Nous avons comparé les résultats obtenus par un découpage aux résultats obtenus avec l'outil commercial *HITEST* seul. Ce générateur automatique de vecteurs de test permet le traitement automatique de circuits équipés de *scan-path* complets [Sys91]. Dans ce cas, nous fournissons la description des cellules formant le *scan-path* et la séquence permettant son chargement et son déchargement pour qu'*HITEST* puisse générer directement des vecteurs. Pour les deux approches le même générateur de vecteurs de test est utilisé, soit directement, soit à travers notre découpage, où il est appliqué successivement à chaque unité testable combinatoire.

En utilisant un seuil TR_{MIN} négatif, nous forçons un découpage en une seule unité testable rassemblant l'ensemble des unités testables élémentaires. Nous présentons dans le tableau suivant les résultats obtenus, pour plusieurs découpages du circuit s15850, comparés à l'outil `HITEST` seul. Le nombre d'unités testables élémentaires pour ce circuit est de 660. La première colonne montre le nombre d'unités testables résultant du regroupement, la deuxième colonne le facteur d'augmentation qui en résulte. Dans les troisième et quatrième colonnes, nous donnons le temps consacré (en secondes) à la génération des vecteurs par `HITEST` sur une machine seule et sur deux machines en parallèle. Ces temps de génération ont été mesurés avec la commande unix `time` sur une SPARC 10 possédant 400 Mo de mémoire RAM. Enfin, les deux dernières colonnes donnent le taux de couverture et la longueur de test de la séquence finale, obtenue après un assemblage utilisant un seuil TR_{MAX} de 20. La procédure permettant d'obtenir un taux de couverture est décrite dans la troisième partie de ce chapitre.

Approche	TR_{MIN}	**# UT**	**FA**	**Génération**		**Taux**	**# cycles**
				(1)	(2)		
HITEST	—	—	—	813.8	—	98,40%	54.570
Découpage	-1	1	1	480.5	—	98,24%	46.095
Découpage	0	41	1	611.8	502.1	98,36%	86.161
Découpage	30	42	1	700.3	450.4	98,43%	93.612
Découpage	50	81	1.03	662.95	425.5	98,13%	214.462
Découpage	80	211	1.31	632.78	410.1	98,45%	865.950

Ces résultats montrent deux choses. Tout d'abord, nous voyons que le découpage en une seule unité testable et l'outil `HITEST` donnent des résultats équivalents en terme de longueur de test et de taux de couverture. Les temps de génération sont plus courts avec un découpage car la quantité de données à traiter par le générateur de vecteurs de test est moins grande.

Un découpage plus fin ne dégrade pas le taux de couverture, mais augmente considérablement la longueur de test. La génération peut cependant être distribuée sur plusieurs machines et permet d'avoir une durée globale de génération plus courte. Une approche consistant à créer des dictionnaires de fautes intermédiaires permet de limiter l'augmentation du nombre de vecteurs de test, mais ne permet plus de distribuer la génération, qui doit se faire alors séquentiellement, unité testable par unité testable.

Nous constatons donc à travers ces résultats que notre méthode n'est pas très efficace sur ce type de circuit. Il sont en effet fortement couplés et de taille raisonnable, ce qui privilégie une approche globale, où l'ensemble du circuit est pris en compte. Le seul avantage apporté par notre méthode est la simplification des données qui sont fournies à l'outil de génération de vecteurs de test, ce qui permet d'obtenir des temps de génération plus courts.

7.1.6 Ajout des vecteurs des unités auto-testées

Pour chaque unité auto-testée, nous possédons une séquence de test. Cette séquence doit s'appuyer sur les connecteurs externes du circuit. Comme nous l'avons défini au chapitre 6, cette séquence possède trois parties : une initialisation, la séquence d'auto-test proprement dite, et la collecte des résultats.

En pratique, la séquence d'auto-test est longue et ne monopolise que très peu de ressources externes du circuit. Si les connecteurs utilisés pendant la séquence d'auto-test ne sont utilisés par aucune unité testable ni par le *scan-path* associé, alors cette séquence peut être parallélisée avec le test des unités testables. Dans le cas contraire, l'ensemble de la séquence d'auto-test est ajoutée à la séquence issue du test des unités testables.

7.2 Prise en compte du *Boundary-Scan*

Le *Boundary-Scan* est une norme visant à simplifier le test d'une carte comportant plusieurs circuits intégrés. En quelques années cette norme s'est imposée au point d'être présente dans la grande majorité des circuits intégrés actuels, comme nous l'avons vu au chapitre 1. La circuiterie *Boundary-Scan* est complexe, elle est donc souvent générée par des outils automatiques [RMK93]. Dans notre méthode de découpage du circuit en vue du test, il est impératif de différencier dans la description, les portes qui appartiennent au cœur du circuit, des portes qui appartiennent au *Boundary-Scan*. En particulier, ces portes ne doivent pas appartenir à des unites testables. En ce qui concerne l'assemblage des tests, l'utilisation des propriétés du *Boundary-Scan* permet de produire une séquence de test réutilisable lorsque le circuit se trouve sur une carte. Il faut pour cela connaître la correspondance entre les connecteurs du circuit et les bascules du registre *Boundary-Scan*.

Alors que le *scan-path* ne possède pas de description formelle, l'architecture *Boundary-Scan* est clairement définie par la norme IEEE 1149.1 [IEE90]. Entre autres, les protocoles d'accès aux ressources de test sont explicitement fixés par le comportement de la machine à états interne au contrôleur de *Boundary-Scan*. Ainsi, n'importe quel circuit répondant à la norme IEEE 1149.1 est censé réagir de la même façon du point de vue de son architecture *Boundary-Scan*.

L'architecture *Boundary-Scan* est extensible. Au delà des ressources imposées par la norme, un nombre non limité de structures de test propriétaires peut être rattaché à l'architecture de base. Un *scan-path* interne peut, par exemple, être intégré au *Boundary-Scan* : sa mise en mode de décalage ou en mode fonctionnel se fera alors à travers les entrées sorties propriétaires de la norme. Les techniques décrites dans le chapire 4, pour l'identification des points observables et contrôlables, restent entièrement applicables au prix de simulations logiques plus longues.

Nous allons montrer comment une architecture *Boundary-Scan* est reconnue automatiquement à l'aide des mêmes techniques de simulations qu'au chapitre 4. Puis nous montrerons quelle est l'interaction du *Boundary-Scan* avec notre méthode de découpage et notre procédure d'assemblage des tests.

7.2.1 Description de l'architecture *Boundary-Scan*

L'architecture *Boundary-Scan* minimale imposée par la norme est présentée sur la figure 7.5. Elle est composée principalement de deux blocs de contrôle et de trois registres à décalage.

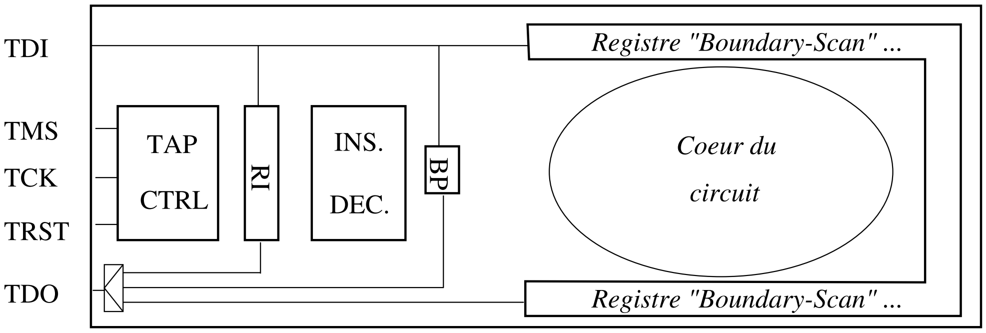


FIG. 7.5 –: Architecture *Boundary-Scan* minimum

Le premier bloc (*TAP CTRL* sur la figure) est la machine à états qui contrôle l'ensemble de la circuiterie. Cet automate synchrone, qui change d'état sur un front montant de l'horloge dédiée TCK, en fonction de la valeur du signal dédié TMS, est complétement défini par la norme. Son diagramme d'état est présenté dans la figure 7.6. Le premier registre à décalage est le registre d'instruction (*RI*), de longueur variable. Il est destiné à contenir l'instruction qui va sélectionner le mode de fonctionnement du *Boundary-Scan*. Le deuxième bloc (*INS. DEC.* sur la figure) est le décodeur d'instruction qui génère les signaux de contrôle à destination du reste de l'architecture. Le deuxième registre est appelé « bypass » (*BP*), et n'est formé que d'un bit. Enfin, le dernier registre est le registre *Boundary-Scan*. C'est un registre périphérique qui s'insère entre chaque signal externe du cœur du circuit et le plot correspondant.

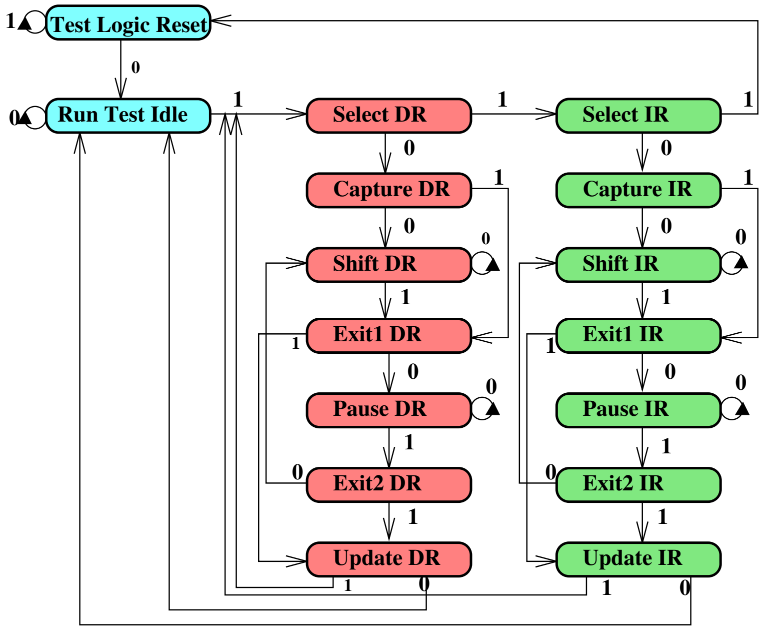


FIG. 7.6 –: Comportement normalisé de la machine à états de l'architecture *Boundary-Scan* par rapport au signal TMS

En ce qui concerne le problème de la reconnaissance automatique d'une architecture *Boundary-Scan*, nous pouvons faire plusieurs remarques.

