ACADEMIE DE MONTPELLIER

UNIVERSITE MONTPELLIER II

SCIENCES ET TECHNIQUES DU LANGUEDOC

THESE

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITE MONTPELLIER II

Discipline : Génie Informatique, Automatique et Traitement du Signal

Formation Doctorale : Systèmes Automatiques et Microélectroniques

Ecole Doctorale : Information, Structures, Systèmes

COMPRESSION DE DONNÉES DE TEST POUR ARCHITECTURE DE SYSTÈMES INTÉGRÉS BASÉE SUR BUS OU RÉSEAUX ET RÉDUCTION DU TEMPS DE TEST

---

présentée et soutenue publiquement

par

Julien DALMASSO

Le 01 Octobre 2010

JURY

M. Yves Bertrand, Professeur des Universités, LIRMM, Université de Montpellier II	Président
M. Bruno ROUZEYE, Professeur des Universités, LIRMM, Université de Montpellier II	Directeur de thèse
Mme. Marie-Lise FLOTTES Chargée de recherches CNRS, LIRMM, Université de Montpellier II	Co-Directeur de thèse
M. Luigi CARRO, Professeur des Universités, Instituto de Informatica, UFRGS Porto Allegre - Brésil	Rapporteur
M. Emmanuel SIMEU, Maître de Conférences HDR, TIMA, Université Joseph Fourier Grenoble	Rapporteur
M. Habib MEHREZ, Professeur des Universités, LIP6, Université Pierre et Marie Curie Paris 6	Examinateur

Il n'est pas de savoir sans discours.
[Jacques Lacan]

C'est dans la pratique qu'il faut que l'homme prouve la vérité.
[Karl Marx]

Ce n'est pas tant l'intervention de nos amis qui nous aide mais le fait de savoir que nous pourrons toujours compter sur eux.
[Epicure]

Voici l'exercice habituel des remerciements. Et comme tout le monde, je m'excuse par avance pour les éventuels oublis que je pourrais commettre lors de cette tâche. Que tous ces malencontreux oubliés acceptent mes remerciements implicites pour leur collaboration, leur soutien ou leur simple présence au cours de ma thèse.

Commençons par le commencement. Je remercie Michel Robert, directeur du LIRMM pendant mes années de présence au sein du laboratoire ainsi que Lionel Torres et Marie-Lise Flottes (nous y reviendrons), responsables du département de microélectronique du LIRMM, de m'avoir accueilli au sein du laboratoire et pour toute l'aide qu'ils ont pu me fournir tout au long de ma thèse.

Je tiens aussi à remercier ici chaleureusement les membres de mon jury de thèse, à commencer par mes deux rapporteurs : Emanuel Simeu (Maître de Conférences HDR à l'université Joseph Fourier – TIMA, Grenoble) et Luigi Carro (Professeur de l'Universidade Federal do Rio Grande do Sul, à Porto Alegre, Brésil) qui ont "subi" mon travail de plein fouet. Messieurs, ce fut un honneur et un plaisir de soumettre mes travaux à votre expertise et à votre science. Je remercie également chaleureusement Habib Mehrez (Professeur de l'Université Pierre et Marie Curie Paris VI - LIP6) d'avoir accepté de faire partie de mon jury de thèse et Yves Bertrand (ancien Directeur de l'UFR Science et Professeur de l'Université de Montpellier II) d'avoir accepté de présider ce jury. Finalement, ce fut un très bon moment en votre compagnie à tous !

Il convient ensuite, et ils le méritent tous amplement, de remercier mon équipe de recherche. A commencer par Bruno Rouzeyre (Professeur de l'université de Montpellier II – LIRMM, Montpellier), directeur de thèse, chef inébranlable, et véritable pilier de mon travail. Pour son savoir, sa conscience scientifique, son expérience, mais aussi, à sa façon, son soutien sans faille, sa confiance et son appui dans les différentes épreuves que j'ai traversées, je le remercie du fond du cœur ! Je tiens aussi à remercier particulièrement Marie-Lise Flottes (Chargée de recherches - LIRMM, Montpellier), co-encadrante de ma thèse, bonne-vivante et véritable grande sœur de l'équipe. Pour ses relectures pointilleuses, nos brainstormings endiablés, mais aussi et surtout pour son écoute, sa patience, sa disponibilité et son humanité qui m'ont aidé au cours de ma thèse, Marie, merci mille fois ! Je tiens aussi à remercier Giorgio Di Natale, l'ami transalpin, qui a rejoint l'équipe au cours de ma thèse, et qui m'a apporté sa joie de vivre, sa simple présence rassurante, sa bienveillance, et qui, avec son collègue Alberto Bosio, arrivé également au Laboratoire en cours de route, ont réussi à me réconcilier avec le peuple italien et

mes origines. Grazie mille à vous deux ! Je poursuis en remerciant chaleureusement Erika Cota (de l'Universidade Federal do Rio Grande do Sul, à Porto Alegre au Brésil) dont la collaboration fructueuse et chaudement amicale a donné lieu à tout un chapitre de cette thèse. Muito obrigado !

Je remercie également les différents chercheurs qui ont croisé ma route au cours de ma thèse que ce soit pour des raisons professionnelles ou amicales. Je pense tout particulièrement à Mariane Comte, la voix de Lirmmac, le groupe de rock éphémère du LIRMM, mais aussi une amie dans mes années passées au laboratoire, toutes mes félicitations à toi et Giorgio pour votre récent heureux événement ! Mes remerciements vont également à Laurent Latorre, Pascal Benoit, Frédéric Comby, également membres éminents de Lirmmac et qui ont su répondre présent quand j'en avais besoin. Encore et toujours des remerciements aux personnes avec qui j'ai collaboré au cours de mes enseignements à l'université de Nîmes. Je pense particulièrement à Jean Triboulet et José Rodrigues avec qui j'ai énormément appris et qui m'ont parfaitement accueilli dans leur équipe enseignante. Merci encore en vrac à Sébastien Krut, Arnaud Virazel, Serge Bernard, Gilles Sassatelli, Etienne Dombre, et pardon encore à ceux que j'oublie.

J'en viens maintenant à mes très chers collègues thésards. On va commencer par les honorés membres de mon bureau dans ses différentes versions : Alex Ngouanga, l'ami réunionnais, Jean-Denis Techer, Julien Vial, Amine Dehbaoui. Et plus particulièrement encore, deux rencontres magnifiques d'outre atlantique et de l'hémisphère sud de surcroit, mes amis brésiliens Daniel Mesquita et Erik Schüler. J'ai découvert un nouveau pays avec vous et uniquement des gens merveilleux. Vous êtes vraiment mes plus belles rencontres dans cette période et j'ai hâte de vous revoir dans votre pays ! Muito obrigado é até logo! J'en profite pour associer à ces remerciements votre compatriote Gabriel Marchesan qui m'a apporté son aide précieuse sur la fin de ma thèse.

Je poursuis ma liste de remerciements avec les thésards qui ont croisé de près ou de loin ma route, en commençant par Alexandre Rousset et Gaël Pages, mes collègues de pause, mais bien plus encore, et aussi Marion Doulcier, Laurent Larguier, Philippe Amat, Aurélien Noce, Olivier Leman, Séverine Riso, etc… et Pardon une fois de plus à ceux que j'oublie.

Je terminerai cette longue liste de remerciements par ceux qui ont été les plus proches de moi pendant cette période. Je veux parler de Florence Jacquey qui a partagé ma vie durant les bons et les mauvais moments de la thèse, nous soutenant l'un et l'autre à tour de rôle et partageant nos désirs, nos espoirs et nos rêves… Bref, merci à elle d'avoir été là, de m'avoir soutenu, rassuré et encouragé tout au long de cette période.

Et pour finir, je tiens à remercier tout particulièrement ma famille: mes parents, ma sœur, mon beau-frère et ma grand-mère. On sait que la famille est toujours là pour vous, mais c'est toujours émouvant de voir à quel point c'est vrai quand on traverse des mauvaises passes. Et ils ne m'ont pas fait défaut une seule fois durant tout ce qui m'est arrivé ces dernières années et je tiens à les assurer de mon affection la plus tendre et la plus sincère, même si je ne le leur dis peut-être pas assez tous les jours. Un énorme merci à vous tous, ma famille !

Compression *: n.f. (lat.compressio, -onis)* ***1.*** *Action de comprimer quelque chose, résultat de cette action.* ***— 2.*** *Réduction, diminution forcée.* ***— Informatique.*** *Compression numérique : technique permettant de réduire le volume des fichiers (en bits) afin de les stocker sur un volume moindre ou de réduire le débit (en bit/s) des données numérisées afin de les transmettre en un temps plus court.*

Sommaire

Introduction 9

Chapitre 1: Motivation 13

1.1. Test des Circuits intégrés 15

1.1.1. Pourquoi le test ? 15

1.1.2. Principe et mise en œuvre du test logique 16

1.1.2.1. Modèles de fautes 17

1.1.2.2. Vecteurs de test – Séquence de test 17

1.1.2.3. Taux de couverture 19

1.1.2.4. Structure de Test 19

1.1.2.5. Testeur 20

1.1.3. Difficultés et coûts 21

1.2. Problématique 22

1.2.1. Réduction des coûts 22

1.2.2. Compression et architecture de test 22

1.2.2.1. Architecture Scan 22

1.2.2.1.1. Chaîne de scan unique 24

1.2.2.1.2. Chaînes de scan multiples 25

1.2.2.1.3. Compression de données 27

1.3. Références bibliographiques 30

Chapitre 2: Compression des données de test 31

2.1. Etat de l'art 35

2.1.1. BIST 36

2.1.1.1. BIST STUMPS 37

2.1.1.2. Séquences déterministes modifiées 38

2.1.1.3. Séquences aléatoires modifiées 39

2.1.1.4. Procédés non basés sur des LFSR 42

2.1.1.5. Conclusion 45

2.1.2. Méthodes de compression verticale 45

2.1.2.1. Codages Statistiques 45

2.1.2.2. Codages RLE 47

2.1.2.3. Codage FDR 48

2.1.2.4. Codes de Golomb 49

2.1.2.5. Conclusion 50

2.1.3. Méthodes de compression horizontale 50

2.1.3.1. Méthode non compressive: Sérialisation des données 51

2.1.3.2. Diffusion (Broadcasting) 52

2.1.3.2.1. Reconfiguration Statique 52

2.1.3.2.2. Reconfiguration dynamique 54

2.1.3.2.3. Méthodes "Illinois scan" 56

2.1.3.2.4. Méthode industrielles (Synopsys, Syntest) 57

2.1.3.3. Décompresseur linéaire 60

2.1.3.3.1. Décompresseur linéaire à structure combinatoire 61

2.1.3.3.2. Décompresseur linéaire à structure séquentielle statique 62

2.1.3.3.3. Décompresseur linéaire à structure séquentielle dynamique 63

2.1.3.3.4. Méthodes industrielles (Mentor) 64

2.1.3.4. Méthode par dictionnaire 65
2.1.4. Adéquation Compression des données de test et Systèmes sur puces 67
2.1.4.1. Problèmes liés au test des IP 68
2.1.4.2. Problèmes liés au test des SoCs 69
2.2. Nouvelle méthode de compression des données de test. 70
2.2.1. Objectifs: 70
2.2.2. Architecture proposée 71
2.2.2.1. Idée Initiale 71
2.2.2.2. Amélioration 72
2.2.2.3. Décompresseur sur puce 73
2.2.2.3.1. Synchronisation et commande 74
2.2.2.3.2. Fonctionnement 76
2.2.3. Algorithme de compression 82
2.2.3.1. Assignation initiale 83
2.2.3.2. Assignations intermédiaires 83
2.2.3.3. Attribution des canaux testeur 85
2.2.3.4. Compression 85
2.2.4. Performance 86
2.2.4.1. Equations temps de test 86
2.2.4.2. Equation volume de données 86
2.3. Résultats et comparaisons 87
2.3.1. Comparaison avec une méthode standard de sérialisation des données 87
2.3.1.1. Principe et description 87
2.3.1.2. Performance 88
2.3.2. Résultats expérimentaux 88
2.3.2.1. Procédure expérimentale 89

2.3.2.2. Analyse des résultats pour un ou 2 circuits complets 89

2.3.2.3. Généralisations aux autres circuits 92

2.4. Conclusion 93

2.5. Références bibliographiques 95

Chapitre 3: Compression des données de test et SoCs 99

3.1. Test des SoCs : principe 101

3.1.1. Architecture de test : la norme 1500 101

3.1.2. Définition de l'ordonnancement du test des cœurs 102

3.1.3. Problèmes induits par le test multi cœurs 105

3.2. Etat de l'art 105

3.2.1. Méthode d'optimisation 105

3.2.1.1.1. Conception des wrappers 105

3.2.1.1.2. Conception du TAM 106

3.2.2. Introduction de la compression 107

3.3. Méthode proposée 108

3.3.1. Présentation 108

3.3.1.1. Compression et temps de test 108

3.3.1.2. Compression et test en parallèle 109

3.3.1.3. Compression et architecture de test 110

3.3.2. Problème et notations 111

3.3.3. Algorithmes 112

3.3.3.1. Temps de test des cœurs 113

3.3.3.2. Exploration des partitions 115

3.3.3.3. Assignation des cœurs et Ordonnancement 116

3.3.3.4. Illustration 118

3.4. Résultats 121
3.4.1. Procédure expérimentale 121
3.4.2. Résultats et analyse 123
3.4.2.1. Wrappers fixés. 123
3.4.2.2. Wrappers libres. 125
3.5. Conclusion 126
3.6. Références bibliographiques 128

Chapitre 4: Compression des données de test et NoCs 131

4.1. Réutilisation du NoC pour le test 135
4.1.1. Réseau sur puce: définition 135
4.1.2. Pourquoi réutiliser le NoC 136
4.1.3. Réutilisation du NoC pour le test 136
4.1.3.1. Modifications structurelles 136
4.1.3.2. Format des données 137
4.1.4. Limitation du test avec un NoC : les ports d'entrée 138
4.1.5. Réduction du temps de test 138
4.2. Etat de l'art 140
4.2.1. Optimisation de l'architecture 140
4.2.2. Test préemptif 140
4.2.3. Test non-préemptif 141
4.3. Méthode proposée 141
4.3.1. Architecture de Test 142
4.3.2. Impact de la compression dur les données de test 143
4.3.3. Problème et notations 144

4.3.4. Test non-préemptif et compression 145

4.3.4.1. Exploration architecturale 145

4.3.4.2. Ordonnancement du test non-préemptif 146

4.3.4.3. Exemple applicatif 147

4.4. Résultats 150

4.4.1. Procédure expérimentale 150

4.4.2. Résultats et analyse 151

4.5. Conclusion 155

4.6. Références bibliographiques 156

Conclusion 159

Liste des publications 163

Table des figures 165

Table des tableaux 169

Annexe A 171

Annexe B 173

Index 175

Introduction

Aujourd'hui, les systèmes électroniques sont présents dans tous les domaines de la vie courante, que ce soit les ordinateurs, les téléphones portables, les appareils multimédia ou encore les applications de domotique. Pour y parvenir, les fabricants ont réduit les prix de ces systèmes et ont proposé en très grande séries des éléments fiables et performants. Des efforts toujours plus importants ont été apportés à la conception, à la vérification et à la fabrication de ces systèmes afin de répondre à ces demandes.

De plus, la réduction des motifs de gravure sur silicium autorise la conception de circuits de plus en plus complexes. Selon la loi empirique de Gordon Moore, réévaluée en 1975, le nombre de transistors au sein d'un microprocesseur double tous les deux ans à coût constant. Les fabricants sont donc amenés à fournir des circuits toujours plus complexes, plus performants et plus fiables tout en maintenant des prix abordables.

C'est dans ce cadre que les ingénieurs ont dû imaginer des solutions pour réduire les coûts de fabrication tout en permettant des performances et une fiabilité toujours plus importantes. Un des points sur lesquels les efforts ont été portés est le test de ces circuits. En effet, le test permet de s'assurer de la qualité des produits vendus. Mais surtout, avec la complexité grandissante des circuits, le test devient, d'une part, de plus en plus complexe à mettre en œuvre, et d'autre part, de plus en plus long à pratiquer. Tout ceci induit une augmentation significative de la part du test dans le coût total du produit. Il est alors naturellement apparu que diminuer le coût du test, tout en lui conservant sa qualité, est un des leviers pour réduire les coûts de production.

De nombreux travaux ont été menés dans ce sens. La plupart cherchent à réduire le coût lié au matériel de test et / ou réduire le temps de test. Pour cela, de très nombreuses techniques de compression des données de test ont été proposées pour ce qui est du test des blocs numériques. Le principe général consiste à compresser hors ligne les données nécessaires au test, les stocker sur un testeur externe, puis à les transmettre à un décompresseur intégré sur le silicium qui reproduit les données originales et les fournit au circuit. Ces méthodes se concentrent principalement sur la réduction d'un ou plusieurs des paramètres suivant:

- la quantité de mémoire testeur nécessaire au stockage de ces données de test
- le nombre de lignes requises au transfert de ces données vers le décompresseur sur puce
- le temps d'application de ces données au circuit et de récupération des réponses

Cependant, les techniques de conception en microélectronique ont continué d'évoluer. Ces dernières années ont vu l'apparition du paradigme des systèmes sur puces. Pour résumer, au lieu d'utiliser plusieurs circuits pour réaliser une fonction, il s'agit de regrouper au sein d'un système complet, embarqués sur une même puce, les différents circuits, ou cœurs, nécessaires à la réalisation de cette fonction tels que, par exemple, de la mémoire, un ou plusieurs microprocesseurs, des périphériques d'interface, etc. Ce type de structure, appelée "système sur puce" (SoC pour system on Chip), augmente significativement la souplesse de production de ces systèmes. En effet, le fabricant peut acheter des parties du système sous forme de blocs IP (Intellectual Property) à des sous-traitants, lui économisant ainsi les coûts de conception de ces cœurs.

Toutefois, l'apparition de cette nouvelle approche de conception pose des problèmes concernant le test de ces structures. Le fait d'intégrer tous les cœurs au sein d'une même puce compromet fortement les possibilités de transfert des données de test aux cœurs puisque le système dispose généralement d'un nombre restreint de broches d'entrée/sortie ne donnant pas accès à l'ensemble des entrées /sorties de tous les cœurs. De nombreux travaux de recherche ont alors porté sur ce problème et ont abouti au final à la création de l'architecture de test définie par la norme IEEE 1500. Il s'agit d'ajouter au système un bus de test (TAM pour Test Acces Mechanism) relié aux entrées/sorties du système, sur lequel les cœurs vont venir se connecter via un anneau d'isolation (wrapper) leur permettant de permuter entre le mode normal de fonctionnement et le mode test. De plus, cette structure permet le test simultané de plusieurs cœurs ce qui peut nettement réduire le temps de test du système.

Plus récemment, une nouvelle structure de système a été proposée. Baptisée "réseau sur puce" (NoC pour Network on chip), elle reprend les principes des réseaux de communication pour assurer le transfert des données entre les cœurs: à chaque nœud du réseau on trouve un routeur, et sur ces routeurs sont positionnés les cœurs. Les données quittent le cœur de départ, et circulent de routeur en routeur jusqu'au cœur de destination.

Ici aussi il a fallu trouver des solutions permettant de réaliser le test de ces systèmes. La structure NoC étant récente et pour l'instant assez peu développée au niveau industriel, peu de travaux ont été menés sur le thème du test dans ces réseaux. Les systèmes basés sur les NoC restant coûteuse à mettre en œuvre mais permettant des transferts de données entre deux nœuds quelconque du réseau, il est alors apparu particulièrement intéressant de réutiliser ces réseaux pour l'acheminement des données de test à quelques modifications des éléments du réseau près, un peu à l'image de ce qui est fait pour les architectures IEEE 1500.

La démocratisation de ces structures de systèmes, qu'elles soient sous forme SoC ou NoC, a poussé les recherches encore plus loin pour réduire le coût du test de ces systèmes. C'est pourquoi ces dernières années ont vu apparaître des techniques de compression de données de test associées aux architectures de test des systèmes. Cependant, les systèmes imposent de nouvelles contraintes. Déjà, l'utilisation d'IP au sein des systèmes récuse un grand nombre des techniques de compression, qu'elles soient académiques ou industrielles puisqu'elles reposent sur la connaissance des

informations structurelles des circuits, informations qui ne sont pas nécessairement fournies dans le cas d'IP.

C'est dans ce cadre que se situent les travaux présentés dans ce manuscrit. Dans un premier temps, en partant du constat que très peu de méthodes de compression des données de test conviennent au test des systèmes, qu'ils soient SoC ou NoC, une nouvelle technique de compression compatible avec le test des systèmes est proposée. Un cahier des charges strict a été mis en place, tenant compte su fait que la structure des blocs IP n'est pas forcément fournies au concepteur, les outils de génération de vecteurs de test ou de simulation de fautes ne sont pas utilisable pour adapter la séquence de test à la technique de compression utilisée. Celle-ci doit donc pouvoir accepter en entrée n'importe quelle séquence fournie par le vendeur du bloc IP.