Tout d'abord, à l'exception des deux blocs de contrôle, l'architecture *Boundary-Scan* est composée de registres à décalage. Or la reconnaissance automatique de registre à décalage a été décrite plus haut : la seule difficulté est de bien définir les modes de fonctionnement de ces registres.

Ensuite, le fonctionnement de la machine à états étant normalisé, les séquences permettant de mettre en mode de décalage tel ou tel registre sont indépendantes de l'implémentation et peuvent donc être générées automatiquement.

Enfin, toutes les extensions de l'architecture *Boundary-Scan*, comme l'intégration d'un *scan-path* interne ou d'un registre d'identification de matériel, doivent selon la norme prendre la forme de registres à décalage insérés entre TDI et TDO.

La reconnaissance automatique d'architecture *Boundary-Scan* apparaît donc comme une extension de la reconnaissance automatique de *scan-path*.

7.2.2 Méthode de reconnaissance

Nous allons exploiter les modes normalisés de l'architecture *Boundary-Scan* pour en identifier les différentes parties. Ces séquences n'utilisent que les cinq entrées-sorties dédiées : TDI, TDO, TMS, TCK et TRST. Un cycle de séquence *Boundary-Scan* représente un front montant de l'horloge TCK.

Identification du registre d'instruction

Le registre d'instruction est un des pivots permettant de mettre en œuvre l'architecture *Boundary-Scan*. Il faut en effet connaître sa longueur pour activer les différents modes de test. C'est donc le premier élément à identifier.

Pour mettre le registre d'instruction en mode de décalage, nous amenons la machine à états de l'architecture en mode « Shift-IR ». Cela se fait en appliquant la séquence 011000 sur l'entrée TMS une fois la machine initialisée à l'état « Test-Logic-Reset » grâce au signal TRST (voir dans la figure 7.6 le diagramme de la machine à états). Le registre d'instruction se retrouve connecté directement entre TDI et TDO.

L'exploitation du graphe sous contraintes correspondant à ce mode de décalage permet alors d'identifier les points mémoires et les portes formant le registre à décalage selon la même méthode que celle décrite plus haut pour la reconnaisance automatique de *scan-path*. Un ensemble d'attributs symboliques et de fonctions de transfert, adapté à l'architecture *Boundary-Scan*, est utilisé.

Puisque nous connaissons le nombre de bits du registre d'instruction, nous allons charger différentes instructions pour identifier le reste de l'architecture.

Identification du registre de bypass

La norme prévoit qu'une instruction composée de '1' sur toute la longueur du registre d'instruction mette le *Boundary-Scan* en mode de « BYPASS ». Un registre de 1 bit est alors positionné entre TDI et TDO. Nous chargeons donc cette instruction, puis nous amenons la machine à états en mode « Shift-DR ». Le registre à décalage qui est alors extrait selon la méthode décrite plus haut est le registre de « bypass ».

Identification du registre *Boundary-Scan*

La norme prévoit qu'une instruction composée de '0' sur toute la longueur du registre d'instruction mette le *Boundary-Scan* en mode « EXTEST », le registre *Boundary-Scan* étant alors positionné entre TDI et TDO. Nous chargeons cette instruction, puis nous amenons la machine à états en mode « Shift-DR ». Nous extrayons alors la liste des points mémoires et des portes appartenant au registre dans le réseau booléen sous contraintes du circuit.

Lorsque l'instruction « INTEST » est chargée et que la machine à états se trouve en mode « Capture-DR », les valeurs présentes sur les sorties du cœur du circuit sont enregistrées dans le registre *Boundary-Scan*. Ces sorties sont les points observables du registre *Boundary-Scan* en mode « INTEST ». Nous pouvons identifier ces signaux grâce à une exploration du réseau booléen sous contraintes en amont des registres de *Boundary-Scan* comme pour l'identification des signaux observables d'un *scan-path* dans le chapitre 4.

Lorsque l'instruction « INTEST » est chargée et que la machine à états se trouve en mode « Update-DR », les valeurs présentes sur les entrées du cœur du circuit sont contrôlées par le registre *Boundary-Scan*. Ces entrées sont les points contrôlables du registre *Boundary-Scan* en mode « INTEST ». Nous pouvons identifier ces signaux grâce à une exploration du réseau booléen sous contraintes en aval des registres de *Boundary-Scan*.

Ces points observables et contrôlables du point de vue du *Boundary-Scan* vont pouvoir servir de points d'arrêt pour l'algorithme de découpage.

Identification des registres optionnels

Les registres de données optionnels du *Boundary-Scan* tels que le registre d'identification, des registres de *scan-path* contrôlés par *Boundary-Scan*, ou encore des registres de contrôle de blocs auto-testés peuvent être aussi identifiés. Il faut pour cela simplement connaître l'instruction permettant de décaler ces registres.

Identification de la logique de contrôle

La logique de contrôle du *Boundary-Scan* est composée, d'un point de vue architectural, de la machine à états et du décodeur d'instructions. D'un point de vue fonctionnel, elle est composée de l'ensemble des portes ayant contribuées à la mise en œuvre d'un des modes. L'ensemble des séquences logiques, appliquées au circuit pour identifier les trois registres à décalage, ne représente qu'un sous-ensemble des possibilités de l'architecture. Nous allons donc compléter les séquences utilisées précédemment par une séquence de 32 cycles qui consiste à balayer exhaustivement l'ensemble des transitions de la machine à états.

Au cours de l'ensemble des séquences permettant la reconnaissance automatique du *Boundary-Scan*, tous les nœuds ayant pris une valeur logique différente de 'X' sont marqués comme faisant partie de la logique de contrôle du *Boundary-Scan* à l'aide du marqueur CONTRÔLE. Ces informations seront exploitées au cours de la phase d'assemblage des vecteurs et des dictionnaires de fautes.

7.2.3 Interaction du *Boundary-Scan* avec la méthode de découpage et d'assemblage

L'interaction de l'architecture *Boundary-Scan* avec notre méthode de découpage dépend du format final de la séquence de test. Si celle-ci doit être appliquée à partir de l'interface *Boundary-Scan*, alors les connecteurs du circuit ne doivent plus être considérés comme des signaux observables et contrôlables. Ce sont les signaux observables et contrôlables du point de vue du registre *Boundary-Scan* et de l'instruction « INTEST » qui les remplacent. Ainsi, notre algorithme de découpage ne fabrique des unités testables qu'à partir du cœur du circuit, sans intégrer de portes appartenant à l'architecture *Boundary-Scan*. En revanche, le coût d'application des tests est démultiplié. En effet, nous devons décaler le registre *Boundary-Scan* pour appliquer des valeurs sur l'interface du circuit. Nous sommes donc ramenés à un problème de multiples *scan-path*, où le registre *Boundary-Scan* est l'un d'entres eux.

Si par contre, le test final doit s'appliquer à l'ensemble des connecteurs du circuit, alors le registre *Boundary-Scan* est rendu transparent en maintenant le signal TRST à 0. Puisque le découpage en unités testables élémentaires se fait dans le graphe du circuit mis sous contraintes, aucune porte du registre *Boundary-Scan* n'appartient à une UTE.

Grâce à la reconnaissance intégrée du *Boundary-Scan*, nous sommes donc capables d'extraire uniquement le cœur du circuit pour en effectuer le test.

De plus, les informations extraites du *Boundary-Scan* servent à générer des séquences dédiées au test de l'architecture elle-même, en se basant sur des algorithmes connus.

7.3 Assemblage des dictionnaires de fautes

Pour quantifier la qualité d'une séquence de test pour un circuit, il faut associer à cette séquence un taux de couverture. Il s'appuie sur un modèle de fautes : c'est le quotient du nombre de fautes détectées par la séquence, divisé par le nombre total de fautes du circuit.

Une faute est la traduction logique d'un défaut physique du composant fabriqué. Il existe de très nombreux modèles de fautes. Certains sont définis au niveau des portes : le modèle des collages stipule que si un circuit est défectueux, alors la sortie d'une porte est « collée » à une valeur logique fixe, 0 ou 1. Dans le cas des *mintermes* [Deb94], une porte n'est testée que si l'ensemble des combinaisons possibles de ses entrées est observé. Certains modèles plus réalistes se basent sur la représentation du circuit en transistors. Ils considèrent qu'une faute se traduit par un circuit ouvert ou par un court-circuit dans la représentation en transistors du circuit. D'autres modèles de fautes enfin sont spécifiques à un type de circuiterie : les fautes de transitions ou les fautes de couplages sont par exemples spécifiques aux matrices des mémoires [dG91].

Il n'est pas possible de comparer des valeurs issues de modèles de fautes différents. Un taux de couverture n'est en effet cohérent que s'il ne concerne qu'un unique modèle de fautes. Si plusieurs modèles de fautes sont utilisés, il faut alors calculer un taux de couverture par modèle.

Un dictionnaire de fautes est la liste de toutes les fautes du circuit avec un statut associé. Dans sa version la plus simple, un dictionnaire précise pour chaque faute si elle est détectée ou non par la séquence. Un dictionnaire peut être plus précis. Le dictionnaire de fautes de l'outil HILO classe par exemple une faute de collage parmi quatre catégories : non détectée, détectée, potentiellement détectée ou indétectable. Il y a alors deux taux de couverture associés à ce dictionnaire : l'un se base uniquement sur les fautes détectées et l'autre prend en compte les fautes potentiellement détectées.

Dans notre méthode de découpage, les séquences de test sont obtenues par des outils extérieurs. Il faut donc s'assurer que les modèles de fautes, les différents statuts et les taux de couverture correspondants aux séquences de test sont compatibles entre eux. Puisque toutes les unités testables sont issues de la même représentation du circuit, les outils de génération de vecteurs de test travaillent sur le même niveau de description pour chaque unité testable. Si de plus, nous utilisons le même type d'outil de génération de vecteurs pour toutes les unités testables, cela assure à priori la cohérence des dictionnaires de fautes entre eux. La représentation du circuit en réseau booléen est par exemple tout à fait adaptée au modèle des collages ou au modèle des mintermes.

A partir de la représentation initiale du circuit non découpé nous construisons un dictionnaire de fautes vide, c'est à dire où chaque faute est marquée « non détectée ». Nous utilisons comme modèle de fautes celui utilisé par l'outil de génération de vecteurs de test des unités testables. Ainsi, à chaque faute issue d'un dictionnaire de fautes d'une unité testable correspond une faute dans le dictionnaire principal. Les dictionnaires de fautes de chaque unité testable sont donc lus un par un pour mettre à jour le dictionnaire principal.

En ce qui concerne les dictionnaires issus des blocs auto-testés, deux approches sont possibles. Soit les modèles de fautes utilisés sont compatibles avec ceux du dictionnaire principal, auquel cas celui-ci est simplement mis à jour comme pour les unités testables; soit le modèle de fautes utilisé n'est pas compatible, auquel cas nous nous servons du dictionnaire artificiel tel que nous l'avons décrit au chapitre précédent. Le dictionnaire original du bloc auto-testé est néammoins conservé.

Notre méthode de découpage forme des unités testables autour des structures de test. Les structures elles-mêmes ne sont pas incluses dans des unités testables. Les fautes associées à ces structures de test ne sont présentes dans aucun dictionnaire de fautes, excepté dans le dictionnaire principal où elles ont le statut « non détectées ». Pourtant nous considérons ici que l'utilisation intensive des structures de test suffit à tester les portes qui les contrôlent. Nous utilisons pour cela les nœuds du réseau booléen marqués CONTRÔLES que nous avons identifiés avec les différentes structures de test. Ces nœuds correspondent par exemple aux signaux de contrôle d'un *scan-path*, ou à la machine à états de l'architecture *Boundary-Scan*. L'ensemble des fautes rattachées à ces nœuds sont marquées comme détectées dans le dictionnaire de fautes principal.

A l'issue de la phase d'assemblage, nous sommes donc capables d'obtenir un dictionnaire de fautes. A partir de ce dictionnaire, nous calculons un taux de couverture global associé à la séquence de test. S'il existe des blocs auto-testés s'appuyant sur un modèle de faute non compatible avec le dictionnaire de fautes principal, les dictionnaires de fautes partiels de ces blocs auto-testés, ainsi que leurs taux de couverture associés, sont conservés.

7.4 Conclusion

Nous avons proposé une approche simple permettant d'assembler les vecteurs de test issus de chaque unité testable. Des techniques de test parallèle d'unités testables permettent de compenser le caractère recouvrant de notre méthode de découpage. L'architecture *Boundary-Scan* n'est pas incompatible avec notre méthode à condition d'identifier les portes qui la composent. Enfin, l'ensemble des dictionnaires de fautes associés à chaque unité testable est rassemblé au sein d'un unique dictionnaire, ce qui permet de définir un taux de couverture global.

Chapitre 8

Mise en œuvre

L'application de notre méthode a donné lieu à la réalisation d'un prototype logiciel. Celui-ci s'appuie sur la chaîne d'outils de CAO ALLIANCE et certains outils commerciaux. Cet environnement nous a permis de valider notre méthode sur des circuits réels.

Nous allons dans la première partie de ce chapitre présenter notre prototype logiciel. La deuxième partie est consacrée à la validation de la méthode présentée dans cette thèse à travers le test d'un circuit complexe : le circuit PCI-DDC.

8.1 Prototype logiciel

Notre méthode s'articule autour de quatre étapes majeures : la reconnaissance des structures de test, le découpage du circuit, la génération des tests sur les unités testables et l'assemblage des résultats. Les deux premières étapes sont regroupées au sein d'un même outil que nous appelons l'outil de découpage. Un autre outil permet l'assemblage des vecteurs de test et des dictionnaires de fautes. Notre méthode est indépendante du choix d'un outil de génération de vecteurs de test : nous avons ainsi utilisé l'outil commercial *HITEST* de la chaine HILO.

8.1.1 L'outil de découpage

L'outil de découpage, appelé *DAST* (Découpage Autour des Structures de Test), est chargé d'identifier les structures de test, d'effectuer le découpage en unités testables élémentaires et le regroupement en unités testables peu recouvrantes, et enfin de sauvegarder sur disque ces unités testables. Le schéma de principe de *DAST* est présenté dans la figure 8.1.

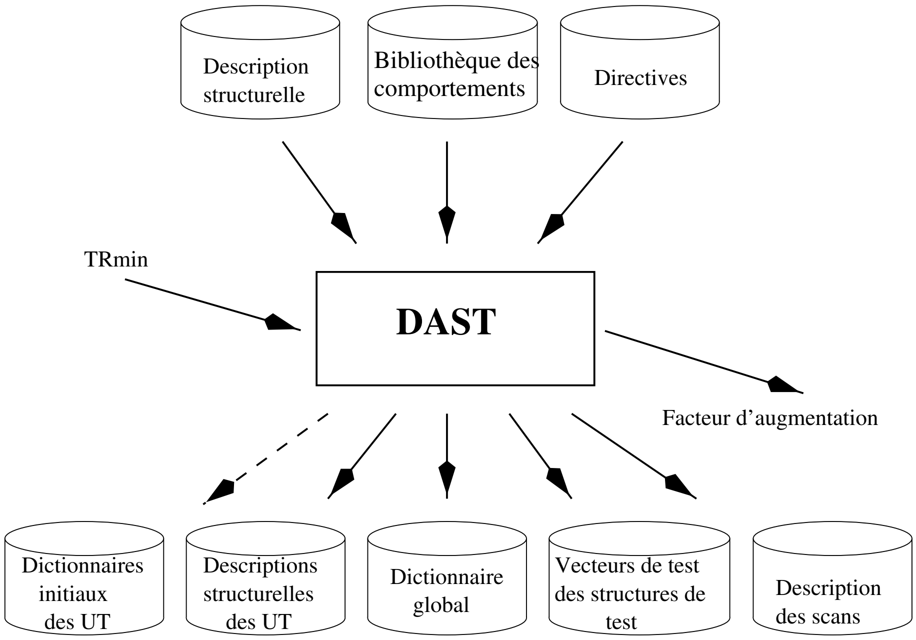


FIG. 8.1 –: Schéma de principe de l'outil de découpage

Le circuit se présente sous la forme d'une description structurelle hiérarchique multi-niveaux et d'une bibliothèque de comportements pour les cellules feuilles. La description structurelle peut être dans de nombreux formats standards dont EDIF et VHDL. Les comportements sont décrits à l'aide du sous-ensemble VHDL de la chaine ALLIANCE. Les directives rassemblent, sous la forme d'un fichier, les informations externes qui caractérisent les *scan-path* du circuit et qui vont permettre d'identifier automatiquement les points contrôlables et observables du circuit. Ce fichier, dont nous donnons un exemple ci-après, permet de spécifier l'entrée et la sortie série du registre à décalage, l'horloge de décalage ainsi que les différentes séquences logiques permettant d'activer ce *scan-path*.

```
SCAN_PATH {
  SCAN_NAME   = chemin_1;
  SCAN_INPUT  = scanin;
  SCAN_OUTPUT = scanout;
  EDGE_CLOCK  = ck;
    func_operation = {
        ck = 1, mode = 1;
        ck = 0;
        } ;
    shift_operation = {
        ck = 1, mode = 0;
        ck = 0;
        } ;
```

```
    func_mode = {
        mode = 1;
        } ;
    shift_mode = {
        mode = 0;
        } ;
}
```

Cette description va être complétée par l'ensemble des signaux observables et contrôlables reconnus par l'outil *DAST*. Ces informations seront exploitées par l'outil d'assemblage. Voici un exemple des informations que va rajouter l'outil de découpage à la description des *scan-path* :

```
  CHAIN = {
          {
            DATA_IN           = g5660 ;
            DATA_OUT          = g1289 ;
            MEMORY_POINT      = g1289.dff_s ;
           },
          {
            DATA_IN           = g9349 ;
            DATA_OUT          = g1882 ;
            MEMORY_POINT      = g1882.dff_s ;
           },
...
          {
            DATA_IN           = g7784 ;
            DATA_OUT          = scanout ;
            MEMORY_POINT      = g73.dff_s ;
           }
           };
}
```

Les signaux `DATA_IN` représentent les points observables du *scan-path*, et les signaux `DATA_OUT` les points contrôlables. `MEMORY_POINT` représente le point mémoire correspondant.

Si le circuit possède des blocs auto-testés, le fichier de directive contient pour chacun la liste des signaux BIST_OBSERVÉS et BIST_CONTRÔLÉS sous la forme suivante :

```
core_rx_rst : BCON
core_ckw_d : BCON
```

```
r3status_from_r3_5 : BCON
core_test_n : BCON
core_test : BCON
...
core_frx_dout_d_32 : BOBS
core_frx_dout_d_31 : BOBS
core_frx_dout_d_30 : BOBS
core_frx_dout_d_29 : BOBS
core_frx_dout_d_28 : BOBS
```

Enfin, le seuil TR_{MIN}, permettant de régler la procédure de regroupement des unités testables élémentaires, est fixé par l'utilisateur.

L'outil $DAST$ effectue les étapes suivantes :

1. Chargement de la description structurelle du circuit et de la librairie de comportement.
2. Construction du réseau booléen orienté que nous avons présenté au chapitre 3.
3. Reconnaissance automatique des signaux observables et contrôlables à partir des informations contenues dans le fichier de description des *scan-path.*
4. Découpage du réseau booléen en unités testables élémentaires, autour des signaux observables et contrôlables, et en unités auto-testées à partir de la liste des signaux BIST_OBSERVÉS et BIST_CONTRÔLÉS.
5. Regroupement des unités testables élémentaires en unités testables peu recouvrantes. Ce regroupement est contrôlé par le paramétre TR_{MIN}.
6. Sauvegarde de chaque unité testable sous la forme d'une description structurelle s'appuyant sur la bibliothèque de comportements, et calcul du facteur d'augmentation.
7. Pour chaque unité testable possédant une intersection non nulle avec une unité auto-testée, sauvegarde d'un dictionnaire de fautes initial.
8. Construction du dictionnaire de fautes global. Les fautes associées aux signaux appartenant à la logique de contrôle des structures de test sont marquées comme détectées au sein du dictionnaire global, ainsi que les fautes associées aux signaux appartenant aux UAT.
9. Génération des vecteurs de test spécifiques aux structures de test.
10. La description complète des chemins de *scan-path* est sauvegardée à destination de l'outil d'assemblage.

Pour chaque unité testable nous disposons d'une description structurelle. Nous pouvons donc effectuer la génération des vecteurs de test indépendamment sur chaque unité testable. Cette génération peut être distribuée sur un réseau de machines. Nous réalisons ces générations grâce à l'outil commercial *HITEST*.