Puis, dans un second temps, deux méthodes de test des systèmes sont proposées. La première est relative au test des systèmes SoC utilisant l'architecture de test IEEE 1500. Cette méthode utilise conjointement un ordonnancement du test des cœurs du système avec une technique de compression horizontale. L'idée générale est d'augmenter le parallélisme du test du système à coût matériel constant afin de pouvoir tester plus de cœur simultanément et ainsi réduire le temps de test total du système. La seconde méthode est, elle, relative au test des systèmes NoC. Ici aussi le principe est d'utiliser la compression pour augmenter le parallélisme du test à coût constant mais, avec les spécificités des NoC, l'objectif est d'augmenter le nombre de ports de test sans ajouter réellement d'entrées/sorties au système. En d'autres termes, cela permet d'utiliser moins d'entrées/sorties pour constituer un port afin de pouvoir constituer plus de ports de test.

Ce manuscrit est organisé en quatre chapitres. Le premier chapitre reprend rapidement les généralités du test des circuits et pose la problématique du test dans le coût total d'un circuit. Dans le deuxième chapitre, après un état de l'art détaillé de la compression des données de test, une nouvelle technique est proposée, compatible avec le test des IP et des systèmes. A la fin de ce chapitre, des résultats détaillés obtenus par cette technique sur des circuits ISCAS'89 sont présentés et analysés. Le troisième chapitre présente une méthode inédite utilisant un ordonnancement du test des cœurs d'un système combiné à une technique de compression des données de test. Les résultats présentés dans ce chapitre sont ceux obtenus avec la technique de compression proposée dans chapitre précédent, mais n'importe quelle autre technique de compression répondant aux critères du test des systèmes peut être utilisée à la place. Le quatrième et dernier chapitre, à l'instar du précédent, présente une méthode de test des NoC combinant ordonnancement du test des cœurs et compression des données de test. Une fois encore, les résultats sont obtenus avec notre technique de compression mais la méthode reste valide avec une autre solution répondant aux critères imposés par le test des systèmes. Enfin nous tirons les conclusions sur les travaux menés dans cette thèse et brossons quelques perspectives pour des travaux futurs.

Chapitre I:

Motivations

1.1. Test des Circuits intégrés 15

1.1.1. Pourquoi le test ? 15

1.1.2. Principe et mise en œuvre du test logique 16

1.1.2.1. Modèles de fautes 17

1.1.2.2. Vecteurs de test – Séquence de test 17

1.1.2.3. Taux de couverture 19

1.1.2.4. Structure de Test 19

1.1.2.5. Testeur 20

1.1.3. Difficultés et coûts 21

1.2. Problématique 22

1.2.1. Réduction des coûts 22

1.2.2. Compression et architecture de test 22

1.2.2.1. Architecture Scan 22

1.2.2.1.1. Chaîne de scan unique 24

1.2.2.1.2. Chaînes de scan multiples 25

1.2.2.1.3. Compression de données 27

1.3. Références bibliographiques 30

Dans ce chapitre, nous exposons la problématique du test des circuits intégrés, son rôle, son utilité, ses contraintes, ainsi que sa position dans un flot de conception industriel. Nous verrons ensuite les difficultés liées à sa mise en place, et son coût.

Dans un second temps, nous poserons le problème de la réduction de ces coûts, puis nous évoquerons les pistes envisagées.

1.1. Test des Circuits intégrés

1.1.1. Pourquoi le test ?

Le principe de base du test d'un circuit intégré est de mettre en évidence un mauvais fonctionnement éventuel du circuit dû à une défaillance physique. Cette mise en évidence s'effectue par deux types de tests. Le premier consiste à effectuer des mesures paramétriques (courant, tension, temps de montée et de descente, ...). Ce test paramétrique fait intervenir des techniques de métrologie particulières et ne sera pas abordé dans le cadre de cette thèse. Le deuxième, qui nous intéresse plus particulièrement ici, consiste à injecter des données logiques dans le circuit et à comparer les réponses du circuit aux valeurs attendues. Ce test logique permet de savoir si tous les éléments (portes logiques, bascules, fils…) du circuit fonctionnent correctement.

Les exigences en termes de qualité et de prix n'ont eu de cesse d'augmenter avec la démocratisation des systèmes électroniques dans l'industrie et le grand public. C'est pourquoi le test occupe désormais une place importante dans le cycle de fabrication d'un circuit intégré pour faire en sorte que le rendement de fabrication (yield) soit le meilleur possible (le rendement est le rapport entre le nombre de circuits bons et le nombre de circuits total). En effet, plus le nombre de circuits rejetés est grand, plus le fabricant perd de l'argent. D'autre part, il ne faut pas que le coût du test prenne une part trop importante dans le prix de fabrication d'un circuit afin de minimiser l'impact sur le prix de vente final.

Le test peut intervenir lors de différentes phases de la fabrication d'un circuit intégré sous différentes formes et avec différents objectifs. Les deux principales étapes où le test intervient sont :

- Une fois que le wafer (plaque de silicium) contenant un très grand nombre de circuits intégrés a été obtenu, chaque circuit intégré est testé individuellement directement sur le wafer par l'intermédiaire d'une machine à pointes et les circuits défectueux sont marqués (Figure 1.1). Ce test permet de détecter notamment les défauts physiques liés aux procédés technologiques. L'objectif est de dissocier les circuits considérés comme sains (sans défauts) des circuits dits fautifs (ne réalisant pas la bonne opération) afin de ne pas les passer dans la phase d'encapsulation (mettre le circuit en silicium dans son boîtier plastique).

- Après l'encapsulation de chaque circuit intégré dans leur boîtier respectif. Les circuits sont testés après avoir été vieillis artificiellement grâce a des procèdes spécifiques (burn-

in). L'objectif de ces tests est de mettre en évidence des défaillances pouvant apparaître avec le temps.

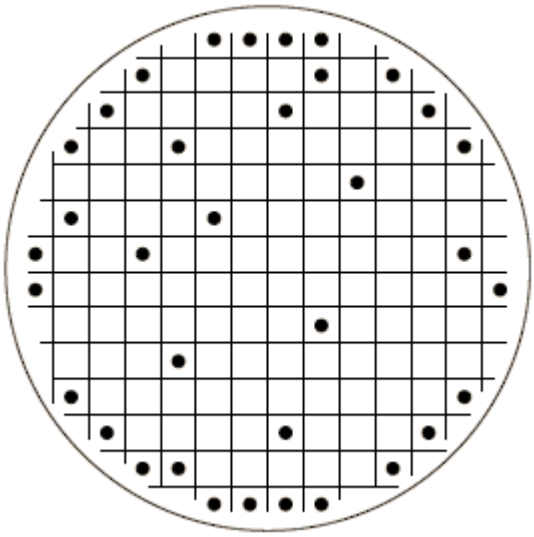


Figure 1.1 : Wafer dont les circuits défectueux sont marqués

1.1.2. Principe et mise en œuvre du test logique

L'objectif du test est de détecter si les circuits ont un comportement conforme à ce qui est attendu. L'idée est de comparer la réponse réelle à un stimulus donné à celle attendue d'un circuit sain soumis au même stimulus. S'il existe une différence, le circuit comporte un ou plusieurs **défauts**.

> **Définition 1 :** Un défaut (aussi appelé défaillance) est la différence, non voulue, entre l'implantation réelle et l'implantation désirée du système [Bush00].

Les défauts dans les circuits VLSI ("Very Large Scale Integration") sont typiquement les suivant [Howe81] :

- défauts du procédé de fabrication : rupture d'oxyde, transistor parasite, …
- défauts du matériau : impureté, fissure du matériau, …
- défauts du boîtier après encapsulation : dégradation des contacts, étanchéité du boîtier

Les défauts de fabrication et les pannes physiques (survenant pendant la vie du circuit et non présentes en sortie de chaîne de fabrication) sont aussi appelés fautes physiques.

Les fautes physiques sont observées par la manifestation d'une erreur :

> **Définition 2 :** Une **erreur** est la manifestation observable d'un défaut. Elle peut se produire en présence de certains stimuli [Bush00].

Parce que l'analyse des fautes physiques est très complexe, du fait de leur diversité et de leur nombre, et que ces fautes dépendent de la technologie employée, des modèles de fautes logiques ont été proposés pour les représenter.

La modélisation des fautes physiques par des modèles logiques simplifie l'analyse car d'une part, le problème devient logique plutôt que physique et, d'autre part, sa complexité diminue puisque de nombreuses fautes physiques peuvent être modélisées par une même faute logique. Enfin certains modèles logiques sont indépendants de la technologie employée, par conséquent les techniques de recherche des stimuli de test et d'identification de fautes (diagnostic) restent valides malgré les changements de technologie.

> **Définition 3 :** Une **faute logique** correspond à la représentation logique d'un défaut ou d'une panne suivant un modèle donné [Bush00].

1.1.2.1. Modèles de fautes

Le premier modèle de faute ayant été utilisé, en particulier pour les circuits logiques discrets, est **le modèle de collage** ("stuck–at") appliqué sur un modèle de description du circuit en portes logiques. Ce modèle consiste à considérer les collages permanents à 0 et à 1 des différentes lignes de la description. Le collage d'une ligne représente le fait qu'une ligne est toujours à la valeur logique 1 ou 0, indépendamment de la valeur qu'elle reçoit: elle est soit reliée en permanence à la masse, soit à la ligne Vcc.

Ce modèle permet de disposer d'un modèle de représentation des défaillances manipulable d'un point de vue logique et correspond d'autre part à une très bonne couverture des défaillances susceptibles d'affecter ce type de circuit (majorité des défaillances affectant les connexions entre la puce et le boîtier). Il présente les caractéristiques suivantes :

- il permet de représenter de nombreux types de défaillances physiques,
- il est indépendant de la technologie
- il permet d'utiliser tous les résultats obtenus en algèbre de Boole pour déterminer des séquences de test
- les tests produits à partir d'une hypothèse de collage détectent aussi de nombreux défauts physiques pourtant non modélisables par un collage
- le taux de couverture vis à vis des fautes de collage est une métrique établie pour la communication entre fournisseurs et clients

Un autre modèle de défaut est le **modèle de court–circuit** ("Bridging fault"). Il a été introduit en raisonnant toujours sur un modèle de représentation à portes du circuit. Pour ce modèle, on considère qu'il y a contact entre deux (ou plusieurs) équipotentielles de la représentation à portes. Il existe beaucoup d'autres types de modèles comme les fautes de délai, de court-circuit [Abra83] … Mais le modèle le plus couramment utilisé pour les circuits logiques reste pour l'instant le collage.

1.1.2.2. Vecteurs de test – Séquence de test

L'objectif du test est de vérifier si le circuit intégré comporte une ou plusieurs de ces fautes (fautes représentant un défaut physique). Dans un premier temps, il faut donc établir la liste des fautes possibles dans le circuit. Ensuite, pour chaque faute, il faut appliquer aux entrées du circuit sous test un stimulus (ensemble de valeurs logiques), aussi appelé **vecteur de test**, capable de mettre en évidence la présence de fautes. Pour cela, les valeurs logiques sur les sorties du circuit sous test sont comparées à celles d'un circuit sain. L'ensemble des vecteurs correspondant à une liste de fautes est appelé **séquence de test**.

Ces vecteurs de test peuvent être obtenus à l'aide d'outils de génération automatique de vecteurs (**ATPG** pour Automatic Test Pattern Generation) ou à l'aide d'outils de simulation de fautes. Lorsqu'un vecteur de test est identifié pour une faute donnée, on dit que cette faute est testable.

A contrario, lorsqu'il n'a pas été possible de trouver de façon manuelle ou automatique des stimuli de test, la faute peut être qualifiée de deux manières différentes [Bush00]:

- **non détectable**, si l'on a pu faire la preuve qu'il n'existe aucun stimuli de test pour lequel une erreur se produit en présence de cette faute (cas des fautes affectant des connexions redondantes)
- **non testable**, si l'on ne peut prouver l'inexistence d'un tel stimulus.

La longueur d'une séquence de test dépend de la façon dont elle a été établie. Il est intéressant de générer les séquences les plus courtes possibles puisque le temps de test (et donc le coût) dépend linéairement de la longueur de la séquence. La plupart du temps, pour tester une seule faute, il n'est pas forcément nécessaire de spécifier toutes les valeurs du vecteur correspondant. Il suffit de propager certaines valeurs jusqu'à la localisation de la faute que l'on veut vérifier et de comparer la réponse du circuit à la réponse théorique. Les valeurs importantes sont assignées à 1 ou 0 selon le besoin, les autres sont ce qu'on appelle des valeurs indéterminées, ou X. Un vecteur contenant des X est aussi appelé **cube de test**. Une technique pour raccourcir les séquences de test est de fusionner plusieurs vecteurs compatibles. Des vecteurs sont dits compatibles si, dans chaque vecteur, les valeurs spécifiées correspondent dans tous les autres vecteurs soit à la même valeur spécifiée, soit à un X. Le vecteur résultant permet donc de tester plusieurs fautes. Cette opération s'appelle la compaction (Tableau 1.1).

Vecteur 1:	1	0	X	X	X	0	X	X	Fautes 1 et 2
Vecteur 2:	1	X	X	X	X	0	1	0	Faute 3
Vecteur 3:	X	0	1	1	X	X	1	X	Fautes 4 et 5
Vecteur résultant:	1	0	1	1	X	0	1	0	Fautes 1, 2, 3, 4 et 5

Tableau 1.1 : Compaction de vecteurs

Il est possible qu'après cette opération, des X subsistent. Une dernière étape consiste à remplir aléatoirement ces valeurs. La simulation de ces vecteurs désormais totalement spécifiés permet alors éventuellement de couvrir des fautes qui initialement ne l'étaient pas. Il est alors possible que certains vecteurs, dont les fautes couvertes sont désormais détectées par d'autres vecteurs totalement spécifiés, puissent encore être retirés de la séquence. On obtient alors une séquence minimisée en termes de nombre de vecteurs.

1.1.2.3. Taux de couverture

Il se peut qu'une séquence de test ne couvre pas complètement l'ensemble des fautes de la liste. Le rapport entre le nombre de fautes détectées et le nombre de fautes total est appelé **taux de couverture de fautes** [Bush00]. Il est défini par la formule suivante:

$$Tc = \frac{\text{nombre de fautes detectées}}{\text{nombre de fautes total}}$$

Le taux idéalement visé est bien sûr 100% mais il est couramment admis qu'un taux de couverture de fautes de 98% est acceptable.

Lorsque le taux de couverture atteint par une séquence est trop faible (inférieur à 95% par exemple) une alternative s'offre au concepteur : soit consacrer davantage de temps à la recherche de nouveaux stimuli, soit modifier le circuit de façon à faciliter l'établissement d'une séquence de bonne qualité.

La complexité des circuits actuels en termes de nombre d'éléments (portes logiques), profondeur séquentielle, présence de cycles et la présence de fautes non détectables rendent souvent difficile la première solution. On utilise donc des techniques de conception en vue du test (DFT pour Design For Testability) comme la technique de scan appliquée aux circuits séquentiels par exemple.

1.1.2.4. Structure de Test

Pour le test d'un circuit intégré, le principe général est d'acheminer les données de test aux entrées du circuit (les vecteurs de test), et de récupérer réponses de test aux sorties du circuit. Ce concept est illustré en Figure 1.2. L'architecture dédiée au test des circuits intégrés est composée de deux éléments principaux qui sont :

- **La source :** désigne le matériel fournissant au circuit les vecteurs de test. Il peut s'agir d'un testeur industriel externe ou d'une structure de test interne au circuit.
- **La cible ou sink :** correspond à l'analyseur de réponses aux données appliquées. De même, il peut être interne ou externe au circuit, sous la forme d'une structure intégrée ou d'un testeur.

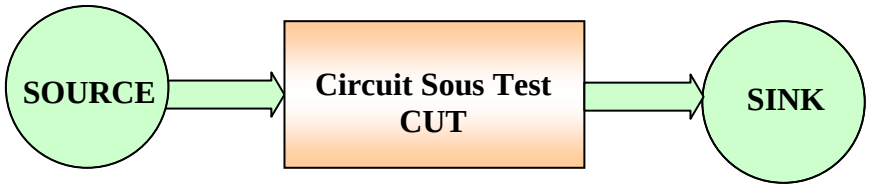


Figure 1.2 : Structure de test

1.1.2.5. Testeur

L'acheminement de ces vecteurs de test aux entrées du **circuit sous test** (également reconnu sous le nom de Circuit Under Test en Anglais ou **CUT**) et la récupération des réponses aux sorties se fait par l'intermédiaire d'une machine appelée **testeur** (Figure 1.3) ou ATE en anglais pour Automatic Test Equipment.



Figure 1.3 : Testeur Agilent 93000

La séquence de test complète, contenant les vecteurs ainsi que les réponses attendues, est stockée dans la mémoire du testeur. Les vecteurs sont appliqués au circuit par les canaux de test (un canal par entrée). Les réponses réelles du circuit sont ensuite retournées au testeur où elles sont comparées avec les réponses attendues (cf. Figure 1.4).

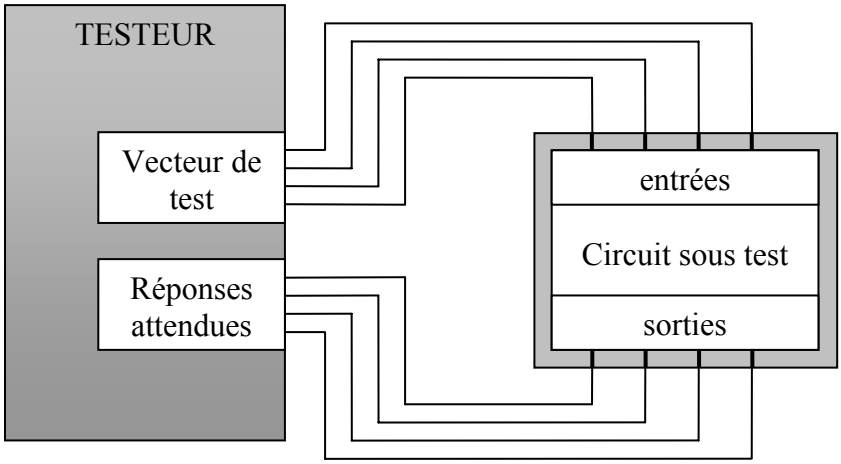


Figure 1.4 : Connexion du testeur au circuit sous test

La part du coût d'un circuit relative au test est liée principalement au coût du testeur et au temps d'occupation du testeur par le circuit. Deux facteurs limitent l'utilisation d'un testeur :

- **La profondeur mémoire :** Le flot de données à transmettre aux circuits peut être très important. Plus il y a de vecteurs à charger, plus long sera le test. de plus, si la place mémoire requise est supérieure à la quantité de données que peut stocker le testeur, le transfert de la séquence complète doit s'effectuer en plusieurs temps par rechargements successifs de la mémoire testeur. Le chargement/déchargement des données dans la mémoire augmente encore le temps d'occupation du testeur. Il est alors impératif de minimiser la longueur des séquences de test.
- **Le nombre de canaux testeur :** Les circuits à tester ont un nombre d'entrées/sorties qui peut être supérieur au nombre d'entrées/sorties du testeur. Le prix d'une broche de test additionnelle est très élevé. De ce fait, il faut minimiser le nombre d'entrées/sorties du circuit sous test en utilisant des méthodes de réduction du nombre de broches de test.

Toute technique permettant de limiter l'espace mémoire et le nombre de canaux de test nécessaires sans sacrifier à la qualité d'un test est donc utile pour réduire le coût de fabrication des systèmes intégrés.

1.1.3. Difficultés et coûts

Avec l'augmentation constante de la densité d'intégration des circuits, on atteint désormais des chiffres avoisinant le milliard de transistors dans un circuit (Figure 1.5).

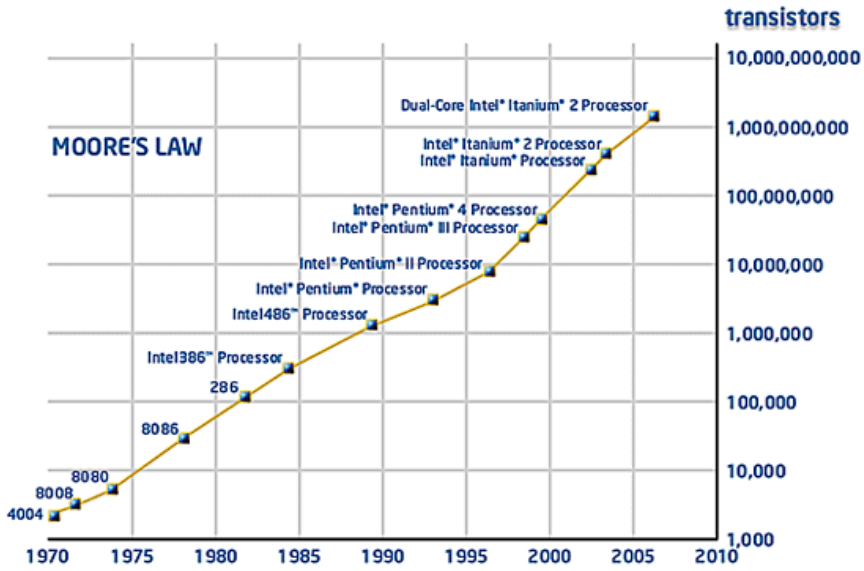


Figure 1.5 : Loi de Moore

Or, plus il y a d'éléments à tester dans un circuit, plus la liste de fautes à vérifier dans ce circuit s'allonge, et donc plus il y a de vecteurs de test. D'autre part, plus le circuit est complexe, plus il est difficile de sensibiliser certaines fautes enfouies au cœur du circuit, et, là encore, plus de

vecteurs de test peuvent être nécessaires. Tous ces vecteurs additionnels se traduisent par une augmentation de la mémoire du testeur nécessaire au stockage de ces vecteurs (coût matériel relatif au testeur) ainsi que par une augmentation du temps d'application de ces vecteurs au circuit (coût en temps de production).

De plus, la rapidité d'évolution de ces circuits nécessiterait une évolution similaire dans le développement des appareils de test. Or, il est impossible de le renouveler à chaque fois que la technologie des circuits intégrés progresse, un testeur coûtant plusieurs centaines de milliers d'euros. Il est toutefois possible d'augmenter la profondeur mémoire d'un testeur ou bien son nombre de canaux maximum mais cela aussi a un coût très élevé.

Tous ces coûts, qu'ils soient matériels ou relatifs au temps du test ont un impact sur le coût final des circuits et sur les marges des entreprises de fabrication de circuits sur silicium.

1.2. Problématique

1.2.1. Réduction des coûts

La réduction des coûts de test est donc devenue un axe primordial de la recherche dans le domaine du test des circuits intégrés, que ce soit au niveau industriel (Synopsys, Mentor, Cadence etc…) ou dans le milieu universitaire. De nombreuses pistes ont été explorées dans le but de réduire les coûts matériels et le temps de test d'un circuit intégré. L'idée principale est d'utiliser un système de compression des données [Toub06] de manière à réduire soit la quantité de mémoire nécessaire au stockage des vecteurs (ce que l'on appellera dans la suite **compression verticale**), soit le nombre de canaux nécessaires au transfert des vecteurs de test aux circuits (que l'on appellera **compression horizontale**). L'idée est de pouvoir faire face à l'évolution des circuits en compensant l'évolution technologique des circuits par rapport à celle des testeurs. Pour cela, il faut optimiser l'utilisation de la bande passante entre le testeur et le circuit sous test: faire passer toujours plus d'information (en l'occurrence des vecteurs de test) avec des ressources matérielles fixées (nombre de canaux de test et mémoire testeur).

1.2.2. Compression et architecture de test

1.2.2.1. Architecture Scan

Dans la suite, nous nous intéressons uniquement aux **circuits numériques séquentiels** puisqu'ils représentent plus de 95% des circuits sur le marché. Un tel circuit se compose de portes logiques et d'éléments de mémorisation (bascules). Les valeurs aux sorties d'un circuit séquentiel dépendent des valeurs appliquées en entrées mais également des valeurs présentes dans les éléments de mémorisation à l'instant précédent. Pour tester une faute, il faut donc 1/ appliquer une séquence de vecteurs sur les entrées pour amener celui-ci dans un état particulier (justification), 2/ appliquer un vecteur qui sensibilise la faute, 3/ appliquer une séquence de vecteurs pour observer l'erreur

produite sur les sorties primaires (propagation). La principale difficulté de cette approche est l'établissement des séquences de justification et de propagation.

Pour contourner cette difficulté, la méthode dite du "**scan path**" a été proposée et est actuellement implantée dans la plupart des séquentiels. Le principe est de considérer les parties de circuits situées entre les bascules comme illustré sur la Figure 1.6, ces parties étant par définition combinatoires. Le test de ce sous ensemble est alors très simple. Le test du circuit complet se résume au test de tous ses sous ensembles. Les données de test doivent être appliquées aux bascules internes ainsi qu'aux entrées primaires. Si le scan est adjacent aux entrées / sorties du circuit, on parle de "**boundary scan**" [IEEE90].

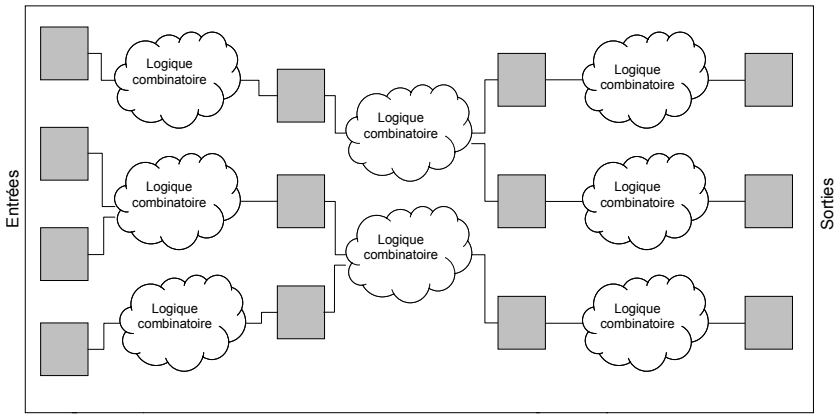


Figure 1.6 : Circuit séquentiel

Dans cette technique, le circuit doit être conçu de manière à présenter deux modes de fonctionnement. Un mode de fonctionnement normal et un mode de test où tous les éléments mémoire – les bascules - sont interconnectés sous la forme d'un (voire plusieurs) registre à décalage appelé **chaîne de scan**. En mode de test, un vecteur de test peut être chargé en mode série à l'intérieur des bascules. Après retour au mode normal, ce vecteur et les données présentes aux entrées primaires du circuit sont appliqués au circuit. Le résultat obtenu (la réponse du circuit au vecteur) est alors stocké dans ces bascules et sur les sorties primaires. La réponse est extraite du circuit en mode série pour être comparée à la réponse correcte.

Les bascules doivent donc être modifiées de façon à supporter le mode de test. Il s'agit en fait d'ajouter une partie multiplexage (Figure 1.7) devant la bascule pour qu'elle prenne en entrée soit le sous-ensemble combinatoire précédent (mode normal), soit la sortie de la bascule précédente dans la chaîne de scan (mode test).

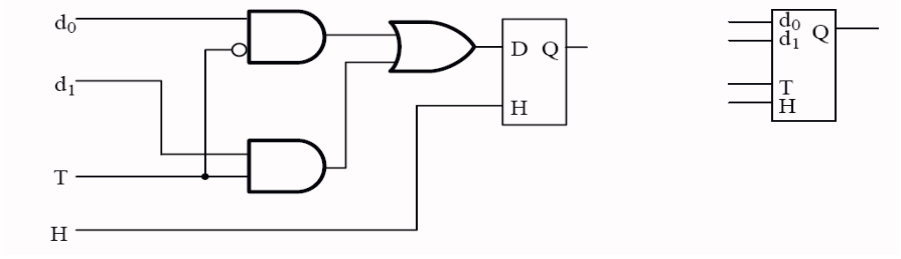


Figure 1.7 : Bascule D modifiée pour le test

Un exemple d'architecture scan est donné Figure 1.8. Les bascules sont modifiées pour être compatibles avec cette architecture. Elles sont également chaînées les unes aux autres afin de former une chaîne de scan. Les entrées I_1 à I_5 sont les entrées primaires fonctionnelles du circuit et les sorties O_1 et O_2 sont les sorties primaires. La chaîne de scan constituée des 3 bascules est marquée en gras, avec l'entrée et la sortie de la chaîne appelées respectivement "Scan_in" et "Scan_out". Le signal de contrôle T, en pointillé sur la Figure 1.8, permet de sélectionner le mode normal, soit le mode test. En mode normal, le signal T vaut '0', les entrées d_0 des bascules sont utilisées. En mode de test, le signal T vaut cette fois '1' et ce sont les entrées d_1 des bascules qui sont sélectionnées, constituant ainsi le registre à décalage de la chaîne de scan).

Lors du test, les données de test arrivent sur l'entrée "Scan_in" et se propagent dans les bascules. Une fois un vecteur stocké dans les bascules du circuit, le circuit est passé en mode normal. Au coup d'horloge suivant, les réponses sont stockées dans les bascules. Le circuit repasse alors en mode test. Le nouveau vecteur est alors chargé dans les bascules tandis que la réponse correspondant au premier vecteur est récupérée à la sortie "Scan_out". L'opération est répétée pour chaque vecteur.

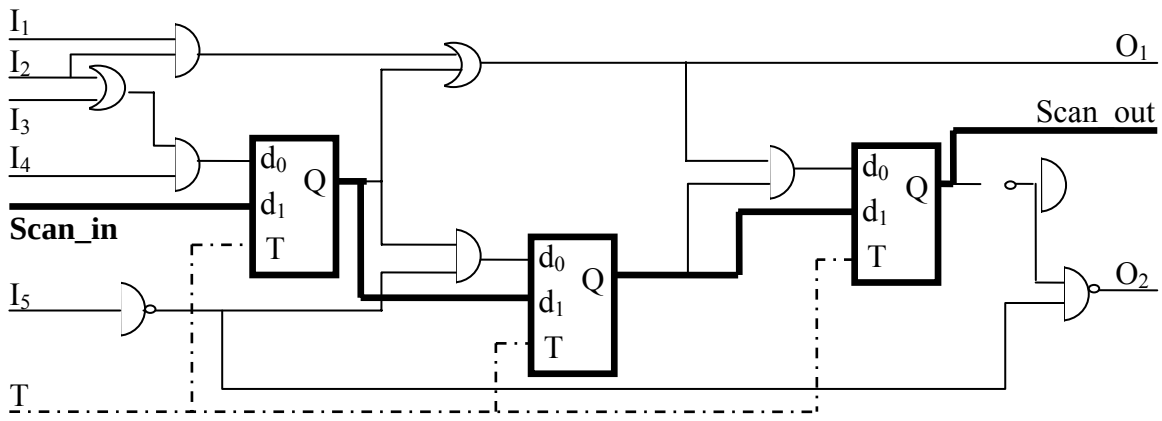


Figure 1.8 : exemple de circuit avec chaîne de scan

1.2.2.1.1. Chaîne de scan unique

Dans le cas d'une chaîne unique (Figure 1.9), toutes les bascules du circuit font partie de la même chaîne, et une seule entrée de test (une seule sortie respectivement) est nécessaire pour acheminer les données de test (le vecteur). La longueur d'un vecteur est égale au nombre de bascules de la chaîne de scan. Il faudra donc autant de cycles que de bascules pour charger le vecteur en série (mode test) dans le registre à décalage (chaîne de scan). L'opération est répétée pour chaque vecteur. Une fois tous les vecteurs appliqués, il faudra également autant de cycles que de bascules pour capturer la dernière réponse.

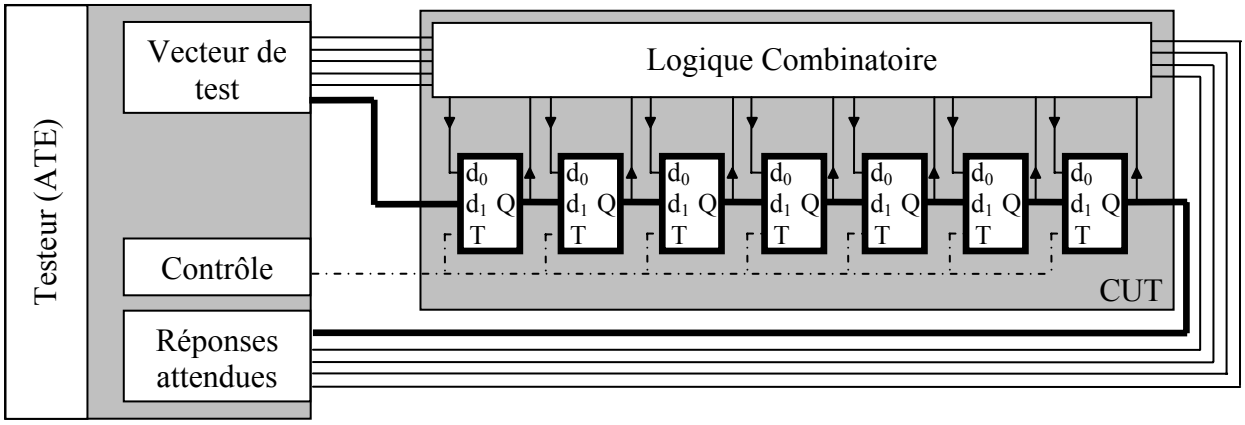


Figure 1.9 : Circuit sous test avec chaîne de scan unique

L'équation suivante donne le temps de test en fonction du nombre V de vecteurs de la séquence et de la longueur L exprimée en nombre de bascules de la chaîne de scan, c'est à dire du nombre de bascules du circuit (dans l'exemple de la Figure 1.9 L=7).

$$\text{Temps de test} = V \times L + V + L$$

Sur l'exemple de la Figure 1.10, il y a 3 vecteurs de 7 bits à charger. Il faut donc 7 cycles par vecteur (d'où 3x7=21 cycles au total), 1 cycle par vecteur pour l'appliquer au circuit et en récupérer la réponse, et 7 cycles pour récupérer la réponse du dernier vecteur. Ce qui fait au total 31 cycles d'horloge pour cet exemple.

Vecteur 3:	1	0	1	0	1	0	0
Vecteur 2:	1	1	1	0	0	0	1
Vecteur 1:	0	0	1	1	1	0	1

Figure 1.10 : Exemple de données de test:
3 vecteurs pour une chaîne de scan unique de 7 bascules

Cette structure convient donc bien à des circuits comportant peu de bascules. Mais dans le cas où le circuit contient plusieurs milliers de bascules – ce qui est relativement courant dans les circuits industriels – le chargement d'un vecteur nécessite un grand nombre de cycles. Le temps de test devient alors prohibitif pour des séquences de test contenant plusieurs milliers de vecteurs.

1.2.2.1.2. Chaînes de scan multiples

L'utilisation de plusieurs chaînes de scan permet de réduire le temps de transfert d'un vecteur. Au lieu d'être regroupées en une seule chaîne de scan, les bascules sont réparties dans plusieurs chaînes, chaque chaîne nécessitant un canal testeur (Figure 1.11). Pour un même nombre de bascules, la longueur des chaînes est évidement plus courte que celle d'une chaîne unique.

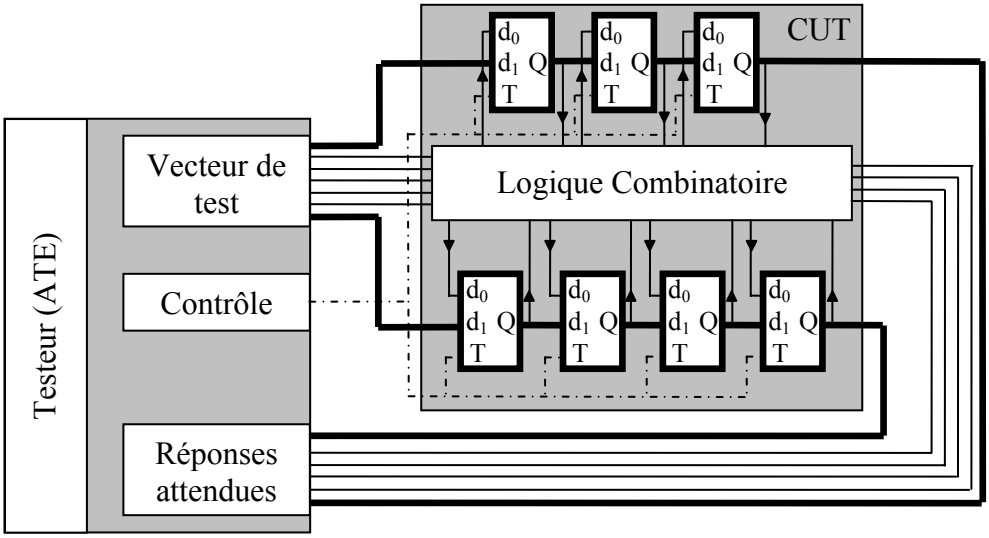


Figure 1.11 : Circuit sous test avec chaînes de scan multiples

Pour charger un vecteur, sur l'exemple de la Figure 1.11, il ne faudra que 4 cycles au lieu de 7 pour le cas d'une chaîne de scan unique. Le temps de test dépend toujours du nombre V de vecteurs dans la séquence et, avec cette structure, du nombre L' de bascules dans la plus longue chaîne. Le temps de test du circuit est donné par la nouvelle équation :

$$\text{Temps de test} = V \times L' + V + L' \quad \text{avec} \quad L' < L$$

Or $$V \times L >> V + L \quad \text{et} \quad V \times L' >> V + L'$$

Le rapport de réduction du temps de test est donc approximativement donné par l'équation suivante:

$$\text{Réduction} \approx \frac{L}{L'}$$

Les données de test sont décrites dans la Figure 1.12. Dans le cas d'un circuit avec N chaînes de scan, les vecteurs sont découpés en L' "slices". Chaque "slice" représente les valeurs d'un vecteur de test à charger dans les bascules de même rang dans les chaînes de scan. Lorsque les chaînes de scan n'ont pas la même longueur, des valeurs indéterminées (X) sont insérées dans les slices aux positions correspondant aux chaînes plus courtes (voir les slices 1 de chaque vecteur sur la Figure 1.12). Ces valeurs sont "expulsées des chaînes les plus courtes par le chargement des slices suivantes.

Dans l'exemple précédent, chaque vecteur est constitué de 4 slices de 2 bits chacun. Le temps de test dans ce cas est de 19 cycles d'horloge. On obtient un rapport de réduction réel de 31 / 19 = 1,63 conforme au rapport théorique 7 / 4 = 1.75.

		Chaîne 1	Chaîne 2
Vecteur 3:	Slice 4:	0	1
	Slice 3:	1	0
	Slice 2:	0	1
	Slice 1:	0	X
Vecteur 2:	Slice 4:	0	1
	Slice 3:	0	1
	Slice 2:	0	1
	Slice 1:	1	X
Vecteur 1:	Slice 4:	1	0
	Slice 3:	1	0
	Slice 2:	0	1
	Slice 1:	1	X

Figure 1.12 : Exemple de données de test:
3 vecteurs pour 2 chaînes de scan de 4 et 3 bascules

Avec ce type de structure de test, le temps de test est effectivement réduit mais au prix d'une augmentation du nombre de canaux testeur. Pour de gros circuits industriels présentant plusieurs milliers de bascules à répartir dans les chaînes de scan, le temps de test est effectivement réduit, mais cette solution s'avère coûteuse en nombre de canaux testeur requis pour la transmission des données de test.

1.2.2.1.3. Compression de données

On a donc vu que le test prenait une part de plus en plus importante dans le prix de revient d'un circuit intégré. En effet, avec l'augmentation incessante de la densité d'intégration des transistors sur une même surface de silicium, le test de ces circuits intégrés devient toujours plus complexe et plus long. De plus les testeurs sont des appareils extrêmement coûteux, qu'il est difficile d'améliorer à un rythme suivant la même évolution technologique que les circuits. Le problème majeur est de faire passer un volume de données toujours plus grand du testeur au circuit sous test tout en essayant de maintenir le temps de test le plus bas possible. Chaque fraction de seconde gagnée sur l'occupation d'un circuit par le testeur est une source d'économie pour le fabricant.

On a vu que l'utilisation de chaînes de scan multiple est une piste pour la réduction du temps de test d'un circuit, mais son application est limitée par le nombre de canaux testeur disponibles.

Le même problème se pose toujours: comment transférer toutes ces données de test au circuit en un temps minimum avec les contraintes liées aux testeurs actuels. C'est dans ce contexte que s'inscrit la première partie de cette thèse.

Une première approche consiste à retirer tout simplement le testeur. Cette technique de test intégré connue sous le nom générique de **BIST** [Bard87] (pour Built-In Self Test en anglais), dont diverses implantations sont décrites dans le chapitre suivant, permet la génération des données de

test, la vérification des réponses et le contrôle de la procédure par des éléments intégrés au sein du circuit (Figure 1.13). L'implantation de ces différents blocs au sein de chaque circuit entraîne une augmentation de la surface sur le silicium. Cette méthode est très efficace lors du test des blocs de logique régulière, comme les mémoires par exemple, mais malheureusement reste souvent difficile pour des blocs de logique aléatoire qui nécessitent des séquences de test particulières et difficiles à intégrer à coût réduit. Ces méthodes sont présentées plus en détail dans la première partie de l'état de l'art dans le chapitre suivant.

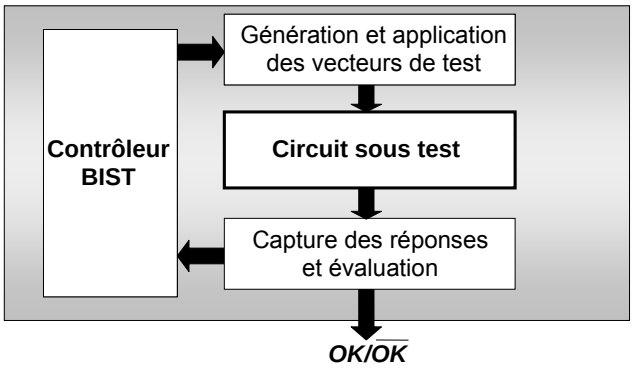


Figure 1.13 : Architecture de test BIST

Une autre façon de procéder est d'insérer dans la chaîne de transmission de données de test des structures de compression-décompression (Figure 1.14) afin de réduire soit le volume de donnés à transmettre (que l'on référencera dans le chapitre suivant sous la dénomination **compression verticale**), soit la bande passante nécessaire à la transmission de ces données (que l'on référencera dans le chapitre suivant sous la dénomination **compression horizontale**). Il s'agit de compresser hors ligne les données de test, de stocker ces données compressées dans le testeur à la place des données d'origine, et de les transférer au circuit puis de les décompresser sur le circuit lui-même grâce à un décompresseur, intégré sur la puce du circuit sous test, qui restituera les données originales au circuit sous test. L'opération inverse étant réalisée pour la transmission des réponses du circuit. Dans ce cas, on parle de compaction de réponses. La compaction des réponses a déjà été l'objet de nombreux travaux et ne sera pas développée d'avantage dans le cadre de cette thèse.