Nous avons choisi de considérer les collages uniques comme modèle de fautes. Nous avons défini un format de dictionnaire de fautes où il est précisé pour chaque faute son statut : active, détectée ou indétectable. Voici un exemple de ce fichier :

```
COVERAGE 81.18
core_txreg_netops3165  STUCK0   DETECTED
core_txreg_netops3165  STUCK1   DETECTED
r3strobe               STUCK0   ACTIVE
r3strobe               STUCK1   DETECTED
core_txreg_netops3435  STUCK0   DETECTED
core_txreg_netops3435  STUCK1   DETECTED
core_txreg_netops3434  STUCK0   DETECTED
core_txreg_netops3434  STUCK1   DETECTED
r3strobe_to_r3         STUCK0   ACTIVE
r3strobe_to_r3         STUCK1   DETECTED
unconnected0           STUCK0   UNDETECTABLE
unconnected0           STUCK1   UNDETECTABLE
logic_one              STUCK1   UNDETECTABLE
logic_one              STUCK0   DETECTED
```

8.1.2 L'outil *HITEST* de la chaine HILO

L'outil *HITEST* de la chaîne HILO [Sys91] est un générateur automatique de vecteurs de test. Il se base sur une représentation structurelle du circuit s'appuyant sur une librairie de comportements décrits en GHDL, un langage proche du VHDL. Sans aucune autre information, il génére des vecteurs sur des circuits combinatoires. Il est aussi capable de générer des vecteurs de test pour des circuits séquentiels synchrones possédant un faible niveau de séquentialité.

Pour générer des vecteurs sur des circuits séquentiels complexes, nous devons fournir à *HITEST* une base de connaissance décrivant les mécanismes de lecture et d'écriture de chaque registre. L'écriture de cette base de connaissance se révèle fastidieuse. Elle demande une connaissance accrue de la structure du circuit et des mécanismes de génération de vecteurs de l'outil *HITEST*.

Un mode simplifié permet la génération de vecteurs de test sur des circuits séquentiels intégrant un *scan-path* complet. La base de connaissance doit pour cela contenir la liste explicite des modèles utilisés dans le circuit pour constituer le chemin de *scan-path*. La connaissance de l'architecture interne du circuit est requise ici, à l'inverse de notre méthode. L'utilisateur doit, en outre, fournir la séquence qui permet de charger et de décharger en série le chemin de *scan-path*. Il doit donc connaître à l'avance la longueur de ce *scan-path*. A l'aide de la base de connaissance, *HITEST* identifie par reconnaissance de formes les points contrôlables et les points observables du chemin de *scan-path*. Il transforme alors les points observables en pseudo-sorties et les points contrôlables en pseudo-entrées. Puis il génère des vecteurs en considérant le circuit comme combinatoire, et en insérant entre deux vecteurs générés un appel à la séquence fournie par l'utilisateur pour décaler le chemin de *scan-path*.

Dans le cadre de notre méthode, l'outil *HITEST* est utilisé pour générer des vecteurs de test sur des unités testables. C'est donc l'outil *DAST* qui se charge d'identifier les structures de test, sans se baser sur une connaissance interne du circuit. Nous n'avons utilisé le mode d'identification des *scan-path* de l'outil `HITEST` qu'à des fins de comparaison 5voi plus haut, page 109).

Enfin, l'outil *HITEST* n'inclut aucun support pour le *Boundary-Scan* ou pour des blocs auto-testés.

La figure 8.2 décrit comment HITEST est utilisé dans notre méthode.

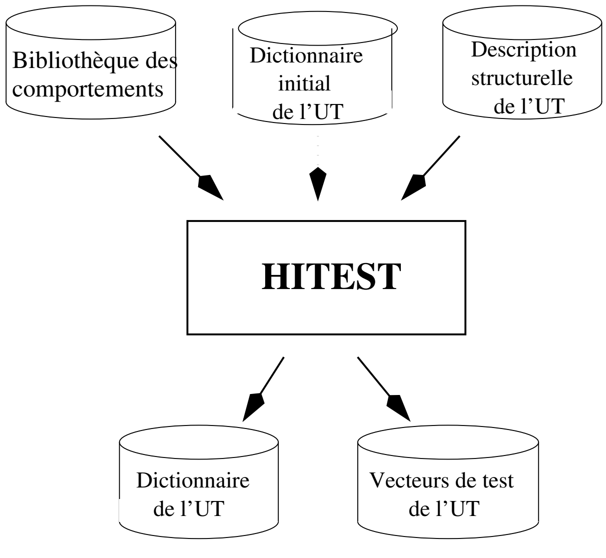


FIG. 8.2 –: Utilisation de l'outil commercial HITEST dans notre méthode

Nous fournissons à l'outil la description structurelle de l'unité testable ainsi que la bibliothèque de comportement. Si l'unité testable possède un dictionnaire de fautes initial, celui-ci est pris en compte par `HITEST`. L'outil fournit en retour le fichier de vecteurs de test et le dictionnaire de fautes associé. L'ensemble de ces informations est décrit dans des formats propres à la chaine `HILO`, mais ne nécessitant que des simples traductions depuis ou vers des formats de la chaine ALLIANCE.

8.1.3 L'outil d'assemblage des tests

L'outil d'assemblage des tests, appelé *TESTA*, permet de rassembler au sein d'une même séquence : l'ensemble des séquences de test issues des unités testables, les vecteurs de test spécifiques aux structures de test fournies par *DAST*, et les vecteurs qui correspondent aux blocs auto-testés. Il permet aussi d'assembler les différents dictionnaires de fautes. Le schéma de principe de l'outil *TESTA* est donné dans la figure 8.3.

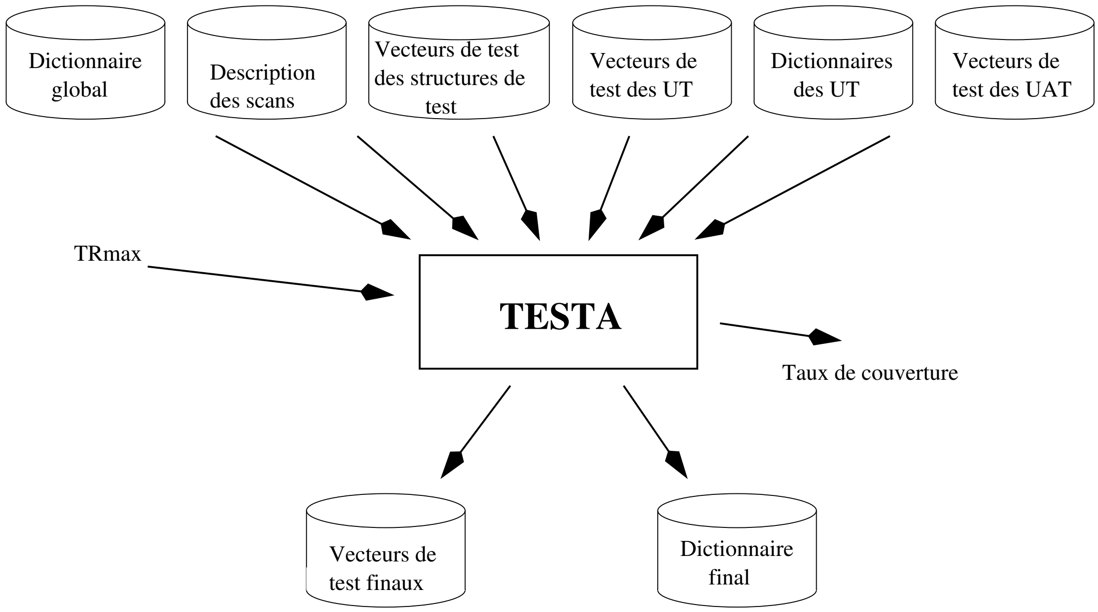


FIG. 8.3 –: Schéma de principe de l'outil d'assemblage

Le dictionnaire global, la description des *scan-path* et les vecteurs de test des structures de test sont issus de l'outil de découpage. Les vecteurs de test des unités testables et leurs dictionnaires associés sont issus des outils de génération automatique de vecteurs de test. Enfin, les vecteurs de test des unités auto-testées, et le seuil TR_{MAX}, qui permet de régler la procédure de test parallèle d'unités testables peu recouvrantes, sont fournis par l'utilisateur.

Les différentes étapes couvertes par l'outil *TESTA* sont :

1. Lecture du fichier de description des *scan-path*.
2. Lecture des fichiers de vecteurs de test des unités testables.

3. Parallélisation des vecteurs de test des unités testables selon le seuil TR_{MAX}.
4. Application des vecteurs sur les unités testables à travers les *scan-path*.
5. Ajout des vecteurs spécifiques aux structures de test et aux unités auto-testées.
6. Sauvegarde de la séquence complète.
7. Chargement du dictionnaire de fautes global.
8. Chargement des dictionnaires de fautes des unités testables et mise à jour du dictionnaire global.
9. Calcul du taux de couverture global.
10. Sauvegarde du dictionnaire final.

Nous obtenons ainsi un dictionnaire de fautes et une séquence de test globale pour le circuit, ainsi qu'un taux de couverture. De plus, si celui-ci n'est pas satisfaisant, il est possible d'identifier les unités testables contenant les fautes non testées et de reprendre la génération des vecteurs sur ces unités testables et non sur l'ensemble du circuit.

8.2 Validation de la méthode sur un circuit réel : PCI-DDC

Pour montrer l'efficacité de notre méthode, nous l'avons appliquée à un circuit complexe contenant des macro-cellules provenant d'un générateur automatique.

8.2.1 Présentation du circuit

Le projet Multi-PC vise à construire une machine parallèle à base de cartes mères de type PC interconnectées par un réseau HSL à 1 Gbit/s [Pot96]. Le circuit PCI-DDC [WDS+97] réalise l'interface entre le bus PCI de la carte mère et le réseau HSL. C'est un circuit de 200.000 transistors, réalisé en technologie CMOS 0.5μ par ALCATEL-MIETEC.

Le circuit PCI-DDC est contrôlé par deux horloges : l'une correspond à l'horloge du bus PCI, l'autre est l'horloge du port de communication avec le réseau HSL. L'interface avec le bus PCI se fait à 33 Mhz sur 32 bits, tandis que l'interface avec le réseau HSL se fait à 80 Mhz sur 8 bits.

La structure interne du circuit est composée de machines à états finis, réalisées en cellules pré-caractérisées, et de cinq mémoires tampons de type « FIFO » permettant de gérer les échanges entre le lien haut débit du routeur Rcube et le bus PCI. Ces cinq blocs sont des macro-cellules, issues d'un générateur paramétrable de « FIFO » asynchrones [DG96]. Ce générateur fournit deux vues pour chaque « FIFO » : une vue physique et un comportement décrit en VHDL. Enfin, la version actuelle du circuit PCI-DDC ne possède pas d'architecture *Boundary-Scan*.

8.2.2 Testabilité du circuit

L'ensemble des registres qui appartiennent aux parties réalisées en cellules pré-caractérisées sont inclus dans quatre *scan-path*. Le premier rassemble 1651 bascules contrôlées par le front montant de l' horloge PCI. Le second rassemble 100 bascules contrôlées par le front descendant de l'horloge PCI. Le troisième est formé des 88 bascules contrôlées par le front descendant de l'horloge Rcube. Le dernier est formé des deux seules bascules contrôlées par le front montant de l'horloge de Rcube. Nous disposons donc de 1841 registres « scannables » au sein des cellules pré-caractérisées du circuit.

Deux des « FIFO » du circuit sont d'une capacité de 16 mots de 33 bits, les trois autres ayant une capacité de 48 mots de 33 bits. Dans chacune d'elle, seuls les 33 registres contenant le mot de sortie et un registre contenant le bit `empty` sont réunis dans un *scan-path* interne à chaque macro-cellule. Notons que le signal de sortie `full` n'est pas contrôlé par un *scan-path*. Au niveau du circuit PCI-DDC, nous trouvons donc deux *scan-path* supplémentaires : le premier rassemble les 34 registres de la « FIFO » contrôlée par l'horloge de Rcube, et le second rassemble les 136 registres des quatre autres « FIFO », contrôlées par l'horloge du bus PCI. La figure 8.4 présente l'interface et les registres de *scan-path* d'une « FIFO ».

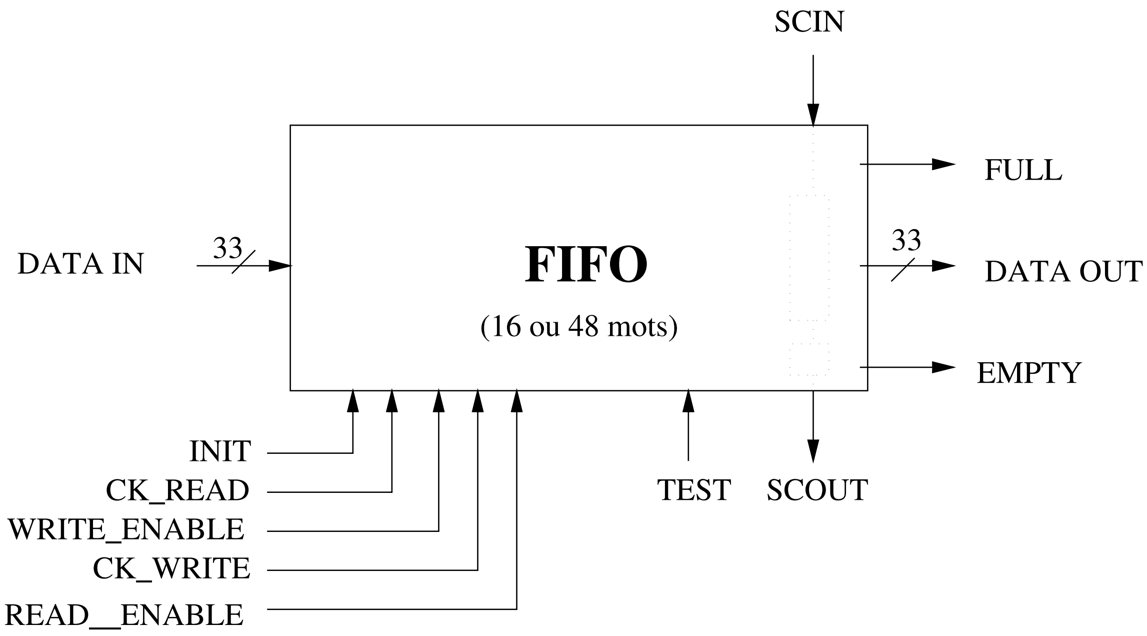


FIG. 8.4 –: Interface des macro-cellules de type « FIFO »

Les « FIFO » possèdent un nombre important de registres internes non « scannables », et ne sont pas équipées d'un mécanisme d'auto-test. En revanche, des séquences fournies par les concepteurs permettent de tester les cinq « FIFO » depuis l'interface du circuit. Il n'est donc pas nécessaire de générer des vecteurs pour ces macro-cellules. Nous considérons que les sorties des « FIFO » sont BIST_OBSERVÉES, et que les entrées sont BIST_CONTRÔLÉES. Ainsi, lors de la constitution du dictionnaire initial, toutes les fautes de collages attachées aux signaux internes des « FIFO » sont considérées détectées.

Bien que ces macro-cellules soient testées, nous devons étudier l'interaction de ces blocs avec notre méthode de découpage.

En ce qui concerne les sorties des « FIFO », celles-ci sont contrôlables par *scan-path* à l'exception du signal `full`. Pour deux des « FIFO » ce signal n'est pas connecté dans le circuit PCI-DDC : notre découpage s'arrête effectivement en frontière de ces macro-cellules. Pour les trois autres « FIFO » les cône de logique qui correspondent aux sorties `full` sont, par contre, inclus dans des unités testables.

Pour ce qui concerne les entrées de données des « FIFO », les concepteurs du circuit ont pris soin d'introduire des registres possédant un *scan-path* en entrée des « FIFO ». Les signaux d'entrées de données des « FIFO » sont donc observables. En revanche, les signaux de contrôle des « FIFO » ne sont pas observables. Pour les deux « FIFO » dont le signal `full` n'est pas connecté, les cônes de logique qui correspondent à ces signaux de contrôle ne sont donc inclus dans aucune unité testable. Ceci met en lumière l'importance d'une « encapsulation » réellement systématique des unités auto-testées ou des macro-cellules, telle qu'elle est décrite au chapitre 6 : le simple fait de n'avoir pas rendu observables les signaux de contrôle a exclu plusieurs portes du découpage en unités testables. L'outil de découpage est cependant capable de fournir la liste des portes ou des signaux qui ne sont pas inclus dans une unité testable.

Pour les trois autres « FIFO », ces cônes de logique sont inclus dans les cônes de logique de la sortie `full`. Ces deux situations sont illustrées par la figure 8.5, où les zones grisées représentent les parties incluses dans des unités testables.

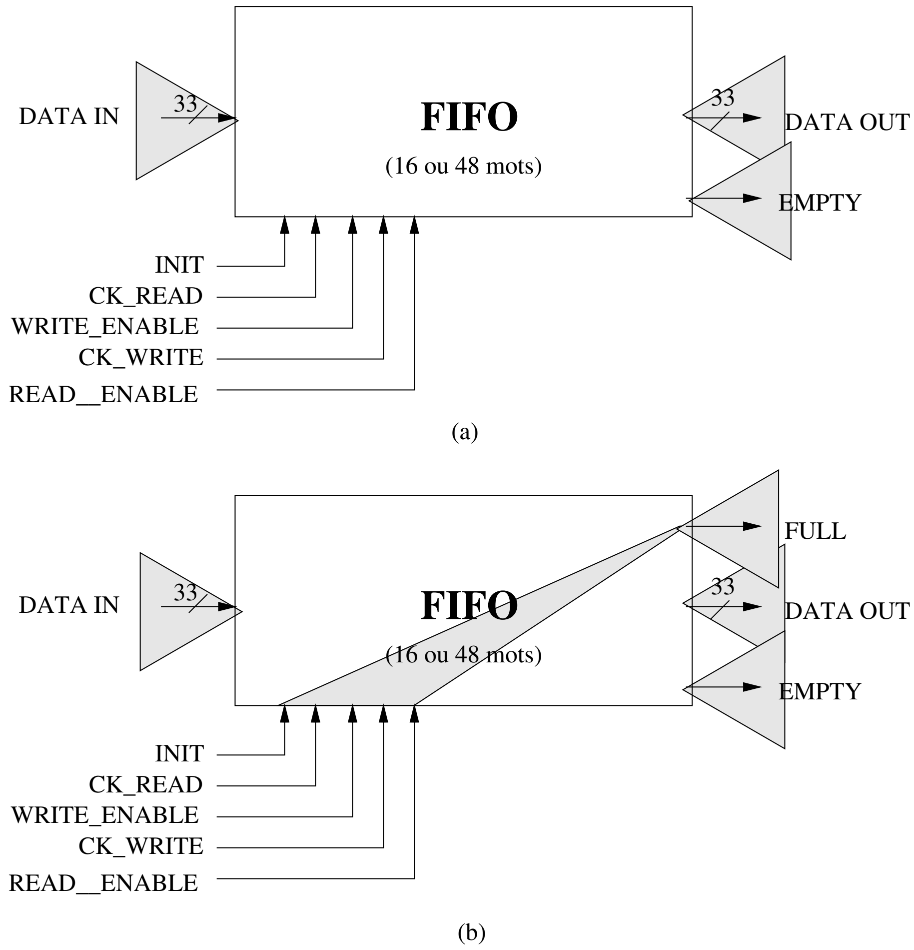


FIG. 8.5 –: Interaction du découpage avec les « FIFO ». (a) sans `full` connecté, (b) avec `full` connecté.

Trois « FIFO », dont toutes les sorties ne sont pas contrôlables, sont donc partiellement incluses dans des unités testables.

8.2.3 Description du circuit

Le circuit PCI-DDC est disponible sous la forme d'une description structurelle hiérarchique s'appuyant sur une bibliothèque de comportements décrits en VHDL. Chaque élément de la bibliothèque correspond à une cellule pré-caractérisée de quelques transistors, à l'exception des « FIFO ». Une « FIFO » de 16 mots représente environ 8.000 transistors, et une « FIFO » de 48 mots représente environ 18.000 transistors.

Comme nous l'avons vu plus haut, la procédure de découpage passe à travers les « FIFO ». Nous avons obtenu une description structurelle de niveau « portes » pour les « FIFO » grâce à un outil d'abstraction fonctionnelle, `Yagle` [LGM92], qui permet de reconstruire une description structurelle en portes orientées à partir d'une description en transistor, ainsi que les comportements associés à chacune de ces portes.

8.2.4 Application de la méthode sur PCI-DDC

Le réseau booléen orienté, correspondant au circuit, est d'une complexité de 50.000 nœuds. Nous donnons en annexe le fichier initial de description des *scan-path* qui permet à l'outil `DAST` d'identifier les registres des six *scan-path*, ainsi que les signaux observables et contrôlables qui leur correspondent. Cette reconnaissance se fait à l'aide de six passes de simulation selon la procédure décrite au chapitre 4. Nous obtenons alors 2171 points contrôlables et 1886 points observables.

Le découpage initial du circuit permet la construction de 1884 unités testables élémentaires et de cinq unités auto-testées.