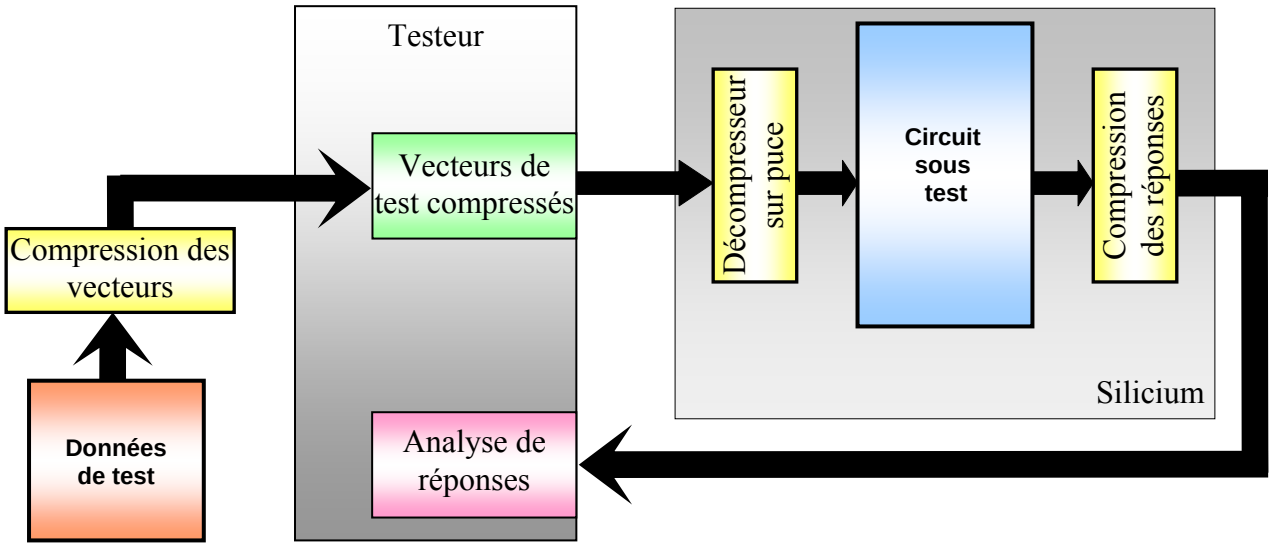


Figure 1.14 : Architecture de Test avec compression

Dans le chapitre suivant, une première partie présente un état de l'art sur les méthodes de compression de données de test. De nombreuses méthodes de BIST, de compression verticale et horizontale y seront détaillées. Nous soulignerons également pourquoi ces méthodes, bien que performantes et satisfaisantes pour le test de circuits intégrés seuls, ne sont pas ou sont mal adaptées à l'utilisation de cœurs IP ("Intellectual Property" en anglais) et, par extension, au test des systèmes sur puces (SoC pour System-on-Chip) qui sont très fréquemment composés d'IP.

Dans un second temps, une nouvelle méthode de compression horizontale des données de test, spécifiquement conçue pour être utilisable dans le domaine d'application des systèmes sur puce sera présentée. Il s'agit d'une approche complètement générale, ne nécessitant pour seule donnée que les séquences de test des circuits et pour laquelle aucune connaissance interne des circuits sous test n'est nécessaire. De plus, aucun autre outil spécifique additionnel n'est requis pour utiliser cette méthode.

Enfin une analyse formelle des performances ainsi que des résultats expérimentaux seront donnés dans une troisième partie. Une comparaison sera effectuée avec une structure standard réalisant la même fonction mais ne comportant pas de compression.

1.3. Références bibliographiques

[Bush00] M.L. Bushnell, V.D. Agrawal: Essentials of electronic testing for digital memory & mixed-signal VLSI circuits. Kluwer Academic Publishers, 2000.

[Howe81] M.J. Howes, V.D. Morgan: Reliability and degradation – Semiconductor devices and circuits. Wiley-Interscience, 1981.

[Abra 81] M. Abramovici, P.R. Menon, D.T. Miller: Critical path tracing – an alternative to fault simulation", Proceedings of the 20th conference on Design automation, p. 214-220, 1983.

[IEEE90] IEEE Standard Board. IEEE std 1149.1-1990: Standard test access port and boundary scan architecture.1990.

[Toub06] N.A. Touba: Survey of Test Vector Compression Techniques. Design & Test Vol. 23, N° 4, 2006, pp. 294-303.

[Bard87] P.H. Bardell, W.H. McAnney, J. Savir: Built-in test for VLSI: pseudorandom techniques. John Wiley and sons, 1987.

Chapitre II:

Compression des données de test

2.1. Etat de l'art 35

2.1.1. BIST 36

2.1.1.1. BIST STUMPS 37

2.1.1.2. Séquences déterministes modifiées 38

2.1.1.3. Séquences aléatoires modifiées 39

2.1.1.4. Procédés non basés sur des LFSR 42

2.1.1.5. Conclusion 45

2.1.2. Méthodes de compression verticale 45

2.1.2.1. Codages Statistiques 45

2.1.2.2. Codages RLE 47

2.1.2.3. Codage FDR 48

2.1.2.4. Codes de Golomb 49

2.1.2.5. Conclusion 50

2.1.3. Méthodes de compression horizontale 50

2.1.3.1. Méthode non compressive: Sérialisation des données 51

2.1.3.2. Diffusion (Broadcasting) 52

2.1.3.2.1. Reconfiguration Statique 52

2.1.3.2.2. Reconfiguration dynamique 54
2.1.3.2.3. Méthodes " Illinois scan " 56
2.1.3.2.4. Méthode industrielles (Synopsys, Syntest) 57
2.1.3.3. Décompresseur linéaire 60
2.1.3.3.1. Décompresseur linéaire à structure combinatoire 61
2.1.3.3.2. Décompresseur linéaire à structure séquentielle statique 62
2.1.3.3.3. Décompresseur linéaire à structure séquentielle dynamique 63
2.1.3.3.4. Méthodes industrielles (Mentor) 64
2.1.3.4. Méthode par dictionnaire 65
2.1.4. Adéquation Compression des données de test et Systèmes sur puces 67
2.1.4.1. Problèmes liés au test des IP 68
2.1.4.2. Problèmes liés au test des SoCs 69
2.2. Nouvelle méthode de compression des données de test. 70
2.2.1. Objectifs: 70
2.2.2. Architecture proposée 71
2.2.2.1. Idée Initiale 71
2.2.2.2. Amélioration 72
2.2.2.3. Décompresseur sur puce 73
2.2.2.3.1. Synchronisation et commande 74
2.2.2.3.2. Fonctionnement 76
2.2.3. Algorithme de compression 82
2.2.3.1. Assignation initiale 83
2.2.3.2. Assignations intermédiaires 83
2.2.3.3. Attribution des canaux testeur 85
2.2.3.4. Compression 85
2.2.4. Performance 86

2.2.4.1. Equations temps de test 86

2.2.4.2. Equation volume de données 86

2.3. Résultats et comparaisons 87

2.3.1. Comparaison avec une méthode standard de sérialisation des données 87

2.3.1.1. Principe et description 87

2.3.1.2. Performance 88

2.3.2. Résultats expérimentaux 88

2.3.2.1. Procédure expérimentale 89

2.3.2.2. Analyse des résultats pour un ou 2 circuits complets 89

2.3.2.3. Généralisations aux autres circuits 92

2.4. Conclusion 93

2.5. Références bibliographiques 95

Dans ce chapitre nous proposons une technique de compression des vecteurs de test destinée aussi bien aux circuits seuls qu'aux cœurs de systèmes sur puces. Cette technique permet de réduire le nombre de broches de test nécessaires (canaux testeur utiles), sans modifier la couverture de fautes de la séquence de test définie au cours de la conception du circuit. Cette méthode a pour paramètres le nombre d'entrées du circuit et le nombre de canaux testeur disponibles. Le décompresseur est indépendant de la structure du cœur à tester, ne demande pas de modifications internes du cœur et peut être partagé par plusieurs cœurs. Elle est donc parfaitement adaptée aux concepteurs utilisant des **IP** (ou "**Intellectual Property**"). Les IP sont des circuits achetés tels quels par les assembleurs de système. On trouve plusieurs types d'IP dans le commerce comme les soft IP (source VHDL ou Verilog qui doit être synthétisée et optimisée au niveau physique) ou hard IP (description au niveau physique du circuit pour une technologie silicium donnée). Cependant, les informations structurelles de ces IP, comme par exemple la liste des portes du circuit (ou "netlist"), ne sont pas nécessairement fournies par les vendeurs. Il est donc impossible pour l'acquéreur d'utiliser un générateur de vecteur (ATPG) puisque celui-ci ont besoin de la description précise du circuit. La plupart du temps, les concepteurs fournissent les séquences de test et les réponses correspondantes à cette séquence uniquement. A l'assembleur de mettre en place le test de ce circuit IP.

Nous présenterons dans un premier temps un panel des diverses méthodes de compression existantes en insistant plus particulièrement sur la compression horizontale toujours dans l'optique de l'utilisation d'IP. Nous verrons ensuite pourquoi ces méthodes sont mal adaptées au test des IP dont seules les séquences de test sont connues.

Nous proposons donc, dans une deuxième partie, une méthode simple et efficace, aisément intégrable, ne nécessitant aucune autre connaissance particulière du circuit à tester, et surtout utilisant un décompresseur séquentiel pouvant être partagé par plusieurs circuits différents. Cette méthode permet de réduire de manière drastique le nombre de canaux testeur requis pour transmettre les vecteurs de test aux circuits et donc de réduire le coût du test d'un circuit.

Ce chapitre se termine par une comparaison de notre méthode avec une méthode standard de sérialisation des données de test. Nous verrons que notre méthode, dans le cadre de la compression horizontale permettant de réduire le nombre de canaux testeur requis pour le test d'un circuit, permet d'atteindre des réductions du temps de test avoisinant les 50% par rapport à la sérialisation des données.

2.1. Etat de l'art

Dans cette partie, nous exposons les principales techniques proposées dans la littérature pour transmettre les données de test au circuit sous test. La majorité des circuits commercialisés ayant une structure séquentielle, et la méthode de conception en vue du test (DFT: Design For Test) la plus largement utilisée étant le scan, notre analyse mettra donc principalement l'accent sur les techniques adaptées au test des circuits séquentiels dotés de chaînes de scan.

Comme proposé au chapitre I, la source des vecteurs de test est le plus souvent un testeur externe relativement coûteux. Il est alors intéressant d'optimiser son utilisation en minimisant le volume des données de test à transmettre ou en réduisant la bande passante nécessaire pour transmettre ces données. Nous avons vu que les problèmes majeurs à résoudre sont d'une part le nombre limité de broches de test et d'autre part la profondeur mémoire limitée dans les testeurs industriels et enfin le temps de test. Pour cela trois voies sont possibles :

- implanter une structure de test interne au circuit ou Buit-In Self Test en anglais (**BIST**). Le nombre de broches d'entrées ne pose plus aucun problème puisque les vecteurs de test sont générés de façon interne. Dans ce cas, seule une broche de test en entrée suffit pour lancer le processus de test.
- réduire le nombre de vecteurs de test à transmettre. Cela revient à réduire la profondeur mémoire utile du testeur. Pour réduire le nombre de vecteurs, il est possible d'utiliser des méthodes de compression que nous appellerons **compression verticale**.
- réduire la taille des vecteurs de test à transmettre. Il s'agit cette fois de réduire le nombre de canaux testeur nécessaires. Pour réduire ce nombre, il est possible de réduire la taille des vecteurs (**compression horizontale**) de deux façons : soit en sérialisant les données de test, mais au prix d'une augmentation drastique de la profondeur mémoire du testeur et donc du temps de test, soit en compressant les données de façon à peu augmenter la profondeur mémoire et en conséquence peu augmenter le temps de test.

2.1.1. BIST

Les techniques de BIST ou Built-In Self Test consistent à ajouter au circuit sous test une structure de test indépendante, intégrée sur la puce. Elle se présente sous la forme décrite dans la Figure 2.1 et comporte les éléments suivants:

- un générateur de vecteurs de test qui est le plus souvent un générateur pseudo aléatoire (par exemple un LFSR ou "Linear Feedback Shift Register"),
- un analyseur de réponses (compacteur de réponses et comparateur),
- une logique de contrôle,
- et enfin, une logique additionnelle et en particulier de multiplexage.

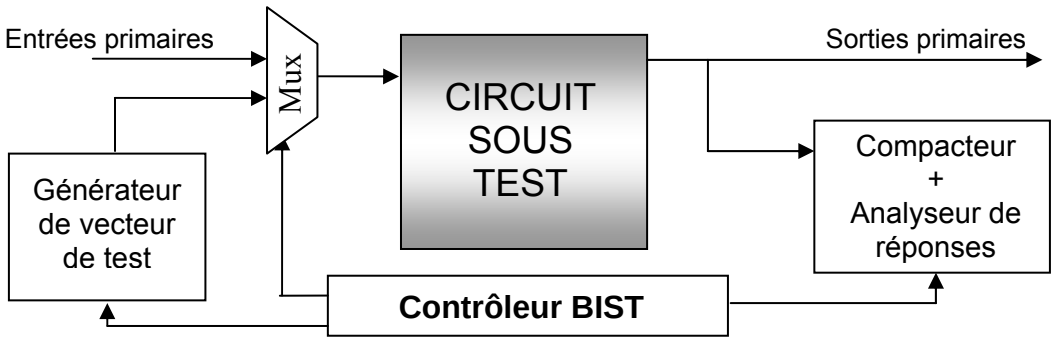


Figure 2.1 : Architecture BIST générale

Initialement, la principale motivation en faveur de l'utilisation du BIST est de limiter, voire d'éliminer l'utilisation d'un testeur externe afin d'en limiter son coût. Classiquement, les vecteurs pseudo-aléatoires sont générés au sein même du circuit et les signaux de contrôles sont assurés par un contrôleur intégré au circuit. La quantité de données échangées entre le testeur et le circuit sous test est réduite de façon drastique puisqu'elle se limite à un signal de démarrage du test et un signal indiquant si le test est correct ou pas. La faible mémoire et les faibles performances requises permettent l'usage de testeurs industriels très faible coût.

Il existe plusieurs façons d'implanter ce type de structure. Les principales techniques utilisant le BIST sont présentées ci-après.

2.1.1.1. BIST STUMPS

Une architecture générique de BIST basée sur une horloge unique connue sous le nom d'architecture STUMPS [Bard87] est illustrée sur la Figure 2.2. Il s'agit d'une structure BIST destinée au test de circuits comportant plusieurs chaînes de scan. La figure représente le circuit sous test et le contrôleur logique BIST. Les différents composants sont:

- le générateur de vecteurs (PRPG pour Pseudo-Random Pattern Generator) généralement implanté à l'aide d'un LFSR, accompagné d'un circuit de décorrélation des bits (phase shifter) qui permet de casser les dépendances linéaires des données des chaînes de scan
- un bloc d'analyse des réponses de test, en sortie des chaînes de scan, composé d'un MISR (Multi Input Signature Register), d'un compacteur et de portes logiques
- le contrôleur qui se compose principalement de deux compteurs : un compteur de vecteurs et un compteur de décalage pour déplacer les données dans les chaînes de scan
- le bloc de décodage (décodeur) qui est utilisé pour piloter les points de test.

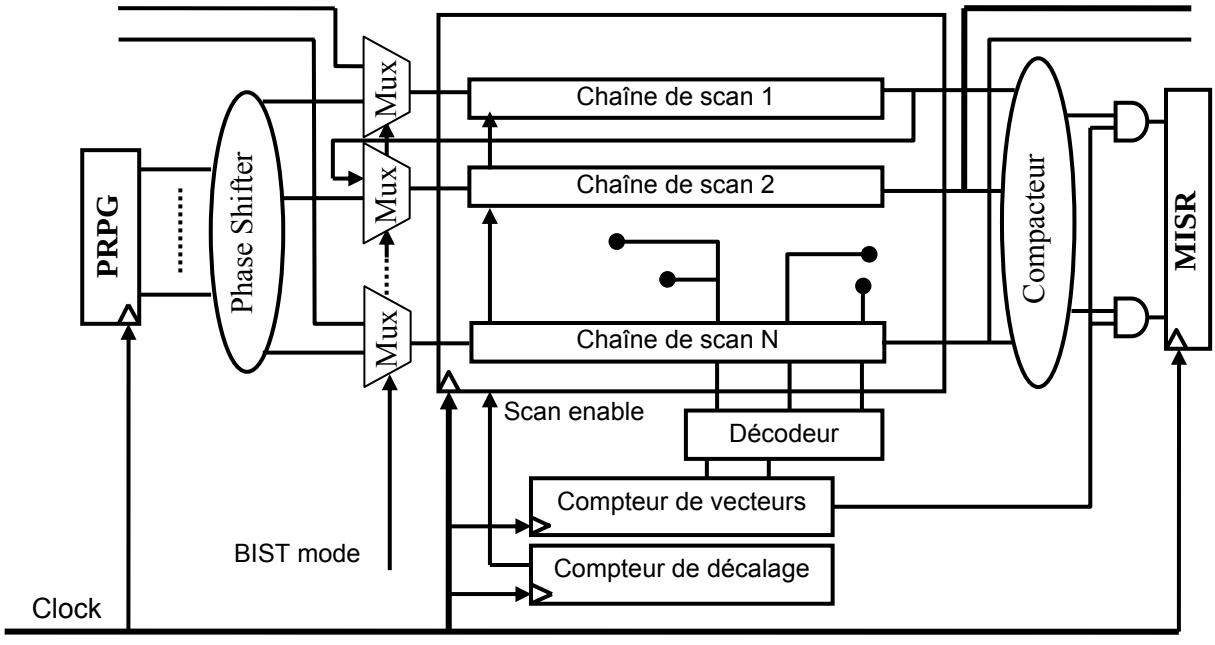


Figure 2.2 : Architecture BIST STUMPS

Cette structure standard permet une génération des vecteurs pseudo-aléatoires directement sur la puce. Le testeur est uniquement utilisé pour donner le départ de la procédure de test et pour récupérer la signature du test indiquant si le circuit est sain ou non. Le test étant indépendant du testeur, il peut être effectué à la fréquence de fonctionnement nominale du circuit, souvent bien plus élevée que les fréquences des testeurs industriels. Le test requiert donc un temps d'occupation du testeur plus faible. Cependant, le caractère aléatoire des vecteurs ne permet pas de garantir un taux de couverture donné pour un nombre de vecteur de test fixé.

2.1.1.2. Séquences déterministes modifiées

Les techniques faisant appel à des procédés BIST et à un test externe sont appelées procédés hybrides. Ils ont pour but d'améliorer la détection des fautes avec peu de surface additionnelle supplémentaire et surtout, requièrent une quantité de données à stocker dans la mémoire du testeur inférieure à celle impliquée lors d'un test entièrement externe. Dans les deux méthodes ci-dessous, des données de test internes et externes sont combinées pour donner la séquence de test finale.

Une première approche de BIST hybride consiste à générer de manière interne des vecteurs pseudo-aléatoires pour détecter les fautes facilement testables. Ensuite, les fautes restantes (dites fautes résistantes [Bard82]) sont détectées à l'aide de vecteurs déterministes complets chargés par le testeur. Par exemple dans la méthode dans [Pres99], l'auteur propose de contrôler le ratio entre le nombre de vecteurs pseudo-aléatoires et le nombre de vecteurs déterministes de façon à limiter le temps de test sans pénaliser le taux de couverture de fautes. On appelle ces vecteurs déterministes des "vecteurs de top-up". La séquence de test totale alors obtenue peut être qualifiée de "séquence hybride". Il faut noter qu'une séquence de top-up peut représenter de 30 à 70% d'une séquence déterministe et donc ne présente qu'un avantage très limité par rapport à un test totalement externe.

Dans [Das00], les auteurs partent de la constatation que dans un vecteur de test déterministe certains bits de ces vecteurs ne sont pas utiles pour la détection des fautes difficiles. Ils peuvent alors être remplacés par des données internes générées par du BIST. Pour cela, une technique consiste à fusionner les données aléatoires du BIST avec des données déterministes issues d'un ATPG afin de générer des "vecteurs hybrides" par opposition aux "séquences hybrides" décrites dans [Pres99].