Nous avons vu qu'un seuil TR_{MIN} trop grand pour effectuer le regroupement des unités testables élémentaires provoque une forte augmentation du nombre de vecteurs. Nous avons donc choisi un seuil TR_{MIN} d'une valeur de 20. Nous obtenons ainsi 10 unités testables. La majorité de ces unités testables est très petite. Il s'agit par exemple des portes situées entre un registre de *scan-path* et une sortie du circuit.

Comme nous l'avons prévu, une partie des trois « FIFO », dont le signal `full` est connecté, se trouve intégré dans des unités testables élémentaires. Grâce à la description structurelle fine des « FIFO », seules les parties réellement concernées sont incluses dans les unités testables, et non pas l'ensemble des « FIFO ». Par contre, une partie des signaux de commande des « FIFO » étant communs, les trois cônes de logiques sont rassemblés dans la même unité testable. Celle-ci à une complexité de 14.000 nœuds.

Enfin, le facteur d'augmentation résultant de ce découpage est de 1,01. Cela signifie qu'en moyenne 1% des portes appartiennent à deux unités testables.

A l'issue du découpage, l'ensemble des unités testables est sauvegardé. Dans le dictionnaire de fautes global généré, toutes les fautes relatives aux signaux internes des « FIFO » sont marquées « détectées ». La description complète des *scan-path* est aussi sauvegardée à destination de l'outil d'assemblage.

L'ensemble de la phase de reconnaissance des structures de test et de découpage dure environ 15 minutes sur une station de travail SPARC10 possédant 400 Mo de mémoire.

La plupart des unités testables obtenues est purement combinatoire. Ainsi il est très facile d'obtenir des vecteurs pour les tester à l'aide de l'outil `HITEST`. L'unité testable contenant des parties de « FIFO » est plus volumineuse et contient plus de 600 registres. `HITEST` a pourtant réussi à générer des vecteurs pour un taux de couverture de 88% sur cette unité testable. L'ensemble de la phase de génération a duré environ 20 minutes, dont 18 minutes pour l'unité testable la plus grosse. La mauvaise répartition des tailles de ce découpage, et la taille modeste de l'unité testable la plus grosse, ne nous ont pas donné la possibilité d'utiliser une distribution de la génération sur un réseau de machines.

A l'issu de la phase de génération, nous avons obtenu 246 vecteurs à appliquer sur les unités testables. L'outil *TESTA* les a transformés en 290.225 cycles sur le circuit PCI-DDC, pour un taux de couverture global de 98,94% selon le modèle des collages uniques. Il faut rajouter à cela les 50.000 cycles fonctionnels fournis par les concepteurs pour tester les « FIFO ».

Notre méthode de découpage a prouvé son efficacité pour ce qui est de sa capacité à utiliser des outils classiques de génération de vecteurs de test sur des circuits complexes. En effet, d'un circuit de 50.000 nœuds, seul le tiers a nécessité une génération déterministe de vecteurs de test. Cette simplification a été rendue possible par la reconnaissance automatique de points observables et contrôlables, et par le découpage autour de ces points. La présence de cinq macro-cellules, bien que moyennement « encapsulées », n'a que très peu perturbé la génération. Malheureusement ce circuit est trop modeste pour tirer pleinement partie du découpage recouvrant.

8.3 Conclusion

Nous avons développé deux outils interfacés avec le générateur automatique de vecteurs de test commercial `HITEST` et s'intégrant dans la chaine ALLIANCE pour valider notre méthode. Ils nous ont permis, entre autres, de générer des vecteurs de test pour un circuit qui possède des macro-cellules.

L'application de notre méthode à des circuits complexes, incluant des macro-cellules ou des blocs auto-testés, permet d'utiliser un outil classique de génération de vecteurs de test qui serait incapable de traiter seul de tels circuits.

Nous pensons que cette approche est particulièrement réaliste pour des circuits complexes, formés d'assemblages de macro-cellules faiblement couplées et issues de bibliothèques hétérogènes, et dont la taille excède les capacités des outils de génération actuels.

Conclusion

Dans ce travail, nous avons cherché à résoudre le problème de la génération des tests sur des circuits numériques complexes possédant déjà des structures de test, et incluant des macro-cellules ou des blocs auto-testés. Les performances des outils actuels de génération automatique de vecteurs de test ne permettent pas un traitement global de ce type de circuit. Il faut alors le découper en sous-parties plus modestes, en s'appuyant sur les structures de test présentes. Celles-ci doivent être automatiquement identifiées au sein du circuit, à partir de leur mode de fonctionnement. En effet, si les concepteurs utilisent des macro-cellules ou des bibliothèques venant de fabricants différents, ils n'ont pas toujours la connaissance de l'architecture interne de leur circuit. Enfin, bien que le circuit soit découpé en sous-parties, nous devons obtenir au final une seule séquence de test et un seul dictionnaire de fautes pour l'ensemble du circuit, sans re-simulation de l'ensemble.

Nous avons observé que la grande majorité des circuits conçus récemment possèdent des chemins de *scan-path*. De même, depuis sa normalisation, un *Boundary-Scan* est présent dans la plupart des circuits commerciaux. Enfin, de plus en plus souvent nous trouvons des parties auto-testées ou des macro-cellules au sein de ces circuits. Nous avons voulu prouver que la génération automatique de vecteurs de test sur ce type de circuit reste possible avec des outils classiques.

La réponse des industriels aux limites de leurs outils de simulation de fautes et de génération automatique de vecteurs de test est un partitionnement du circuit. Cela peut impliquer l'utilisation de structures spéciales dans le circuit, qui sont alors insérées au cours de la conception. Cette perspective n'est cependant possible que si les concepteurs maîtrisent l'ensemble des bibliothèques et des composants utilisés dans leur circuit. Lorsqu'à l'inverse les points d'accès naturels du circuit sont utilisés, la volonté d'obtenir un découpage en sous-parties totalement indépendantes ne permet pas de maîtriser la complexité de celles-ci.

De même, des outils universitaires proposent des solutions visant à insérer des structures de test en vue d'un découpage automatique. D'autres identifient par reconnaissance de formes des structures de test, présentes dans le circuit, et proposent un découpage en sous-parties indépendantes. Ces solutions ne nous ont pas parues adaptées au type de circuit visé par notre étude.

Nous avons présenté, dans le chapitre 3, notre méthode de test, basée sur un découpage structurel peu recouvrant. Les principales étapes de cette méthode sont :

1. une reconnaissance automatique des points observables et des points contrôlables dans le circuit, sans s'appuyer sur des techniques de reconnaissance de formes,

2. un découpage en sous-parties peu recouvrantes autour de ces points observables et contrôlables,

3. un assemblage optimisé des vecteurs de test et des dictionnaires de fautes associés à chaque sous-partie, afin d'obtenir une séquence unique et un seul dictionnaire de fautes.

Nous avons montré comment décrire un circuit sous la forme d'un réseau booléen à plat. Ce réseau est issu de la représentation structurelle, en portes, du circuit, et des comportements de chaque élément de la bibliothèque.

Nous avons proposé, dans le chapitre 4, une méthode fonctionnelle de reconnaissance automatique de signaux observables et contrôlables dans le réseau booléen du circuit. Cette méthode s'appuie sur des techniques de simulation, logiques et symboliques, déjà utilisées dans le cadre de la vérification de règles de testabilité. Puisqu'elle n'utilise aucune bibliothèque préféfinie, cette reconnaissance automatique s'adapte à toutes formes de *scan-path* dont les mécanismes fonctionnels de mise en œuvre sont connus, ainsi qu'à certaines formes d'auto-test et au *Boundary-Scan*. Cette étape de reconnaissance nous permet de marquer au sein de la représentation du circuit les signaux qui serviront de frontières au découpage.

Dans le chapitre 5, nous avons défini la notion d'unité testable. Puis nous avons détaillé les algorithmes permettant, d'une part de découper un circuit en unités testables élémentaires, puis de regrouper d'autre part ces unités testables pour obtenir un découpage peu recouvrant du circuit. Nous avons défini une représentation simplifiée des unités testables, se basant sur la liste de leurs entrées, et nous avons précisé la notion de taux de recouvrement entre deux unités testables. Un paramètre réglable permet d'obtenir, à l'issue du regroupement, un compromis entre la taille et le nombre des unités testables d'une part, et leur caractère recouvrant d'autre part. Nos unités testables étant indépendantes, il est possible de distribuer la génération des vecteurs de test sur un réseau de machines.

Le chapitre 6 est consacré à la présence de bloc auto-testés ou de macrocellules dans un circuit. Ces blocs ne sont pas incompatibles avec notre méthode de découpage. Lorsque ces blocs possèdent des frontières observables et contrôlables, l'intégration se fait à moindre coût. Nous proposons quelques solutions matérielles et logicielles permettant cette intégration dans le cas contraire. Deux exemples précis de blocs auto-testés viennent illustrer cette intégration.

Nous avons présenté au cours du chapitre 7 une approche simple permettant d'assembler les vecteurs de test issus de chaque unité testable. Nous appliquons en parallèle les vecteurs destinés à des unités testables indépendantes, voire peu recouvrantes. Cela permet de compenser le caractère recouvrant de notre découpage. Nous montrons que la présence d'une architecture *Boundary-Scan* est tout à fait compatible avec notre méthode. Enfin, nous proposons une manière d'assembler les différents dictionnaires de fautes. Cela nous permet d'obtenir un seul dictionnaire de fautes, ainsi qu'un taux de couverture global. Nous n'effectuons pas pour cela une simulation de fautes avec le circuit complet.

Nous avons réalisé un prototype logiciel se basant sur la chaîne ALLIANCE. Ce prototype est composé du générateur de vecteurs de test commercial HITEST, et de deux nouveaux outils : l'outil de découpage et l'outil d'assemblage des résultats. Nous nous sommes servis de ce prototype pour confronter notre méthode au test d'un circuit réél, conçu au laboratoire, PCI-DDC. Ce circuit de 200.000 transistors, bien que n'étant pas assez complexe au vue de nos objectifs, nous a permis de mettre en lumière les forces et les faiblesses de notre méthode.

Nous pensons qu'il serait intéressant de mieux maîtriser la taille des unités testables. En effet, nous avons observé une grande disparité entre elles. Nous obtenons par exemple pour le circuit PCI-DDC beaucoup de petites unités testables très simples, et une unité testable complexe. La prise en compte explicite de la taille des unités testables dans l'algorithme de regroupement permettrait d'équilibrer les tailles des unités testables. Nous pourrions alors mieux exploiter la puissance des outils de génération de vecteurs de test, et rendre la distribution de cette génération plus efficace.

Si nous voulons tirer pleinement partie de la génération distribuée de vecteurs de test, il faut intégrer celle-ci à l'outil de découpage. Il faudrait alors déclencher la procédure de génération dès qu'une unité testable est constituée, avant de passer à la fabrication de la suivante.

Peut-être serait-il intéressant d'intégrer dans notre découpage les notions de points fonctionnellement contrôlables et fonctionnellement observables ? Ce sont des points du circuit où il est possible de lire, ou d'écrire, n'importe quelle valeur grâce à une courte séquence fonctionnelle. C'est, par exemple, le cas du bus de sortie d'un banc de registre : si la séquence permettant de charger une valeur quelconque dans ce banc de registre depuis l'extérieur est connue, alors les sorties du banc de registre sont fonctionnellement contrôlables. Ces points nous permettraient de ne pas limiter les frontières des unités testables aux points observables et contrôlables, et d'étendre ainsi la souplesse de notre découpage. Cela pourrait être particulièrement utile dans le cas de l'intégration de blocs auto-testés dont les frontières ne sont pas observables et contrôlables.

Nous avons souligné au chapitre 2 que les outils proposant un découpage du circuit étaient souvent eux-même des outils d'aide à la conception. La représentation du circuit en réseau booléen orienté, sur laquelle s'appuie notre découpage, peut être facilement issue d'un modèle comportemental de ce circuit. Il est ainsi facile d'obtenir, très tôt dans le cycle de conception du circuit, une première idée de l'efficacité de notre découpage. Cette possibilité peut permettre au concepteur de faire les choix favorisant un découpage optimal, pour ce qui concerne, bien sûr, les parties dont il maîtrise la testabilité.

Annexe 1 : Fichier de description du « scan-path » de PCI-DDC

Ce fichier sert à initier la reconnaissance automatique des chemins de *scan-path* du circuit PCI-DDC par l'outil *DAST*.

```
-- Description du scan-path de pci-ddc
-- Sequence 1
SCAN_PATH {
  SCAN_NAME   = s_msu33;
  SCAN_INPUT  = r3status(0);
  SCAN_OUTPUT = r3dout(0);
  EDGE_CLOCK  = clk;
  TYPE        = FULL_DFF_SCAN;
    fonc_operation = {
        r3status(6) = 0, test = 0, rst_n = 0, clk = 0;
        r3status(6) = 0, test = 0, rst_n = 0, clk = 1;
        } ;
    shift_operation = {
        r3status(6) = 1, test = 1, rst_n = 0, clk = 0;
        r3status(6) = 1, test = 1, rst_n = 0, clk = 1;
        } ;
    fonc_mode = {
        r3status(6) = 0, rst_n = 0, test = 0;
        } ;
    shift_mode = {
        r3status(6) = 1, rst_n = 0, test = 1;
        } ;
}
-- Sequence 2
SCAN_PATH {
  SCAN_NAME   = s_msd33;
  SCAN_INPUT  = r3status(1);
  SCAN_OUTPUT = r3dout(1);
  EDGE_CLOCK  = clk;
```

```
  TYPE        = FULL_DFF_SCAN;
    fonc_operation = {
        r3status(6) = 0, test = 0, rst_n = 0, clk = 1;
        r3status(6) = 0, test = 0, rst_n = 0, clk = 0;
        } ;
    shift_operation = {
        r3status(6) = 1, test = 1, rst_n = 0, clk = 1;
        r3status(6) = 1, test = 1, rst_n = 0, clk = 0;
        } ;
    fonc_mode = {
        r3status(6) = 0, rst_n = 0, test = 0;
        } ;
    shift_mode = {
        r3status(6) = 1, rst_n = 0, test = 1;
        } ;
}
-- Sequence 3
SCAN_PATH {
  SCAN_NAME   = s_msd80;
  SCAN_INPUT  = r3status(2);
  SCAN_OUTPUT = r3dout(2);
  EDGE_CLOCK  = hslclk;
  TYPE        = FULL_DFF_SCAN;
    fonc_operation = {
        r3status(6) = 0, test = 0, rst_n = 0, hslclk = 1;
        r3status(6) = 0, test = 0, rst_n = 0, hslclk = 0;
        } ;
    shift_operation = {
        r3status(6) = 1, test = 1, rst_n = 0, hslclk = 1;
        r3status(6) = 1, test = 1, rst_n = 0, hslclk = 0;
        } ;
    fonc_mode = {
        r3status(6) = 0, rst_n = 0, test = 0;
        } ;
    shift_mode = {
        r3status(6) = 1, rst_n = 0, test = 1;
        } ;
}
-- Sequence 4
SCAN_PATH {
  SCAN_NAME   = s_msu80;
  SCAN_INPUT  = r3status(3);
  SCAN_OUTPUT = r3dout(3);
```

```
  EDGE_CLOCK  = hslclk;
  TYPE        = FULL_DFF_SCAN;
    fonc_operation = {
        r3status(6) = 0, test = 0, rst_n = 0, hslclk = 0;
        r3status(6) = 0, test = 0, rst_n = 0, hslclk = 1;
        } ;
    shift_operation = {
        r3status(6) = 1, test = 1, rst_n = 0, hslclk = 0;
        r3status(6) = 1, test = 1, rst_n = 0, hslclk = 1;
        } ;
    fonc_mode = {
        r3status(6) = 0, rst_n = 0, test = 0;
        } ;
    shift_mode = {
        r3status(6) = 1, rst_n = 0, test = 1;
        } ;
}
-- Sequence 5
SCAN_PATH {
  SCAN_NAME   = s_fifod80;
  SCAN_INPUT  = r3status(4);
  SCAN_OUTPUT = r3dout(4);
  EDGE_CLOCK  = hslclk;
  TYPE        = FULL_DFF_SCAN;
    fonc_operation = {
        r3status(6) = 0, test = 0, rst_n = 0, clk = 1;
        r3status(6) = 0, test = 0, rst_n = 0, clk = 0;
        } ;
    shift_operation = {
        r3status(6) = 1, test = 1, rst_n = 0, clk = 1;
        r3status(6) = 1, test = 1, rst_n = 0, clk = 0;
        } ;
    fonc_mode = {
        rst_n = 0, r3status(6) = 0, test = 0;
        } ;
    shift_mode = {
        rst_n = 0, r3status(6) = 1, test = 1;
        } ;
}
-- Sequence 6
SCAN_PATH {
  SCAN_NAME   = s_fifou33;
  SCAN_INPUT  = r3status(5);
```

```
  SCAN_OUTPUT = r3dout(5);
  EDGE_CLOCK  = clk;
  TYPE        = FULL_DFF_SCAN;
    fonc_operation = {
        r3status(6) = 0, test = 0, rst_n = 0, clk = 0;
        r3status(6) = 0, test = 0, rst_n = 0, clk = 1;
        } ;
    shift_operation = {
        r3status(6) = 1, test = 1, rst_n = 0, clk = 0;
        r3status(6) = 1, test = 1, rst_n = 0, clk = 1;
        } ;
    fonc_mode = {
        rst_n = 0, r3status(6) = 0, test = 0;
        } ;
    shift_mode = {
        rst_n = 0, r3status(6) = 1, test = 1;
        } ;
}
-- End
```

Bibliographie

[Aba89] Magdy S. Abadir. TIGER: Testability Insertion Guidance Expert System. In *Proceedings of the International Conference on Computer Aided Design*, pages 562–565, 1989.

[Aut91a] COMPASS Design Automation. *TEST TOOLS : BISTRAM Compiler*, 1991.

[Aut91b] COMPASS Design Automation. *TEST TOOLS : BISTROM Compiler*, 1991.

[BDSS90] Frans Beenker, Rob Dekker, Rudi Stans, and Max Van Der Star. Implementing Macro Test in Silicon Compiler Design. *IEEE Design & Test of Computers*, pages 41–51, April 1990.

[Bot86] P.S. Bottorf. Test Generation and Fault Simulation. In Elsevier Sciences Publishers B.V., editor, *VLSI TESTING – Advance in CAD for VLSI, volume 5*, pages 29–64, 1986.

[Bry90] R.E. Bryant. Symbolic Simulation – Techniques and Applications. In *ACM IEEE*, pages 517–521, 1990.

[CAG+92] D. Crestani, A. Aguila, M.-H. Gentil, P. Chardon, and C. Durante. Automatic Partitioning for Deterministic Test. In *Proceedings of the EURODAC*, pages 322–325, 1992.

[CPC94] Alfred L. Crouch, Matthew Pressly, and Joe Circello. Testability Features of the MC68060 Microprocessor. In *Proceedings of the International Test Conference*, pages 60–69, 1994.

[Cre91] D. Crestani. *Partitionnement et test hiérarchisé des parties combinatoires d'un circuit séquentiel synchrone.* PhD thesis, Université Montpellier II – Sciences et Techniques du Languedoc, Décembre 1991.

[Deb94] Philippe Debaud. *TAUMMI : un système de génération de vecteurs, de simulation de fautes et de compilation en vue de test utilisant* *une méthode localement exhaustive*. PhD thesis, Université Pierre et Marie Curie, 1994.

[dG91] A.J. Van de Goor. *Testing semiconductor memories - Theory and Practice*. Edition Wiley, 1991.

[DG96] Karim Dioury and Alain Greiner. *Asynchronous FIFO Datasheet, document interne LIP6*, 1996.

[DM93] M. Davis-Moradkian. *Problème de Partitionnement dans la Technologie des VLSI*. PhD thesis, Université Pierre et Marie Curie, Laboratoire MASI, Mars 1993.

[EW77] E.B. Eichelberger and T.W. Williams. A Logic Design Structure for LSI Testability. In *Proceedings of the 14th Design Automation Conference*, pages 462–468, 1977.

[FKY89] Shigehiro Funatsu, Masato Kawai, and Akihiko Yamada. Scan Design at NEC. In *IEEE Design & Test of Computers*, pages 50–57, 1989.

[Gel87] P. Gelsinger. Design and Test of the 80386. *IEEE Design & Test of Computers*, June 1987.

[GIKN89] Patrick Gelsinger, Sundar Iyengar, Joseph Krauskopf, and James Nadir. Computer Aided Design and Built In Self Test on the i486 CPU. In *Proceedings of the International Conference on Computer Design*, pages 199–202, 1989.

[GMDz90] T. Grüning, U. Mahlstedt, W. Daehn, and C. Özcan. Accelerated Test Pattern Generation by Cone-Oriented Circuit Partitioning. In *Proceedings of the European Design Automation Conference*, pages 418–421, 1990.

[GP94] Alain Greiner and Frédéric Pétrot. A public domain high performances portable ROM generator. In *proceedings of the 20th EuroMicro*, pages 414–419, 1994.