Dans cette approche de BIST hybride, un système de multiplexage (Figure 2.3) a pour fonction de charger soit des données générées de façon interne par le générateur de vecteurs (PRPG), soit des données de test externes chargées par les entrées de test (SDI). La différence principale avec la méthode précédente réside dans la manière dont les données externes et aléatoires sont utilisées. Ici, les données externes sont appliquées non pas en tant que vecteurs de test complets, mais comme complément dans les chaînes de scan : au lieu de charger entièrement les chaînes de scan avec des valeurs issues du générateur aléatoire, les bascules de même rang des chaînes (une slice d'un vecteur de test) peuvent recevoir des valeurs depuis les entrées de test externes.

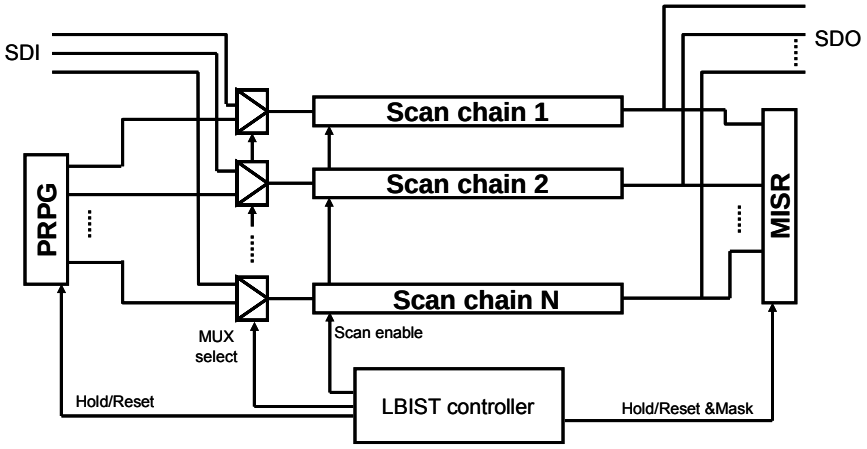


Figure 2.3 : Architecture BIST hybride

Cette méthode réduit le volume des données de test envoyées par le testeur d'un ordre de grandeur de 50% sur les circuits de référence ISCAS'89 alors que l'utilisation de vecteurs de top-up utilisée dans [Pres99] apporte une réduction de l'ordre de 30%.

Cependant cette méthode impose une synchronisation des données internes (BIST) et externes (vecteurs déterministes) pour que les bits des vecteurs déterministes soient chargés dans les bascules des chaînes de scan qui leurs sont attribuées. De plus, le contrôleur BIST est plus complexe que dans la méthode précédente puisque les multiplexeurs doivent être commandés séparément.

Ces méthodes sont plus performantes que l'architecture STUMPS en termes de détection de fautes, mais elles nécessitent une plus grande surface additionnelle, une plus grande synchronisation avec le testeur, et plus de mémoire et de bande passante pour le testeur. De plus la connaissance du circuit est indispensable pour pouvoir utiliser un ATPG pour la génération des vecteurs déterministes.

2.1.1.3. Séquences aléatoires modifiées

Les procédés décrits ci-après correspondent à un type de test hybride qui ne demande pas d'appliquer des données de test additionnelles, mais de produire des données de contrôle qui altèrent les vecteurs générés en interne de manière à détecter plus de fautes. Les deux techniques suivantes sont classées parmi les techniques aléatoires pondérées. Une logique additionnelle modifie certains bits des vecteurs pseudo-aléatoires pour améliorer la détection des fautes difficiles. Les données de contrôle de ce procédé sont soit stockées sur le testeur, soit générées par une structure interne.

Dans la méthode proposée dans [Pome93], l'idée est d'augmenter la couverture de fautes à l'aide d'un réseau de portes logiques modifiant la séquence générée par le LFSR.

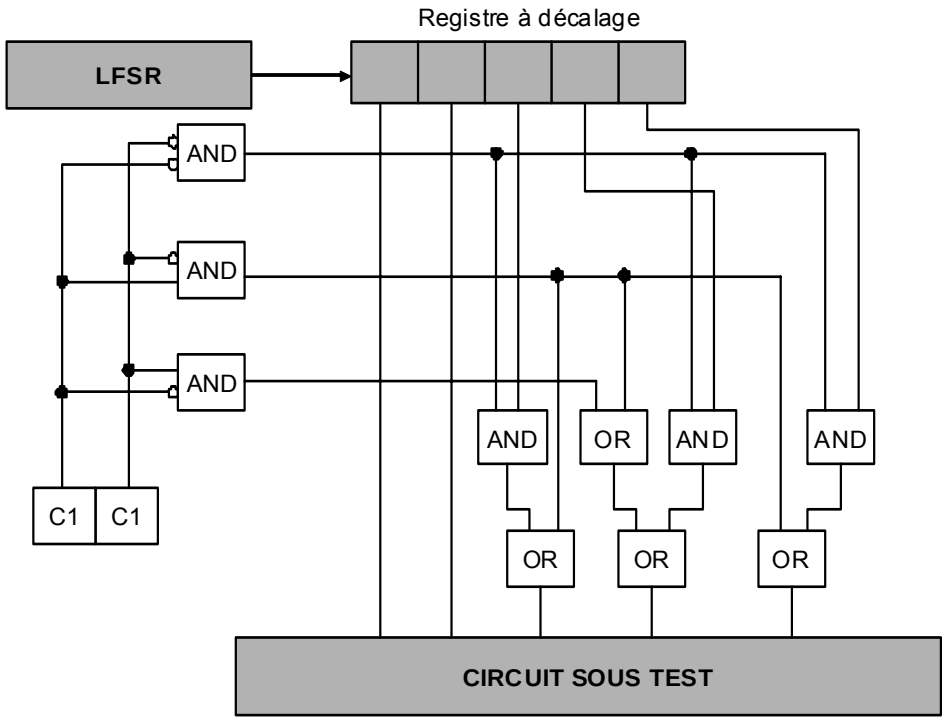


Figure 2.4 : Test aléatoire pondéré interne

Dans l'exemple de la Figure 2.4, un LFSR composé de 5 cellules est utilisé pour générer des vecteurs pseudo-aléatoires. Les deux entrées C1 et C2 sont commandées par un compteur par deux implanté dans le circuit. Les vecteurs pseudo-aléatoires générés par le LFSR sont modifiés par le circuit logique à l'entrée des chaînes de scan en fonction des poids générés par le compteur. A chaque coup d'horloge, le compteur par deux fournit un index (C1, C2) utilisé pour changer de poids. Ces valeurs sont calculées au préalable par simulation de fautes. L'efficacité du test provient du fusionnement des données aléatoires dans le réseau logique et de l'utilisation de l'index. Cette méthode permet de ne pas utiliser de données externes. Toutefois, malgré les améliorations apportées par cette technique, la séquence de test reste très longue pour obtenir une couverture de fautes suffisante et la connaissance architecturale du circuit est, là encore, nécessaire pour la simulation de fautes.

Une idée similaire est proposée dans [Jas01]. Il s'agit de biaiser la génération pseudo aléatoire de manière à ce que les fautes difficiles puissent être détectées. Une LUT ("Look-up Table" ou table de correspondance) a été ajoutée à l'architecture du BIST. Elle permet d'affecter des poids à l'entrée des portes logiques qui se trouvent entre le LFSR et les chaînes de scan (Figure 2.5). Le contenu de la LUT est préalablement déterminé à l'aide d'un ATPG et de simulation. A chaque coup d'horloge, le testeur fournit un index sur k bits pour récupérer les poids contenus dans la LUT et les appliquer.

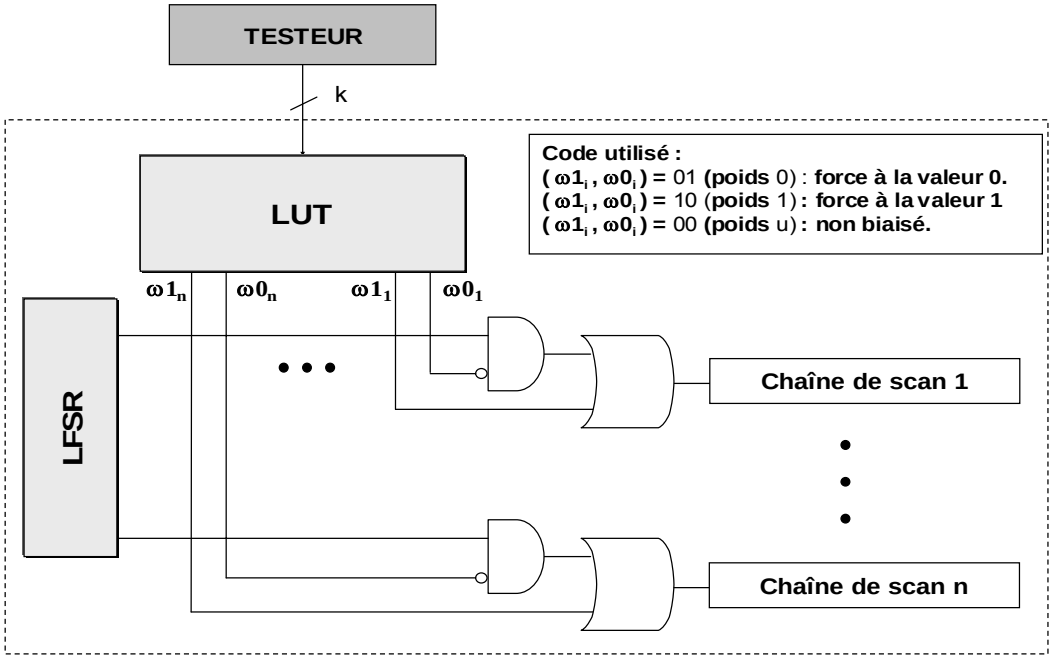


Figure 2.5 : Test Aléatoire pondéré par le testeur

Cet index de k bits correspond à l'adresse de la ligne de la LUT. Comme k est très inférieur à n (nombre de bits sur lesquels sont codés les poids), le nombre de canaux testeur à utiliser est ainsi minimisé. Par exemple, si 7 poids de 64 bits doivent être stockés dans la LUT, un index de 3 bits seulement sera suffisant. Cette méthode permet une réduction significative du volume de test (de l'ordre de 99%) par rapport à une séquence BIST classique complétée avec des vecteurs de top-up. Il faut quand même noter que le test nécessite là aussi une synchronisation testeur-circuit complexe et une fréquence d'application des vecteurs au circuit plus lente. Une fois de plus, le circuit doit être connu pour utiliser l'ATPG et la simulation de fautes.

Un troisième procédé présenté dans [Wund96] [Hell00] s'inspire d'une technique appelée "bit-flipping". L'idée est de faire correspondre la séquence générée par le LFSR à une séquence générée au préalable par ATPG. L'utilisation du bit-flipping permet de modifier la valeur de certains bits dans les vecteurs pseudo-aléatoires du LFSR (Figure 2.6-a) afin de recouvrer le taux de couverture initial ou au moins de s'en approcher. Cette opération est réalisée par un réseau combinatoire de portes logiques et par un bloc réalisant la fonction de bit-flipping.

Une autre méthode présentée dans [Kay01] n'utilise qu'une porte ou-exclusif contrôlée par l'entrée "Testeur" (Figure 2.6-b) pilotée par un testeur externe pour inverser les valeurs fournies par le LFSR. Ces données de contrôle sont stockées dans la mémoire du testeur externe. Le test est donc, dans ce cas, cadencé par le testeur.

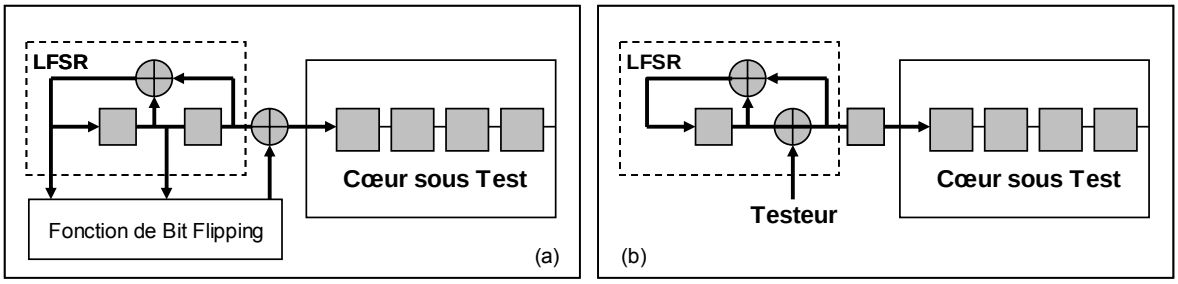


Figure 2.6 : Test Aléatoire modifié par "Bit-Flipping"

Les auteurs ont montré que de nombreux bits contenus dans les séquences LFSR n'ont pas à être modifiés. Une grande partie des bits de contrôle reste donc assignée à '0'. Cela implique que les données à stocker sur le testeur contiennent de nombreuses séquences de '0'. L'idée est d'exploiter cette caractéristique pour utiliser un principe de codage par répétition de façon à compresser les données testeur. L'entrée "Testeur" est alors précédée d'un décodeur. Il s'agit là de combiner une technique de BIST et une technique de compression verticale. Plusieurs de ces techniques seront décrites dans la partie suivante. Elles ont pour inconvénient majeur de nécessiter des schémas de synchronisation complexes entre le testeur et le circuit. Cette méthode permet de réduire le volume de données à stocker dans la mémoire du testeur de l'ordre de 30 à 77% par rapport à une méthode de scan classique.

2.1.1.4. Procédés non basés sur des LFSR

Toutes les méthodes précédentes nécessitent d'intégrer dans le circuit du matériel supplémentaire, spécifique au test, sous la forme de LFSR et de logique de contrôle. Or il existe d'autres méthodes permettant de générer des vecteurs pseudo-aléatoires ne requérant pas de LFSR additionnel. En effet, on peut trouver sur ces circuits des éléments nécessaires au fonctionnement normal aux circuits tels que des unités arithmétiques, des mémoires, des processeurs divers, etc. L'idée de base de ces techniques non basées sur des LFSR est de réutiliser certains de ces éléments déjà présents dans le circuit en tant que générateur de vecteurs. Cette réutilisation permet de diminuer l'implantation de logique supplémentaire pour le test et donc de réduire les coûts relatifs à la surface additionnelle de silicium requise pour le test de ces circuits.

Une première méthode consiste à réutiliser une structure arithmétique. L'utilisation d'une structure additionneur/accumulateur comme générateur aléatoire présente l'avantage que l'additionneur/accumulateur est éventuellement présent dans l'environnement d'un circuit au sein d'un système sur puce multi-cœurs. Les auteurs de [Dors98] proposent une méthode de calcul pour trouver les valeurs d'initialisation (ou graines) **r** du registre/accumulateur et de la constante **c** qui permet de régénérer les vecteurs déterministes désirés (Figure 2.7-a). Une valeur unique pour **r** et **c** est suffisante pour retrouver ces vecteurs dans la séquence produite, mais au prix d'un temps de génération parfois important. En effet, 16 coups d'horloge sont nécessaires dans le cas de la Figure 2.7-b pour générer les 5 vecteurs désirés. Les auteurs expliquent comment calculer un nombre

restreint et suffisant de valeurs pour **r** et **c** de façon à réduire les temps de test. Par exemple les 2 doublets (r,c)=(01110,01111) ; (01001,10000) sont suffisants pour obtenir les 5 vecteurs t1,…,t5 en 7 coups d'horloge (Figure 2.7-c). Une file de registres et une mémoire de type ROM ou RAM sont utilisées pour mémoriser les doublets (r,c).

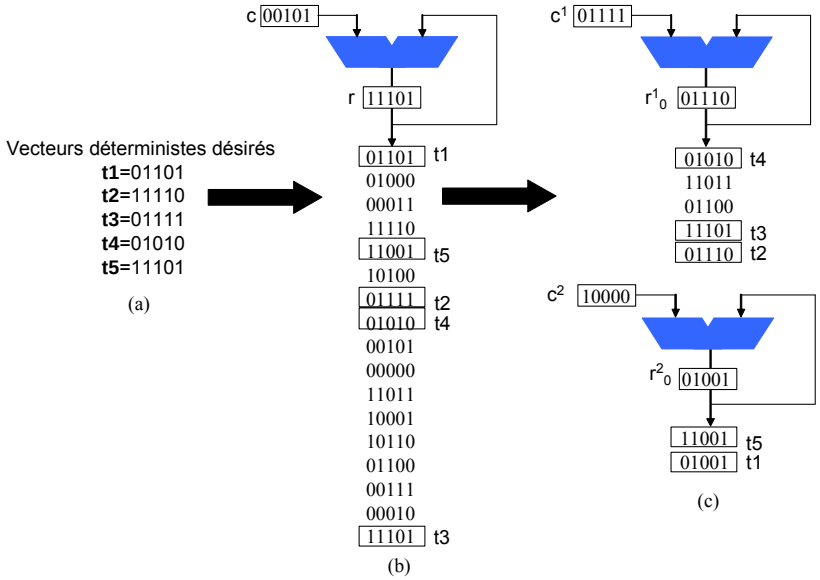


Figure 2.7 : Test intégré par procédé arithmétique

Une autre solution utilisable dans le cas du test de circuits sécurisé est de réutiliser la structure de cryptage en tant que générateur de vecteurs pseudo-aléatoires. Dans [Doul07], une méthode de génération de vecteurs utilisant une structure d'AES est présentée. Les auteurs exploitent le caractère aléatoire de l'AES. Une graine d'initialisation est introduite en tant que texte clair dans la structure cryptographique qui est ensuite rebouclée afin de générer de manière autonome des vecteurs de test pseudo aléatoires.

La Figure 2.8 présente l'architecture de l'AES et les modifications apportées (en grisé) pour l'utiliser en tant que générateur de vecteur. Ce mode de fonctionnement nécessite l'addition de nouveaux signaux de contrôle. Le signal select est à '0' pendant la première étape en mode générateur de vecteur. Il permet de réinitialiser la graine. Une fois le signal à '1', l'AES boucle sur lui-même et les vecteurs de test sont produits dans le registre de sortie à chaque cycle d'horloge.

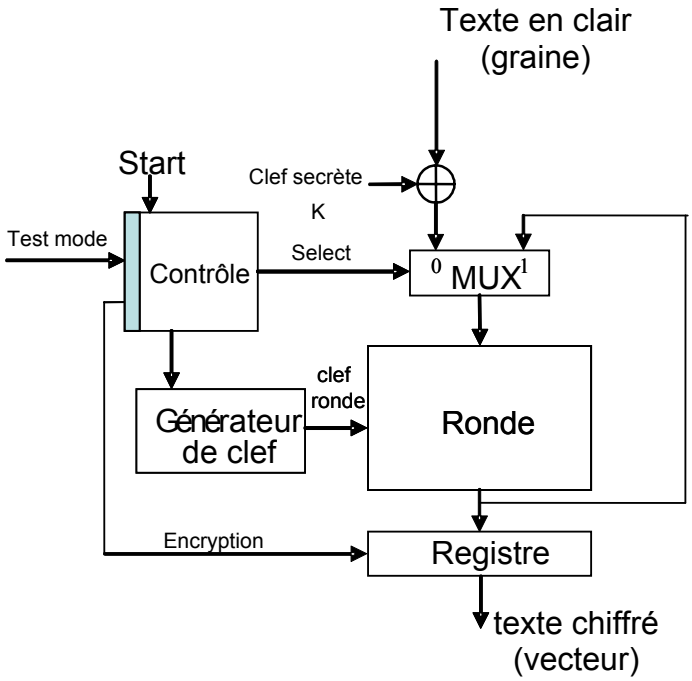


Figure 2.8 : Test intégré par réutilisation d'une structure sécurisée AES

Les auteurs comparent les résultats obtenus à l'aide de l'AES à des résultats LFSR. Les séquences générées ont été appliquées à trois circuits de la gamme des benchmark ISCAS'89. Le Tableau 2.1 reporte le taux de couverture obtenu pour ces 3 circuits implantés avec 1, 16 ou 128 chaînes de scan.

Circuit	Nombre de vecteurs	Nombre de chaînes de scan	FC (%)	
			LFSR	AES
s9234	21483	1	95.77	95.96
		16	95.54	95.96
		128	95.52	95.84
s13207	15000	1	99.37	95.83
		16	94.62	96.02
		128	86.48	95.42
s38584	84898	1	91.26	91.93
		16	90.82	92.25
		128	91	91.53

Tableau 2.1 : Comparaison entre LFRS et AES utilisé en générateur de vecteur

La réutilisation de structure déjà existante apparaît comme une alternative intéressante à l'implantation d'un générateur de vecteurs de test en fournissant de bonnes performances en termes de couverture de fautes. Ces solutions permettent une économie en termes de surface additionnelle de silicium.

2.1.1.5. Conclusion

Les méthodes de BIST permettent de réduire, voire de supprimer l'utilisation d'un testeur externe pour effectuer le test du circuit. Elles sont souvent peu coûteuses en termes de surface de silicium additionnelle et en temps de test. Cependant, ces techniques requièrent la possibilité de modifier la structure interne du circuit ou, tout au moins, de connaître cette structure. En effet ces méthodes nécessitent de générer des vecteurs additionnels et/ou modifier la séquence initiale, tâches pour lesquelles il est nécessaire de connaître la structure du circuit sous test (ATPG, Simulation de fautes…). Pour cette raison, les techniques de BIST ne sont donc pas applicables au cas du test des IPs, dont la structure est, par définition, inconnue et non modifiable.