[GSB91] Rajiv Gupta, Rajagopalan Srinivasan, and Melvin A. Breuer. Reorganizing Circuits to Aid Testability. *IEEE Design & Test of Computers*, pages 49–57, 1991.

[GZ90] Naga Gollakota and Ahmed Zaidi. Fault Grading the INTEL 80486. In *Proceedings of the International Test Conference*, pages 758–761, 1990.

[HFGR94] M. Hirech, O. Florent, A. Greiner, and E. Rejouan. Application of a Redefinable Symbolic Simulation Technique in VLSI Testability Design Rules Checking. In *IEEE 27th Annual Simulation Symposium*, pages 255–261, April 1994.

[Hir93] Mokthar Hirech. *Spécification et développement d'un vérificateur de règles de testabilité des circuits VLSI*. PhD thesis, Université Pierre et Marie Curie, 1993.

[HSBP95] Graham Hetherington, Greg Sutton, Kenneth M. Butler, and Theo J. Powell. Test Generation and Design for Test for a Large Multiprocessing DSP. In *Proceedings of the International Test Conference*, pages 149–156, 1995.

[HVTL94] C. Hunter, E. K. Vida-Torku, and J. Leblanc. Balancing Structured and Ad_hoc Design for Test : Testing of the PowerPC 603 Microprocessor. In *Proceedings of the International Test Conference*, pages 76–83, 1994.

[IEE87] IEEE Standards Board, 345 East 47th St. New York 10017. *IEEE Standard VHDL Language Reference Manual : IEEE Standard 1076-1987*, 1987.

[IEE90] IEEE Standards Board, 345 East 47th St. New York 10017. *IEEE Standard Test Access Port and Boundary-Scan Architecture : IEEE Standard 1149.1-1990*, May 1990.

[JDA93] Don D. Josephson, Daniel J. Dixon, and Barry J. Arnold. Test Features of the HP PA7100LC Processor. In *Proceedings of the International Test Conference*, pages 764–772, 1993.

[JPP89] Wen-Ben Jone, C. A. Papachristos, and M. Pereira. A Scheme for Overlaying Concurrent Testing of VLSI Circuits. In *Proceedings of the 26th Design Automation Conference*, pages 531–536, 1989.

[KAM+96] Sankaran Karthik, Mark Aitken, Glidden Martin, Srinivasu Pappula, Bob Stettler, Praveen Vishakantalah, Manuel d'Abreu, and Jacob A. Abraham. Distributed Mixed Level Logic and Fault Simulation on the Pentium Pro Microprocessor. In *Proceedings of the International Test Conference*, pages 160–166, 1996.

[KMZ79] B. Koenemann, J. Mucha, and G. Zwiehoff. Built-in Logic Block Observation Techniques. In *proceedings of the International Test Conference*, pages 37–41, 1979.

[Lau94] Marc Laurentin. *Abstraction fonctionnelle de circuits intégrés CMOS*. PhD thesis, Université Pierre et Marie Curie, 1994.

[LCE86] David O. Lahti and Grace C. Chen-Ellis. PROSPECT : A Production System for Partitionning and Evaluating Chip Testability. In *Proceedings of the International Test Conference*, pages 360–367, 1986.

[LGM92] M. Laurentin, A. Greiner, and R. Marbot. DESB, a Functional Abstractor for CMOS VLSI circuits. In *Proceedings of the European Design Automation Conference*, pages 22–27, 1992.

[LKL93] Wen-Joung Lai, Chen-Pin Kung, and Chan-Sang Lin. Test Time Reduction in Scan Designed Circuits. In *Proceedings of the 4th European Test Conference*, pages 489–493, 1993.

[MDNSV88] Hi-Keung Tony MA, Srinivas Devadas, A. Richard Newton, and Alberto Sangiovanni-Vincentelli. Test Generation for Sequential Circuits. In *IEEE Transactions on CAD*, pages 1081–1093, 1988.

[MG] M.Minouk and G.Bartnik. *Graphes, algorithmes, logiciels*. DUNOD informatique. ISBN:2-04-046470-7.

[MGzD90] Udo Mahlstedt, Torsten Grüning, Cengiz Özcan, and Wilfried Daehn. CONTEST: A Fast ATPG Tool for Very Large Combinational Circuits. In *Proceedings of the International Conference on Computer-Aided Design*, pages 222–225, 1990.

[NG96] Wayne Needham and Naga Gollokota. DFT Stategy for INTEL Microprocessor. In *Proceedings of the International Test Conference*, pages 396–399, 1996.

[OBT91] Steven Oostdijk, Frans Beenker, and Loek Thijssen. A Model for Test-Time Reduction of Scan Testable Circuits. In *Proceedings of the 2th European Test Conference*, pages 243–252, 1991.

[Pic96] F. Pichon. Testability Features for a Submicron Voice-code ASIC. In *Proceedings of the International Test Conference*, pages 377–383, 1996.

[Pot96] Frédéric Potter. *Conception et réalisation d'un réseua d'interconnexion à faible latence et haut débit pour machines multiprocesseurs*. PhD thesis, Université Pierre et Marie Curie, Avril 1996.

[PY93] Rajiv Patel and Krishna Yarlagadda. Testability Features of the SuperSPARC Microprocessor. In *Proceedings of the International Test Conference*, pages 773–781, 1993.

[RMK93] M. F. Robinson, F. Mailhot, and J. Konsevich. Technology-Independant Boundary Scan Synthesis (Technology and Physical Issues). In *IEEE International Test Conference*, pages 157–166, 1993.

[SMG90] Thomas Spohrer, Daniel Marquette, and Michael Gallup. Test Architecture of the Motorola 68040. In *Proceedings of the International Conference on Computer Design*, pages 191–194, 1990.

[Sta90] Cordt W. Starke. Design for testability and diagnosis in a VLSI CMOS System/370 processor. In *IBM Journal of Research and Development*, pages 355–362, March/May 1990.

[Sys91] GenRad Design Automation System. *Hitest reference manual*, December 1991.

[VTMPR95] E. Koft Vida-Torku, Charles H. Malley, Sung Park, and Rowland Reed. Design and Test of the PowerPC 603 Microprocessor. In *Proceedings of the International Test Conference*, pages 378–384, 1995.

[Vuo97] Huu-Nghia Vuong. *Une nouvelle méthode de simulation par évaluation directe des expressions logiques représentées par des graphes : application à des circuits intégrés décrits par un sous-ensemble du langage VHDL.* PhD thesis, Université Pierre et Marie Curie, 1997.

[WDS+97] F. Wajsbürt, J.L. Desbarbieux, C. Spasevski, S. Penain, and A. Greiner. An Integrated PCI component for IEEE 1355 Networks. In *EMMSEC 97*, 1997.

[WP83] T. W. Williams and K. P. Parker. Design for Testability – A Survey. In *Proceedings of the IEEE*, volume 71, January 1983.

A TEST METHOD FOR INTEGRATED CIRCUITS, BASED ON A SPLITTING AND OVERLAPPING STRATEGY

Abstract

Our study presents a "divide and conquer" strategy for digital circuit testing, in order to generate deterministic patterns using classical ATPG tools. Today's integrated circuits have scan-paths, some are using built-in self-test or macro-cells, and almost all have a Boundary-Scan architecture. Unfortunately, design reuse and heterogeneous libraries do not allow designers to choose the style and the location of the test structures. We propose a splitting process which constructs overlapping testable units around the test structures. Theses testable units can be processed independently by ATPG tools across a network of workstations.

The boundaries of the testable units are the observable and controllable signals of the structural flattened view of the circuit. We identify automatically these signals, using logical and symbolic simulations. Next the circuit is split in testable units, and the overlap factor due to this splitting process is minimized. Gates from Boundary-scan logic, or gates from area with built-in self test are not included in testable units. We can now obtain test vectors and a fault dictionary for each testable unit using a classic ATPG tool.

We merge all the test vectors and the fault dictionaries to obtain one global test sequence and one global fault dictionary for the circuit. Test vectors from testable units are run in parallel to save test length.

A set of tools based on our test method has been realized in the framework of the ALLIANCE CAD system, using HITEST, a commercial product, as the APTG tool. With these tools, we have been able to process a real circuit which integrates macro-cells.

Keywords

VLSI testing, Boundary-scan, built-in self-test, design reuse, automatic partitioning, ATPG tool, distribution.

Résumé

Nous nous plaçons dans le cadre d'un circuit possédant déjà des structures de test. Nous avons observé que les circuits intégrés d'aujourd'hui possèdent tous des chemins de scan-path, certains utilisent le test intégré ou des macro-cellules, et la plupart ont une architecture Boundary-Scan. Dans ces circuits, l'utilisation de blocs issus de bibliothèques ou de fabricants différents, ou encore la réutilisation de composants, ne permet pas de maîtriser l'implantation des structures de test. Nous proposons d'effectuer un découpage du circuit autour de ses structures de test en unités testables indépendantes de complexité raisonnable vis à vis des outils de génération de vecteurs de test.

Nous effectuons une identification automatique des points observables et contrôlables dans la représentation structurelle du circuit à l'aide de simulations. Puis nous découpons le circuit en unités testables. Puisque nous ne maîtrisons pas l'emplacement des points observables et contrôlables du circuit, ces unités testables peuvent être recouvrantes. Nous essayons de rendre le découpage le moins recouvrant possible. Nous n'incluons pas dans ces unités testables les portes venant de l'architecture Boundary-Scan, ni les blocs incluant leur propre mécanisme de test. Les unités testables étant indépendantes, nous pouvons distribuer la génération sur un réseau de stations de travail.

Nous proposons une technique d'assemblage des vecteurs de test et des dictionnaires de fautes issus de la génération, afin d'obtenir une séquence unique et un dictionnaire unique pour l'ensemble du circuit, sans effectuer une simulation de fautes globale. Nous parallélisons les vecteurs issus d'unités testables non-recouvrantes ainsi que ceux issus d'unités testables peu recouvrantes.

Nous avons realisé un prototype logiciel s'appuyant sur la chaîne ALLIANCE et sur le générateur de vecteurs de test commercial HITEST. Ce prototype nous a permis de confronter notre méthode à des circuits réels.

Mots-clés

test des circuits intégrés, Boundary-scan, test intégré, réutilisation de composants, découpage automatique, génération de vecteurs de test, distribution.

Discipline : Informatique

Intitulé et adresse du laboratoire :

Laboratoire d'Informatique de Paris 6 (LIP6), équipe Architecture des Systèmes Intégrés et Micro-Informatique (ASIM), 55-65 213, université Pierre et Marie Curie, 4 place Jussieu, 75252 Paris Cédex 05, France.