2.1.2. Méthodes de compression verticale

La compression verticale consiste à réduire le volume des données de test à transférer sur une broche du testeur vers le circuit sous test. Ces méthodes sont essentiellement basées sur un système de codage, méthode qui est fréquemment utilisée dans le domaine de la compression de fichiers en informatique. Les données de test sont encodées hors ligne puis stockées dans la mémoire du testeur. Elles sont ensuite transmises à un décodeur intégré dans le circuit. Ce décodeur recompose le flot de données et les transmet au circuit sous test.

Les données de test peuvent être codées soit avec un codage fixe, c'est-à-dire que chaque code compressé envoyé par le testeur au décodeur est de taille fixe, soit avec un codage variable, c'est-à-dire que les codes compressés envoyés au décodeur sont de taille variable. De même, les données décodées peuvent se présenter sous la forme de mots de taille fixe ou variable. Les tailles fixes ou variables des codes et des mots à coder entraînent la mise en place de schémas de synchronisation particuliers plus ou moins compliqués entre le testeur et le circuit.

2.1.2.1. Codages Statistiques

Les codages statistiques sont du type fixe vers variable. Cela signifie que des codes de longueur variable sont utilisés pour représenter des blocs de données de taille fixe. L'idée est d'utiliser des codes comportant peu de bits pour coder les blocs les plus fréquents dans l'ensemble de test. Le but est de minimiser la longueur totale de la séquence codée. Bien entendu, aucune compression ne peut être effectuée si tous les blocs ont la même fréquence d'occurrence.

Le plus courant des codages statistiques est le codage de Huffman (1952). Cette méthode de compression de données est utilisée en particulier dans le cadre de la transmission de messages par télécopie et sur minitel ou encore du format de l'image compressée TIFF spécifié par Microsoft® et Aldus®.

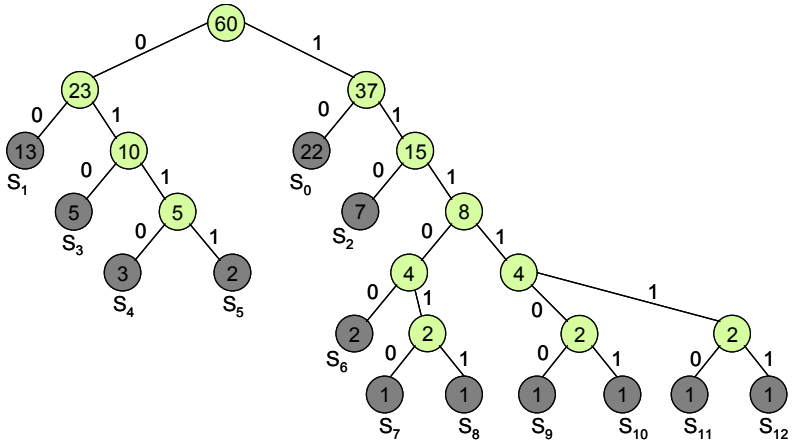


Figure 2.9 : Arbre correspondant au codage de Huffman pour les symboles du Tableau 2.2 (4ème colonne)

Les auteurs de [Jas99] et [Jas03] proposent une méthode basée sur ce codage. Ils utilisent une architecture de décompression simple dans le but d'obtenir le meilleur rapport compression/surface. L'utilisation de cubes de test (séquences non complètement spécifiées) comme donnée à encoder peut permettre d'augmenter les fréquences d'occurrence de certains symboles. En effet, les bits non spécifiés (les X) peuvent être assignés de manière à être codés comme des symboles fréquents afin d'augmenter le taux de compression.

Les caractéristiques du code proposé sont les suivantes :

- Si la fréquence d'un symbole est faible, le code commence par le préfixe '0', le reste du code est le symbole lui-même.
- Si la fréquence d'un symbole est élevée, le code commence par le préfixe '1'suivi du code de Huffman.

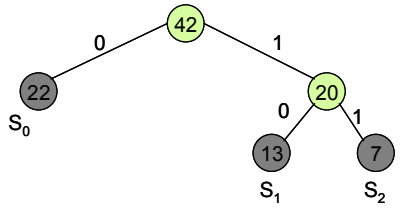


Figure 2.10 : Arbre correspondant au codage de Huffman pour les symboles les plus fréquents (5ème colonne du Tableau 2.2)

Le Tableau 2.2 illustre l'exemple du codage de Huffman et Huffman modifié pour des données composées de symboles (dans notre cas, les vecteurs de test) sur 4 bits. Les colonnes sont remplies en fonction des arbres donnés Figure 2.9 et Figure 2.10. Les fréquences d'occurrence des symboles sont indiquées dans la troisième colonne. On peut voir dans ce tableau la différence entre le codage de Huffman (4ème colonne) est le codage de Huffman modifié (5ème colonne).

Symboles	Valeurs	Fréquence	Huffman classique	Huffman Modifié
S_0	0010	22	01	**1**0
S_1	0100	13	00	**1**10
S_2	0110	7	11	**1**11
S_3	0111	5	010	**0**0111
S_4	0000	3	0110	**0**0000
S_5	1000	2	0111	**0**1000
S_6	0101	2	11100	**0**0101
S_7	1011	1	111010	**0**1011
S_8	1100	1	111011	**0**1100
S_9	0001	1	111100	**0**0001
S_{10}	1101	1	111101	**0**1101
S_{11}	1111	1	111110	**0**1111
S_{12}	0011	1	111111	**0**0011
S_{13}	1010	0	-	-
S_{14}	1110	0	-	-
S_{15}	1001	0	-	-

Tableau 2.2 : Comparaison entre code de Huffman et code de Huffman modifié

Lorsqu'il s'agit de symboles à faible fréquence d'apparition (pour les symboles S_3 à S_{12}), le codage classique augmente la taille du code alors que le codage modifié garde toujours la même taille que le codage d'origine avec le préfixe en plus (5^{eme} colonne). Cette méthode (Huffman modifié) permet ainsi de réduire la taille du décompresseur par rapport à celui utilisé pour un code de Huffman classique. En effet, la taille du décodeur croît de manière exponentielle avec la taille des codes à générer.

2.1.2.2. Codages RLE

Le codage RLE (Run-Length Encoding) est un codage variable vers fixe : des symboles (données de test) de longueur variable sont associés à des mots binaires (codes) de longueur fixe. Le but est de réduire le nombre de bits à mémoriser en remplaçant chaque série de bits successifs identiques par leur nombre. Par exemple au lieu de mémoriser la suite 0000001 comprenant 7 bits, on la remplace par le code 110 (3 bits) qui signifie qu'il faut générer 110= 6 (en base 10) fois le bit 0 avant de générer un 1. Ce codage ne peut être efficace que si un grand nombre d'éléments successifs sont identiques. Pour cela les valeurs non spécifiées des cubes de test (les X) sont remplacées par une valeur unique (le '0' par exemple). Chaque chaîne de '0' de longueur L est codée sur b bits. La longueur b du code s'exprime avec la formule suivante $0 \leq L \leq 2^{b-1}$.

Dans [Jas98], les auteurs proposent une utilisation astucieuse de ce système de codage. Pour charger la chaîne de scan interne du cœur, une chaîne de scan cyclique est utilisée. Cette chaîne de scan cyclique utilise un ou exclusif qui fait la somme entre le dernier bit de la chaîne et la sortie du décodeur comme le montre la Figure 2.11 . La chaîne de scan cyclique contenant un vecteur, pour charger suivant il suffit de présenter la différence arithmétique entre les deux vecteurs. En tirant

partie du fait que deux vecteurs successifs sont proches (au sens arithmétique du terme), cette différence est composée de beaucoup de 0. Du fait de l'utilisation du codage fixe, le nombre de bits à transmettre au décodeur RLE est plus faible que le nombre de bits originaux (uniquement dans le cas où le nombre de 0 est supérieur à la taille du code). C'est cette différence qui est envoyée depuis le testeur au décodeur sur puce. Lorsque le vecteur attendu est présent dans la chaîne de scan cyclique, ce vecteur est alors appliqué à la chaîne de scan interne en parallèle.

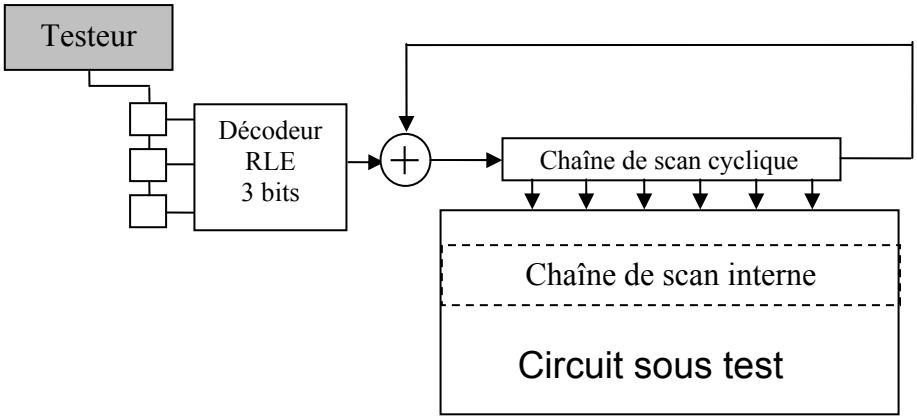


Figure 2.11 : Exemple de décodage 3bits pour un code RLE

2.1.2.3. Codage FDR

Une variante de codage RLE est décrite dans [Chan01] [Chan03]. Selon les auteurs, le codage RLE classique ne permet pas de coder les grandes salves de '0' de façon efficace (la taille du code est presque aussi grande que la taille du message original). Le codage proposé dans cette méthode se nomme codage FDR (Frequency-Directed Run-length). Il utilise le même principe de codage des séquences de '0' successifs, cependant il s'appuie sur les fréquences d'occurrence des séquences de '0'. Le but ici est d'associer les codes les plus courts aux salves de '0' les plus fréquentes. Par exemple, si l'ensemble de test est constitué d'avantage de salves 0001 que de salves 001 on associera un code plus court au mot 0001 qu'au mot 001. Il s'agit là d'un code variable vers variable. Les séquences de 0 sont divisées en groupe A_1, A_2, A_3,…, A_k où k est déterminé par la longueur L_{max} de la plus longue séquence de '0'. Chaque mot codé est composé de deux parties : un préfixe et une queue. Le préfixe est utilisé pour identifier l'appartenance à un groupe et la queue pour identifier l'élément du groupe. Un exemple de code FDR est donné dans le Tableau 2.3. Connaissant la fréquence d'occurrence de chaque salve de '0' successifs, les codes les plus courts sont associés aux salves les plus fréquentes. Une table de correspondance permet de retrouver le code d'origine calculé.

Groupe	Nombre de 0 successifs	Préfixe	Queue	Code
A_1	0 1	0	0 1	00 01
A_2	2 3 4 5	10	00 01 10 11	1000 1001 1010 1011
……..				

Tableau 2.3 : Exemple de code FDR

Cette méthode de décodage des vecteurs de test permet d'obtenir une bonne compression des données de test mais demande un temps de test non négligeable et, dans la méthode présentée dans [Chan03], 3 broches de test minimum pour pouvoir assurer la synchronisation entre le testeur et le circuit.

2.1.2.4. Codes de Golomb

Le code de Golomb est un autre type de codage variable vers variable. Il se définit par deux parties distinctes : d'une part un préfixe et d'autre part une queue, l'ensemble formant le code. Le principe présenté dans [Chan00] est de coder les salves de '0' de longueur l par un code plus petit.

Le taux de compression dépend d'un paramètre m qui détermine la taille des groupes de codage où en d'autres termes, le nombre de codes possédant le même préfixe. La détermination de ce paramètre a reçu beaucoup d'attention dans la littérature, mais dans la plupart des cas, il est déterminé expérimentalement. Le nombre de groupes est déterminé par la plus longue séquence de 0. Chaque groupe de codes A_k est différencié par un préfixe composé de (k-1) 1 suivi d'un 0 noté $\{1^{(k-1)}\ 0\}$. Chaque élément est composé du préfixe précédemment défini et d'une queue de $\log_2$ m bits. Le Tableau 2.4 illustre le codage des chaînes de zéro pour m=4.

Groupe	Nombre de 0 successifs	Préfixe	Queue	Code
1	0 1 2 3	0	00 01 10 11	000 001 010 011
2	4 5 6 7	10	00 01 10 11	1000 1001 1010 1011
……..				

Tableau 2.4 : Exemple de code de Golomb pour m=4

Dans la méthode de [Chak01], une machine à états finis commence par décoder le préfixe du code de Golomb. Dans le cas où un zéro se présente en premier sur la broche de donnée, le

compteur démarre pour m cycles de décodage afin de charger la queue du code. Par contre, si le premier bit qui se présente vaut '1', alors la machine d'état décode le préfixe jusqu'à ce qu'un zéro apparaisse. Ensuite, elle démarre le décodage de la queue du code sur m bits. Le décodage de la queue prend au plus m cycles.

Comme précédemment, 3 broches testeur sont nécessaires pour décoder et synchroniser les données. Une broche pour transférer les données, une autre pour synchroniser le chargement des données et une dernière pour transmettre l'horloge.

2.1.2.5. Conclusion

Contrairement aux méthodes basées sur du BIST, ces méthodes de compression verticale, n'utilisent que les propriétés des cubes de test. Elles ne requièrent aucune connaissance structurelle du circuit en lui-même et sont donc compatibles avec le test des IP.

Ces techniques exploitent la présence de valeurs indéterminées (les X) dans les séquences pour les assigner de manière à favoriser la compression. Or, les séquences de test de la plupart des circuits industriels peuvent contenir jusqu'à 95% de valeurs non déterminées [Toub06]. Ces méthodes sont donc particulièrement efficaces en ce qui concerne la réduction du volume de données de test à transférer sur une broche du testeur.

Cependant, ces méthodes posent des problèmes en ce qui concerne la synchronisation entre le testeur et le circuit et, plus particulièrement, les chaînes de scan. En effet, les tailles de mot variables nécessitent une attention particulière puisque le chargement des chaînes de scan s'avère irrégulier. L'intégration sur la puce d'un mécanisme de synchronisation complexe est donc nécessaire en plus de l'interface de compression. Et le problème est encore bien plus important dans le cas où le circuit est doté de chaînes de scan multiples. En effet, il faut intégrer au circuit un décompresseur et un mécanisme de synchronisation par chaînes de scan, ce qui rend le coût total en silicium prohibitif. De plus, les chaînes de scan doivent être synchronisées entre elles. Cela implique de devoir attendre que la chaîne la plus longue soit complètement chargée avant d'appliquer le vecteur au circuit, ce qui pourrait, au final, pénaliser le temps de test puisque la compression nécessiterait plus de cycles testeur qu'un transfert sans compression.

2.1.3. Méthodes de compression horizontale

Dans cette partie sont présentées les principales méthodes de compression horizontale. L'objectif de ces méthodes est en premier lieu de réduire le nombre de canaux testeur requis pour transmettre les données de test à des circuits comportant plusieurs chaînes de scan. Comme il a été montré dans le chapitre précédent, si aucune méthode de compression n'est appliquée, il faut au moins une broche testeur par chaîne de scan ainsi qu'un signal de contrôle.

Cependant, comme on l'a vu dans le Chapitre 1, les circuits actuels contiennent de plus en plus de transistors, de plus en plus de fonctionnalités, et de plus en plus de bascules. Tout cela implique évidemment une augmentation du temps de test de ces circuits. Or, comme on l'a aussi vu dans le Chapitre 1, pour réduire de manière significative le temps de test, l'utilisation de chaînes de

scan multiples, plus courtes, permet de réduire efficacement ce temps. Cette solution s'avère limitée par le nombre de canaux disponibles sur les testeurs industriels actuels. Il est toutefois possible d'étendre le nombre de canaux d'un testeur, mais cela a un prix très élevé. Plusieurs méthodes ont été proposées pour permettre d'alimenter un grand nombre de chaînes de scan avec un nombre de canaux testeur limités. Elles sont détaillées dans la suite.

2.1.3.1. Méthode non compressive: Sérialisation des données

Une première solution simple et évidente est de sérialiser les données de test. Cette méthode ne réalise aucune compression à proprement parler. Elle permet juste de relier les canaux testeur aux chaînes de scan par l'intermédiaire de registres à décalage. Dans le cas où M canaux testeur sont utilisés pour alimenter N chaînes de scan, chaque slice de N bits est mise sous la forme de $\lceil N/M \rceil$ mots de M bits comme décrit sur la Figure 2.12. Chaque bit de ces mots alimente un des M registres à décalage. Après les $\lceil N/M \rceil$ décalages, les registres sont pleins et leur contenu est chargé dans les N chaînes de scan en un cycle d'horloge. Cette opération est répétée pour les L' slices (où L' est la longueur de la plus longue chaîne de scan). Un vecteur est alors complètement chargé dans les chaînes de scan. Un coup d'horloge est alors nécessaire pour appliquer le vecteur complet au circuit et récupérer la réponse.

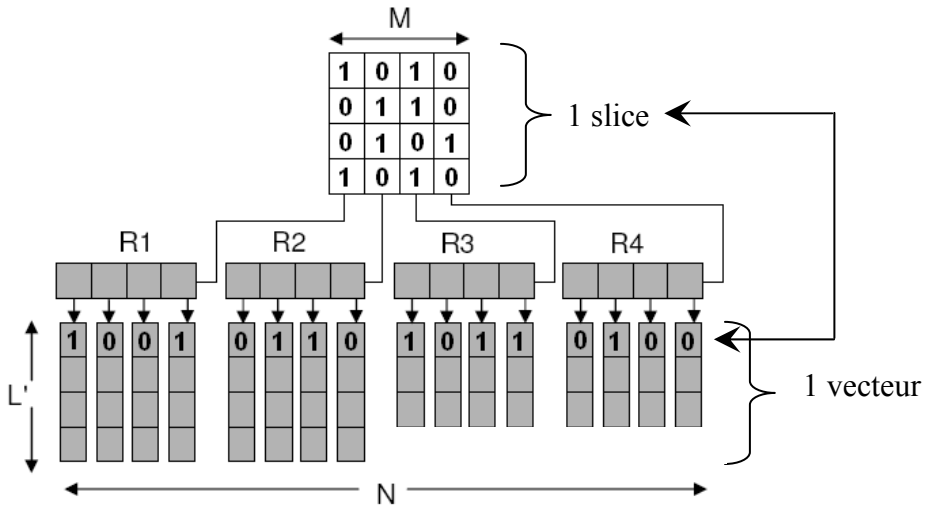


Figure 2.12 : Sérialisation des données de test.

Notons V le nombre de vecteurs de la séquence de test, L' le nombre de slices par vecteur (il s'agit aussi de la longueur de la plus longue chaîne de scan), N le nombre de chaîne de scan et M le nombre de canaux testeur disponibles. Le temps de test correspondant à cette méthode s'exprime selon la formule suivante:

$$\text{Temps de test} = V \times L' \times \left(\left\lceil \frac{N}{M} \right\rceil + 1 \right) + V + L'$$

Pour chaque slice de chaque vecteur, il faut $\lceil N/M \rceil$ cycles d'horloge pour charger les registres et 1 cycle pour charger les chaînes avec le contenu des registres. Il faut ensuite 1 cycle par vecteur pour capturer les réponses et L' cycle pour sortir la dernière réponse des chaînes de scan.

Avec un tel schéma de transmission des données de test, aucune compression n'est utilisée. Les valeurs indéterminées (X) sont donc inutiles et une séquence complètement spécifiée, et donc plus courte en termes de nombre de vecteurs (cf. Section 1.1.2.2), peut être utilisée pour réduire le temps de test.

Cette méthode très simple n'a quasiment besoin d'aucun matériel additionnel autre que les registres à décalage. Cependant, elle est très coûteuse en termes de temps de test puisqu'il est proportionnel au rapport de réduction $\lceil N/M \rceil$ entre le nombre de canaux testeur disponibles et le nombre de chaînes de scan. Or l'objectif de la compression horizontale est de fournir un grand rapport de réduction $\lceil N/M \rceil$ avec pas ou peu d'augmentation du temps de test.

2.1.3.2. Diffusion (Broadcasting)

Une première façon de réaliser la liaison entre les canaux testeur et les chaînes de scan est d'insérer un système permettant la diffusion des données de test d'une seule entrée vers plusieurs chaînes de scan. En effet, il a été remarqué qu'il arrivait fréquemment que plusieurs bascules de même rang doivent contenir la même valeur logique. Les méthodes présentées ci-dessous utilisent cette propriété pour alimenter plusieurs chaînes de scan par une même entrée de test.

2.1.3.2.1. Reconfiguration Statique

Par reconfiguration statique, nous entendons ici que la configuration du décompresseur n'est pas modifiée durant le chargement d'un vecteur de test. A l'inverse, en reconfiguration dynamique, la configuration du décompresseur est modifiée à chaque chargement de slice.

Ce type de technique est appliqué dans [Tang03]. Les auteurs décrivent une technique basée sur l'utilisation de switchs reconfigurables (Figure 2.13-a) qui permettent de changer les liaisons entrées – chaînes de scan. Dans cet exemple, le switch, connecté aux 2 entrées testeur, alimente soit les chaînes (1,2) et (3,4) dans la configuration de la Figure 2.13-b, soit les chaînes(1,3) et (2,4) dans la configuration de la Figure 2.13-c.

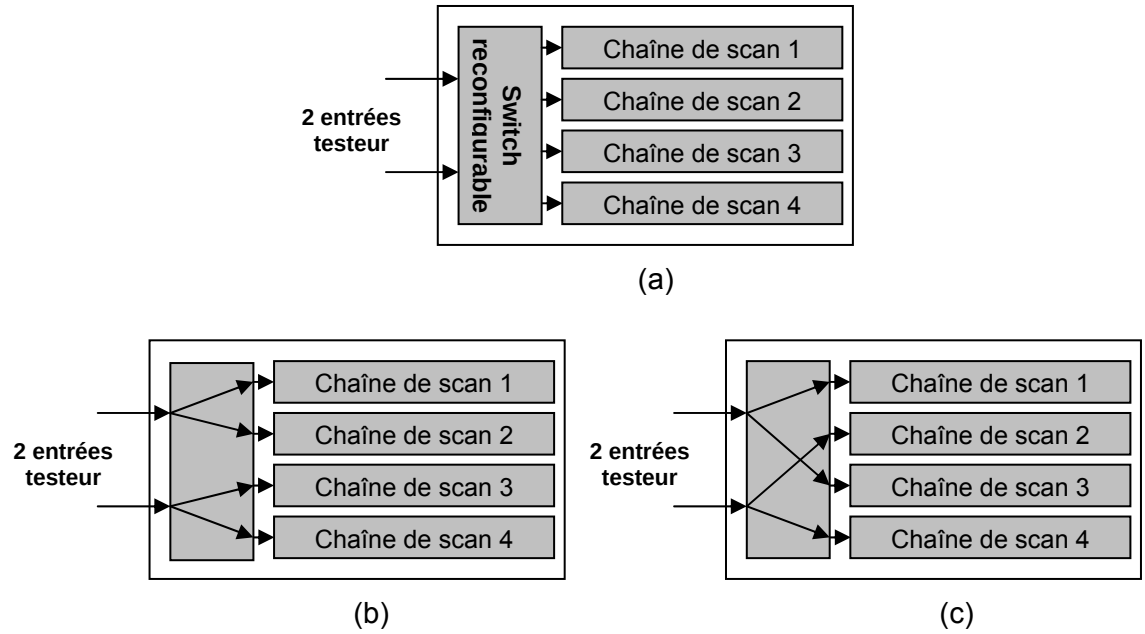


Figure 2.13 : Switch reconfigurables pour 2 entrées et 4 chaînes de scan

L'alimentation de plusieurs chaînes de scan par une même entrée suppose que ces chaînes contiennent des valeurs compatibles. Le contenu de deux chaînes est dit compatible si toutes les valeurs spécifiées d'un vecteur dans les bascules de même rang sont égales (Figure 2.14).

Chaîne 1:	1	X	X	0	X	0	1
Chaîne 2:	X	0	1	0	0	0	X

(a)

Chaîne 1:	1	X	X	**0**	X	0	1
Chaîne 2:	X	0	1	**1**	0	0	X

(b)

Figure 2.14 : Exemple de chaînes de scan compatible (a) et incompatible (b)

Les auteurs proposent une heuristique pour minimiser le nombre de configurations (recherche d'une solution minimale dans un graphique d'incompatibilités).

Il faut remarquer que cette structure est indépendante du circuit sous test. Elle est basée sur un modèle appelé "réseau oméga" déjà utilisé dans le cadre de réseaux multiprocesseurs [Patt90]. Ce modèle utilise des switchs 2x2 élémentaire décrit Figure 2.15. Plusieurs de ces switchs sont assemblés pour former le décompresseur MxN qui nous intéresse ici.

Les signaux de commande nécessaires au changement de configurations de ce décompresseur sont générés par un contrôleur intégré. Il peut être conçu à partir d'un registre ou d'un compteur avec un décodeur approprié.

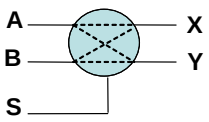


Figure 2.15 : Switch élémentaire du réseau oméga

L'inconvénient majeur de cette structure réside dans le contrôleur utilisé pour commander le switch. En effet, la taille sur la puce peut varier énormément en fonction du nombre minimal de configurations requis pour acheminer toutes les données de test. De plus, une fois la structure du décompresseur établie (switch+contrôleur), il est impossible de modifier les données de test puisque la compatibilité des chaînes de scan n'est plus garantie. Le décompresseur est ici indépendant du circuit sous test mais pas des données à compresser.

2.1.3.2.2. Reconfiguration dynamique

Une autre méthode appelée "circular scan" [Orai04] est basée sur le principe de la reconfiguration des entrées des chaînes de scan. La reconfiguration s'effectue à chaque chargement d'une nouvelle slice. Les configurations sont envoyées par le testeur au décodeur sur puce. Le principe général ici est de boucler les sorties des chaînes de scan sur leur entrée respective et de ne modifier que les bits spécifiés de chaque slice.

L'architecture est donnée Figure 2.16. Elle comprend un décodeur permettant d'alimenter les chaînes de scan avec une seule entrée de donnée de test. Le décodeur est commandé par les entrés de sélection. Ces entrées sont destinées à transmettre l'adresse de la chaîne de scan qui doit recevoir la donnée de test. Soit M le nombre de canaux testeur disponibles, cette structure peut alimenter jusqu'à 2^{M-1} chaînes de scan. A chaque décalage, le décodeur désigne la chaîne qui doit recevoir la donnée de test. Le décodeur commande pour cela les multiplexeurs présents à l'entrée de chaque chaîne. Le multiplexeur correspondant de la chaîne sélectionnée transmet la donnée de test, les autres rebouclent les chaînes restantes sur elle-même. Le premier vecteur doit être chargé intégralement en série, chaîne par chaîne. Les réponses servent ensuite de motifs à partir desquels les vecteurs suivants sont reconstruits.

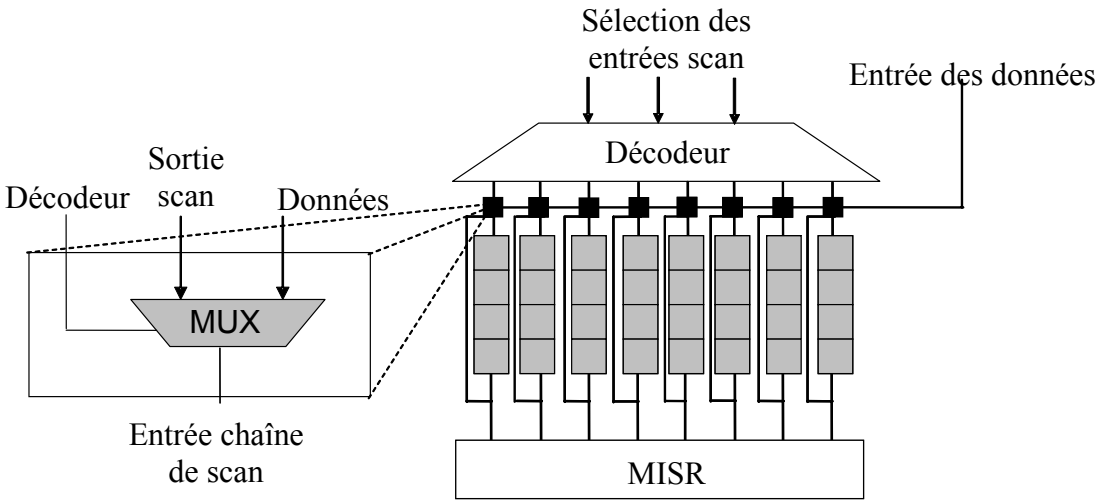


Figure 2.16 : Architecture Circular Scan

Cette méthode exploite, de la même façon que les techniques de compression verticale, le faible nombre de valeurs spécifiées dans les cubes de test [Toub06]. Par conséquent, il n'y a que très

peu de valeurs spécifiées dans chaque slices. De plus, si la valeur spécifiée correspond à la valeur présente dans la réponse, lors du bouclage, il ne sera pas nécessaire de la modifier.

Avec cette structure, on ne peut charger dans les chaînes qu'une seule valeur par slice. Si plusieurs valeurs doivent être modifiées, il faut faire tourner l'ensemble du vecteur pour charger une autre valeur. C'est donc la slice qui contient le plus grand nombre de valeurs à modifier qui définit le nombre de rotations du vecteur. Or, plus le nombre de rotations d'un vecteur est grand, plus le temps nécessaire pour l'application de ce vecteur est grand.

Les auteurs ont proposé dans [Arsl04] une amélioration de cette structure visant à réduire le nombre de rotations nécessaires à l'établissement des valeurs spécifiées dans les vecteurs. Ils proposent une architecture qui permet des décalages dans les chaînes de scan, non seulement dans le sens vertical (sens utilisé habituellement), mais aussi dans le sens horizontal. L'architecture des bascules est maintenant définie sous la forme d'une grille (Figure 2.18). Les bascules sont modifiées pour permettre ce nouveau type de décalage (Figure 2.17). Un contrôleur de direction des rotations (DR) commande les nouvelles bascules.

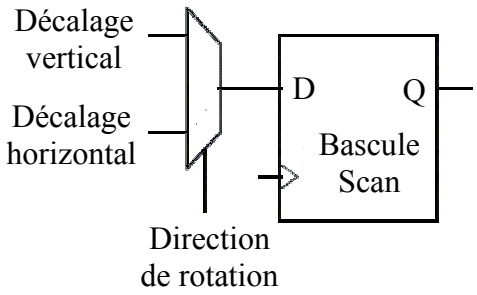


Figure 2.17 : Bascule Pour le Circular Scan modifié

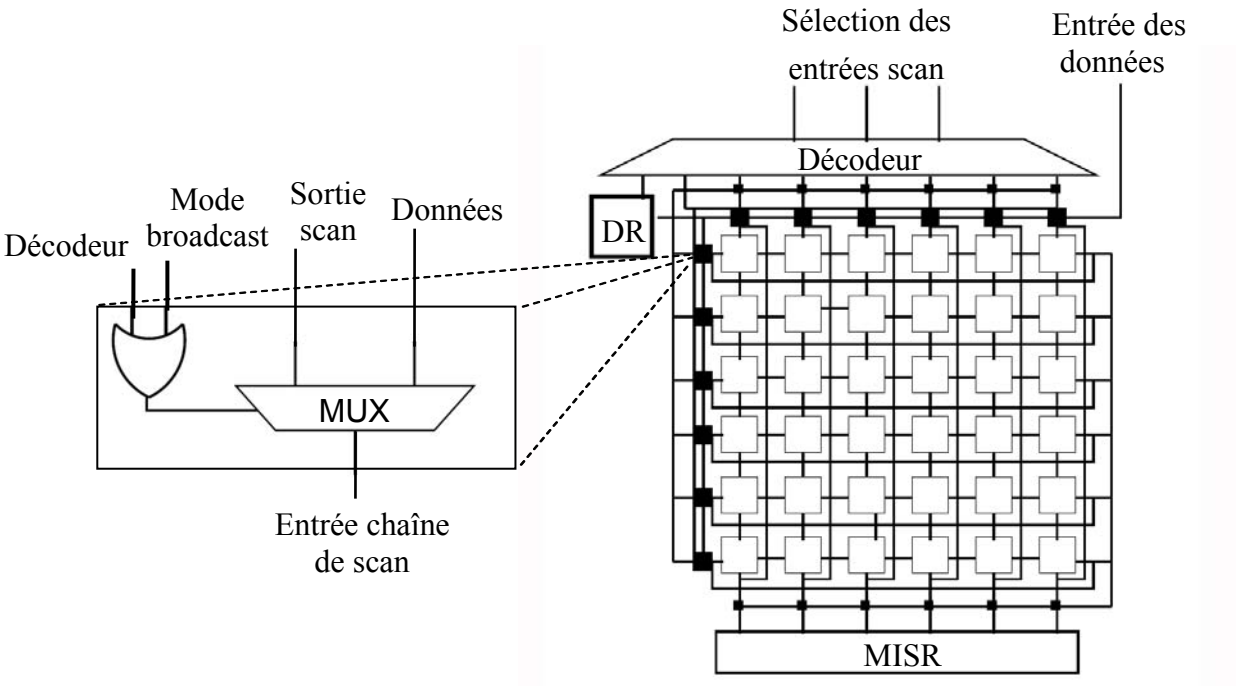


Figure 2.18 : Architecture Circular Scan Modifiée

Cette méthode permet de ne pas porter atteinte au taux de couverture de fautes de la séquence. Elle risque toutefois d'augmenter sévèrement le temps de test puisque certaines slices peuvent demander plusieurs rotations. De plus, pour que cette méthode fonctionne de manière optimale, les auteurs utilisent des vecteurs générés spécifiquement pour cette architecture. Cela suppose de connaître les détails architecturaux du circuit. Cette méthode est donc difficilement applicable au test d'IP.

2.1.3.2.3. Méthodes "Illinois scan"

Les auteurs de [Hamz99] proposent de partitionner les chaînes de scan en plusieurs sous-ensembles (segments) de scan, chaque ensemble requérant une broche testeur. L'architecture présentée Figure 2.19 possède deux modes de fonctionnement en fonction de la valeur présente dans la bascule de contrôle CFF:

- Dans un premier temps la chaîne de scan est subdivisée en n segments, tous chargés en parallèle à partir de l'entrée de scan_in (SI). Il s'agit du mode "émission". Il en résulte que les bascules de même rang dans les différents segments reçoivent la même donnée.

- Dans un deuxième temps, comme certaines fautes ne peuvent êtres détectées par des vecteurs composés de n segments strictement identiques, la chaîne de scan est chargée de façon série comme dans une architecture classique. Les fautes non détectables sous le premier mode de configuration deviennent ainsi détectables s'il ne s'agit pas de fautes dites "non testables".

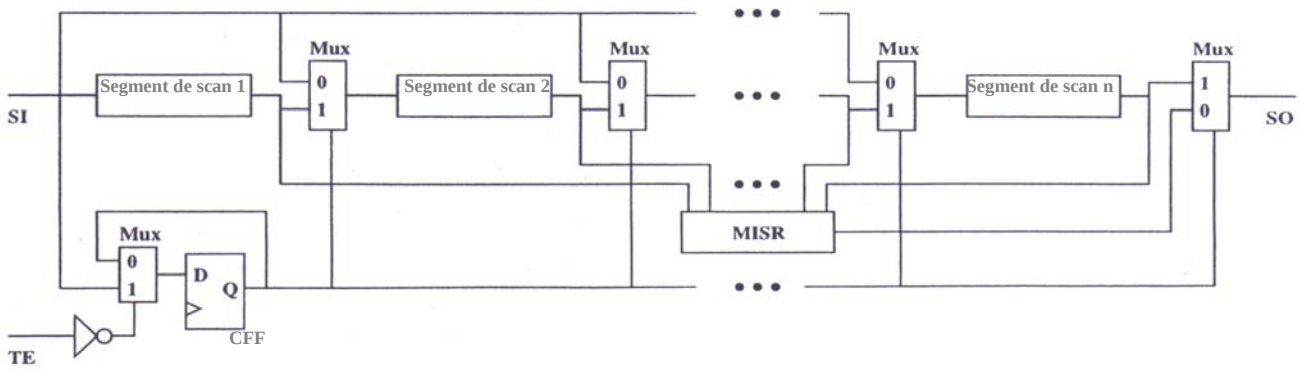


Figure 2.19 : Architecture Parallèle-Série Full Scan (PFFS)

Un générateur de vecteurs de test spécifiquement conçu pour cette architecture est utilisé pour générer des vecteurs de test applicables en mode émission. Les vecteurs appliqués en mode série sont, bien sûr, plus coûteux en termes de nombre de bits et de temps d'application que les vecteurs appliqués en mode émission. Les auteurs ont donc proposé plusieurs améliorations à cette idée de base de façon à réduire le nombre de vecteurs chargés en série.

Dans une première version de ces travaux, le nombre de segments n et le nombre k de bascules qui les composent sont choisis par le concepteur. Cependant, il est difficile de prédire sans recours à la simulation quelle configuration mène à la réduction maximale du temps de test et du volume de données.

Dans [Pate02], les auteurs proposent un algorithme incrémental permettant de parcourir un ensemble de solutions architecturales et de trouver, parmi ces solutions, la plus performante. Ils utilisent les propriétés arithmétiques de décomposition du nombre k de bascules des segments en puissance de 2 pour déterminer des règles sur les performances de ces différentes architectures (par exemple 1024, 512, 256, 128, 64, 32, 16, 8, 4, 2, 1… bascules par segment)

Dans [Pand02], les auteurs utilisent une architecture double : une première où les segments contiennent k bascules (déterminé par l'algorithme incrémental précédent), et une seconde, où, après un ré-aiguillage de la chaîne permettant de les reconfigurer, les segments contiennent k+1 bascules. Les fautes non détectées dans le premier mode émission peuvent devenir détectables dans le second puisque les contraintes de similitude entre données mémorisées dans les bascules de même rang dans le premier mode n'existent plus dans le deuxième. Le mode série classique n'est utilisé que pour les fautes restantes.

Les auteurs de [Chan07] décrivent une technique de reconfiguration statique (à chaque vecteur) de l'ordre des bascules à l'intérieur des segments. L'utilisation de ces multiples configurations permet de briser les dépendances linéaires entre les segments et ainsi améliorer le taux de couverture des vecteurs transmis en parallèle. De fait les auteurs annoncent qu'il n'est plus besoin d'avoir recours à des vecteurs de top-up chargés en série pour garantir le taux de couverture.

2.1.3.2.4. Méthode industrielles (Synopsys, Syntest)

Cette technique Illinois Scan est à l'origine de l'outil développé par Synopsys®et commercialisé sous le nom de DFTMAX®. Un décompresseur, basé sur leur technologie baptisée "Adaptive Scan", alimente les chaînes de scan avec les vecteurs compressés transmis par le testeur.

Le schéma de l'architecture RSSA (pour "Reconfigurable Shared Scan-in Architecture") publiée dans [Sama03] est donné Figure 2.20. Il s'agit, là encore, de couper les M chaînes de scan en N segments et d'alimenter des groupes de segments par une même valeur venant d'une même entrée testeur. La constitution des segments est basée sur une analyse de compatibilité des chaînes de scan afin d'optimiser l'architecture résultante.

La différence avec l'architecture PSFS de l'Illinois Scan se situe au niveau de l'alimentation des segments. Dans l'Illinois Scan, les segments sont les "morceaux" d'une même chaîne de scan et sont alimentés par l'entrée testeur qui alimentait cette chaîne originale. Dans l'architecture RSSA de l'Adaptive Scan, les segments peuvent appartenir à un groupe différent pour chaque nouveau vecteur transmis (par exemple, avec 4 segments et 2 entrées, pour un premier vecteur, les 2 groupes seraient constitués respectivement des segments 1 et 2 et des segments 3 et 4, puis pour un second vecteur, respectivement des segments 2 et 4 et des segments 1 et 3). La reconfiguration est rendue possible par la présence des multiplexeurs en tête des segments de chaînes de scan.

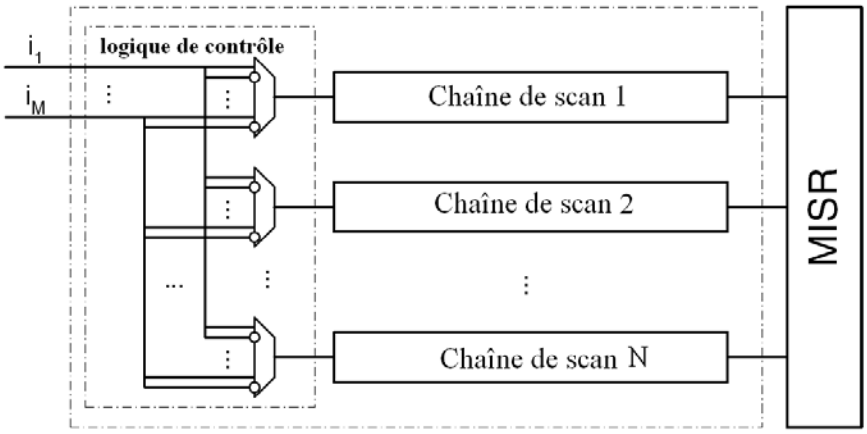


Figure 2.20 : Architecture RSSA de la technologie "Adapative Scan"

La génération des vecteurs destinés à cette architecture reconfigurable repose sur un générateur de vecteur de test spécifique. En effet, la procédure de génération est la suivante:

1) sélectionner la première configuration des segments

2) définir la liste de fautes et lancer une analyse de compatibilité pour définir la prochaine configuration

3) générer les vecteurs pour la configuration courante

4) répéter les étapes 2 et 3jusqu'à ce que toutes les fautes détectables soient testées (dans le pire des cas, un vecteur par faute)

Il faut donc recourir à une suite d'outils spécialement conçus pour cette architecture. Synopsys® fournit, bien évidemment, cette suite d'outils mais il est impossible à un concepteur ne disposant pas de la suite Synopsys® d'utiliser une telle méthode de compression des données. De plus, il est également impossible à un concepteur souhaitant tester un IP, sur lequel, rappelons-le, il est impossible d'utiliser un ATPG ou de la simulation de fautes, d'utiliser un telle architecture de test.

Les mêmes auteurs ont également publié une évolution de cette architecture RSSA dans [Sitc04]. Ils intègrent ici la reconfiguration dynamique des chaînes de scan. A la différence de la reconfiguration statique qui a lieu à chaque nouveau vecteur, la reconfiguration dynamique a lieu lors de chaque décalage des différentes valeurs d'un même vecteur dans les chaînes de scan. En fonction d'un signal de contrôle co, les multiplexeurs de la Figure 2.21 (a) peuvent reconfigurer les liaisons entre les canaux testeurs si1 et si2 et les 4 chaînes de scan.

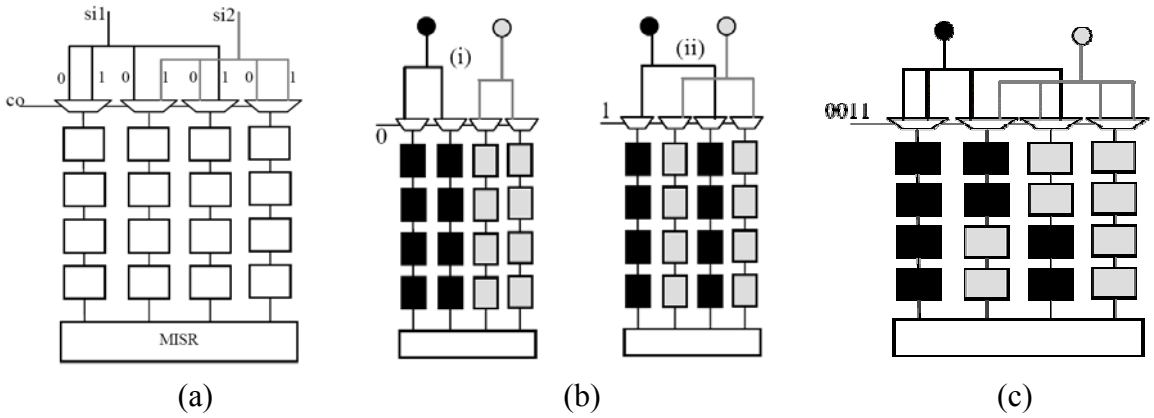


Figure 2.21 : Reconfiguration dynamique de l'architecture RSSA

Sur cet exemple, les 16 bascules du circuit sont réorganisées en 4 chaînes de scan. Ces chaînes sont connectées à 2 canaux testeur par l'intermédiaire des multiplexeurs. Dans une première configuration (Figure 2.21 (b)-(i)), les 2 premières chaînes sont connectées à la première broche testeur si1 et les 2 dernières à la deuxième broche testeur si2. Dans la seconde (Figure 2.21 (b)-(ii)), ce sont les chaînes 1 et 3 qui sont connectées à la broche si1 et les chaînes 2 et 4 à la broche si2. A chaque décalage, il est possible d'utiliser une configuration des multiplexeurs différente de la précédente. Il résulte de cette reconfiguration dynamique un positionnement des données de test dynamique dont un exemple est donné Figure 2.21 (c). Lors des deux premiers décalages, co vaut '0' et la première configuration est sélectionnée. Lors des deux décalages suivants, co vaut '1', et la seconde configuration est sélectionnée.

Les remarques concernant le test des IP faites pour la première version de l'architecture RSSA sont, bien entendu, toujours valables dans ce cas. Il faut rajouter à cela une attention toute particulière à apporter à la synchronisation entre les signaux de commande et le décalage dans les chaînes de scan.

Une autre technologie commercialisée par SynTest® tire également profit de la reconfiguration dynamique en l'appliquant cette fois à un réseau de décompression combinatoire composé de portes logiques. Cette technologie baptisée "VirtualScan" et publiée dans [Wang04], comme celle commercialisée par Synopsys®, associe une architecture de test et un générateur de vecteurs spécifiques. Là encore, les deux éléments sont indissociables, privant encore le concepteur d'une liberté de choix sur la méthode de compression des données de test à employer.

L'architecture générale du test est sensiblement similaire à la structure de Synopsys®. Les bascules d'une chaîne de scan originale sont reparties équitablement en K nouvelles chaînes. Le décompresseur décrit Figure 2.24 permet de relier les M canaux testeurs aux N=K*M chaînes de scan. Le décompresseur décrit se compose de plusieurs blocs: un réseau de décompression combinatoire, un bloc de connexions aux chaînes de scan (composé de multiplexeurs), et d'un contrôleur. Ce contrôleur peut être soit combinatoire, soit séquentiel. Il contrôle le réseau de

décompression (de manière à réduire les corrélations entre les valeurs de test) et les multiplexeurs du bloc de connexions. Ce signal permet de revenir à une configuration de l'architecture sans compression (M canaux testeur pour M chaînes de scan) de manière à pouvoir transmettre des vecteurs de top-up qui pourraient s'avérer nécessaires.

Le générateur de vecteurs de test produit directement les vecteurs compressés, les plus optimisés, destinés à cette architecture VirtualScan en une seule passe (au lieu de deux pour les méthodes non industrielles: une passe pour la génération des vecteurs, et une passe pour la compression de ces vecteurs).

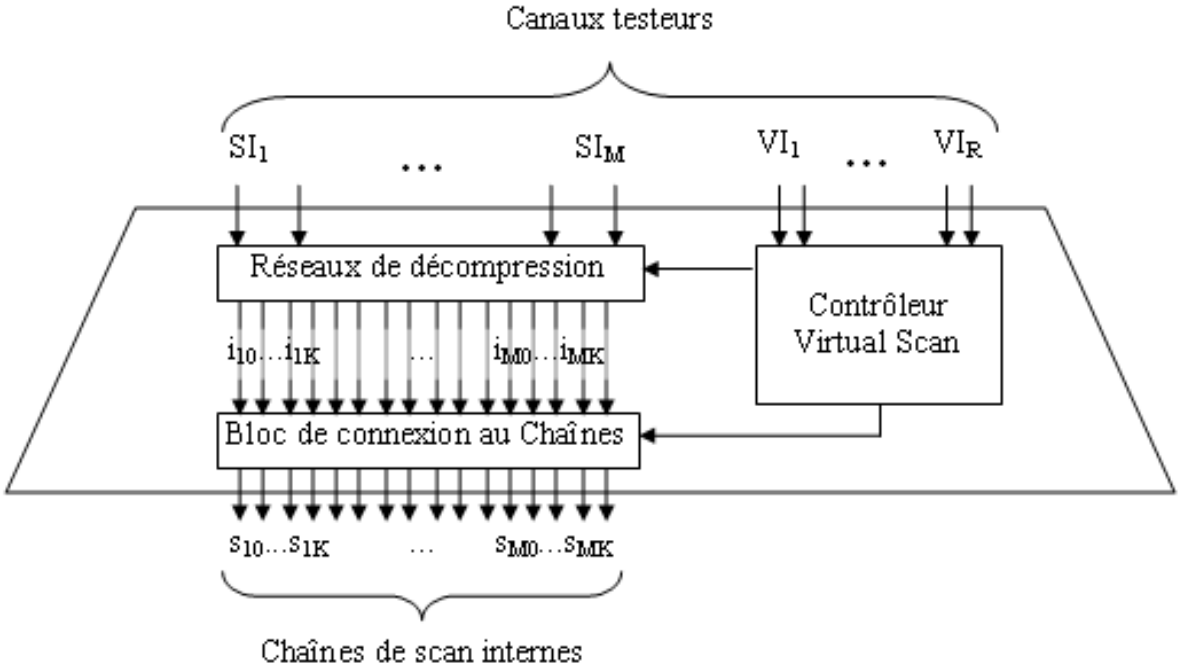


Figure 2.22 : Décompresseur VirtualScan de SynTest

Ces méthodes industrielles exploitent donc le principe de la diffusion des données de test en y intégrant la reconfiguration, qu'elle soit statique ou dynamique, des connexions entre les canaux testeurs et les chaînes de scan. Elles sont extrêmement performantes en termes de réduction du temps de test et du volume de données à stocker. Cependant, dans le cas de test d'IP, les données de test sont fournies avec le circuit et il n'est pas possible d'utiliser un générateur de vecteur externe. De plus, les chaînes de scan ne peuvent en aucun cas être modifiées. Dans ces cas là, ces méthodes de diffusion ne peuvent pas être utilisées.

2.1.3.3. Décompresseur linéaire

Une autre catégorie de techniques de décompression est basée sur l'utilisation de décompresseur linéaire. Ces structures représentent un système d'équations linéaires AX=Y, où X représente la slice compressée et Y la slice originale à envoyer au circuit. Compresser un cube de test revient à résoudre ce genre de système. Un cube de test est compressible si et seulement si la dimension du système est au moins égale au nombre de bits spécifiés du cube de test. Il est donc

très possible de ne pas pouvoir compresser un cube. Cependant, dans le cas de décompresseurs possédant un polynôme primitif (un LFSR par exemple), si le nombre de variables libres est suffisamment élevé par rapport au nombre de bits spécifiés, la probabilité de ne pas pouvoir encoder un cube de test devient négligeable [Toub06]. Ci-dessous se trouvent les principales méthodes publiées se basant sur l'utilisation de décompresseurs linéaires, qu'elles soient combinatoires ou séquentielles.

2.1.3.3.1. Décompresseur linéaire à structure combinatoire

Un premier groupe de méthodes utilise des décompresseurs linéaires combinatoires. Tout d'abord dans [Bayr01], un réseau de portes XOR est utilisé pour décompresser les vecteurs. Pour une compression de N canaux testeurs vers M chaînes de scan, le réseau de décompression (Figure 2.23 (b)) correspond à un ensemble de M équations linéaires à N inconnues, dépendant des valeurs du vecteur scan à appliquer (Figure 2.23 (a)). En effet, le réseau de décompression est la transposition combinatoire d'un LFSR de longueur M dont le polynôme est choisi de façon à limiter les dépendances linéaires entre les N chaînes de scan. Les équations sont celles du LFSR.

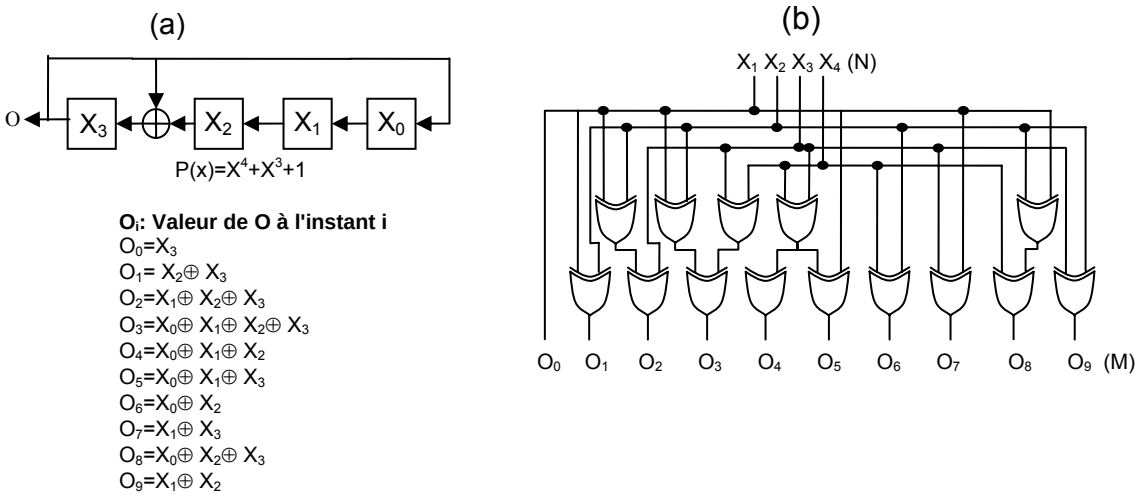


Figure 2.23 : Séquence LFSR et réseau de décompression associé

Il est possible que le système d'équations n'ait pas de solution pour certains cubes de test, en d'autres termes que ces cubes ne soient pas compressibles. Le taux de couverture de fautes obtenu avec la séquence initiale non compressée ne peut donc être garanti en appliquant uniquement les cubes initiaux compressés.

Pour s'approcher de la couverture de fautes initiale, il est possible de compléter la séquence compressée par d'autres vecteurs devant être compressibles. Ces vecteurs sont soit aléatoires, soit déterministes. Dans ce dernier cas, il est nécessaire de relancer le générateur de vecteurs en incluant le décompresseur dans la description du circuit.

Que ce soit pour des vecteurs aléatoires ou des vecteurs déterministes, il est nécessaire de pouvoir faire de la simulation de fautes et donc de disposer de la description structurelle du circuit, ce qui n'est pas toujours possible en particulier dans le cas du test d'IP.

Pour s'assurer de la compressibilité de tous les cubes de test, on peut ajouter à un réseau de portes logiques, un système de reconfiguration qui va permettre de modifier l'espace de sortie du décompresseur linéaire, et ainsi de disposer au final de plusieurs réseaux de décompression. Cette idée est présentée dans [Bala05].

Les auteurs proposent d'insérer des multiplexeurs entre les sorties du réseau de portes et les chaînes de scan (Figure 2.24). Ces multiplexeurs sont commandés par des bits de configurations qui peuvent être soit stockés dans une ROM sur puces, soit envoyés par le testeur par des canaux additionnels. Les différentes configurations nécessaires sont obtenues par des manipulations algébriques basées sur la technique du pivot de Gauss.

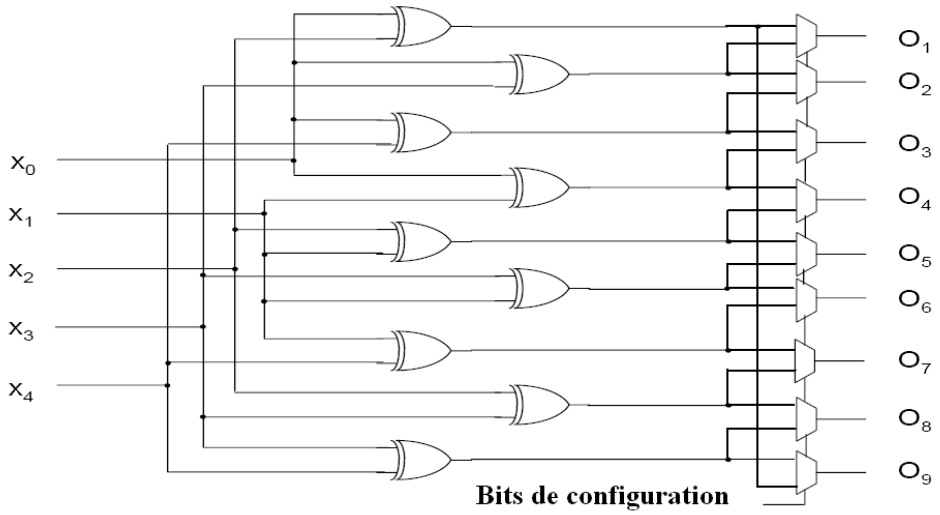


Figure 2.24 : Décompresseur linéaire reconfigurable

Le nombre de configurations possible est toutefois limité par le nombre de bits de configuration. Or, une fois ce nombre fixé, si la séquence vient à être modifiée (par exemple par l'ajout de vecteurs additionnels lors de la phase de production), il n'est pas garanti que la structure de décompression soit à même de pouvoir générer les vecteurs additionnels. En effet, ces structures basées sur des décompresseurs linéaires combinatoires ne disposent pas d'un système permettant de transférer d'éventuels vecteurs de top-up qui pourraient être utilisés soit pour garantir le taux de couverture, soit lors de l'ajout de vecteurs à la séquence originale lors de la phase de production du circuit par exemple.

2.1.3.3.2. Décompresseur linéaire à structure séquentielle statique

A la différence des décompresseurs linéaires combinatoires qui sont basés sur un réseau de portes logiques (le plus souvent des XORs), les décompresseurs séquentiels sont construits à partir de machines d'état fini (comme par exemple un LFSR). Une méthode utilisant ce genre de structure est celle publiée dans [Kone91]. Les auteurs proposent une architecture basée sur un LFSR qui est

réinitialisé à chaque vecteur de manière à pouvoir diriger la séquence produite par le LFSR. Il ne s'agit pas là de reconfiguration statique mais de réinitialisation statique.

Le testeur est chargé d'envoyer les graines pré-calculées (de la même manière que précédemment, en résolvant le système linéaire AX=Y) pour fixer les valeurs des slices générées par le LFSR. La taille des graines est fixée par le nombre maximum s de bits spécifiés dans un vecteur. Les auteurs ont prouvé qu'avec une taille de graine égale à s+20, la probabilité qu'une solution au système linéaire n'existe pas est inférieure à 10^{-6}. Entre le chargement de deux graines, le LFSR génère les slices de manière autonome. Cela implique de porter une attention particulière à la synchronisation entre le décompresseur et le testeur.

D'autres auteurs proposent de compresser la graine du LFSR avec une technique d'encodage statistique [Kris02]. Ils mettent en avant le fait qu'il vaut mieux encoder les graines, beaucoup moins nombreuses, plutôt que directement les slices des cubes de test. Un autre avantage dans le fait de compresser les graines plutôt que les slices est qu'il est possible de choisir une graine différente dans l'espace de solution du LFSR alors que les valeurs spécifiées des cubes de test ne sont pas modifiables. Les auteurs proposent une architecture, présentée Figure 2.25, où un décodeur (la technique employée ici est la compression de Huffman, le décodeur est réalisé à partir d'une machine d'état fini) va alimenter le LFSR, accompagné d'un "Phase Shifter" de manière à diminuer les corrélations entre les sorties du LFSR en mode autonome.

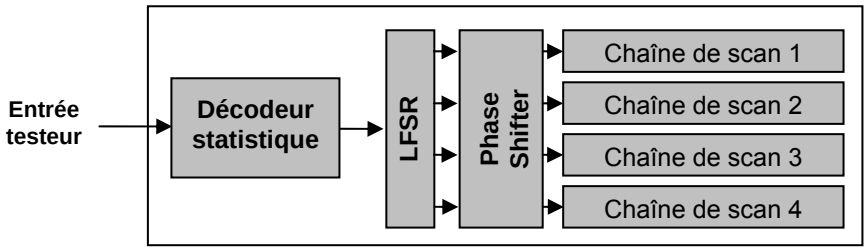


Figure 2.25 : Structure combinée décompresseur séquentiel et encodage statistique

2.1.3.3.3. Décompresseur linéaire à structure séquentielle dynamique

L'inconvénient de ces méthodes dites statiques réside dans le fait que le testeur est inutilisé pendant les phases de fonctionnement autonomes du LFSR (entre chaque nouveau vecteur) alors que la génération des slices est stoppée lors de l'envoi de la graine suivante. On constate là un manque d'efficacité de ces structures.

Pour pallier ce problème, les auteurs de [Kris01] proposent une méthode de réinitialisation dynamique dans laquelle la graine est modifiée de manière incrémentale lors de la génération des slices. La technique est illustrée Figure 2.26. Une porte XOR est rajoutée dans la boucle du LFSR. Dans le cas d'un LFSR d'une taille de r = s+20 bits (où s est le nombre maximal de bits spécifié dans un cube de test de la séquence complète), une structure classique devrait nécessiter une graine de l = r bits. Or, avec cette méthode, les auteurs proposent d'utiliser une graine plus petite (l < r).

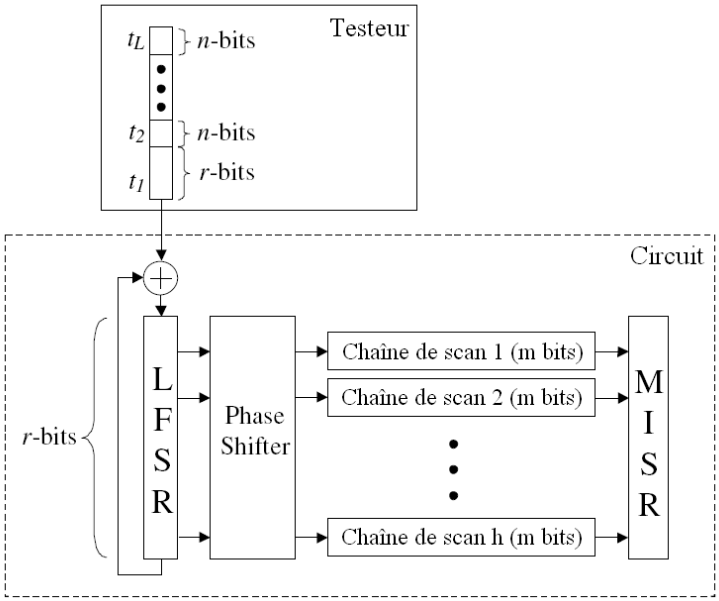


Figure 2.26 : Méthode de réinitialisation dynamique d'un décompresseur linéaire séquentiel

Une première graine de r bits est envoyée au LFSR pour le premier vecteur. A partir du second vecteur, à chaque décalage dans les chaînes de scan, un bit de la graine est décalé dans le LFSR par le XOR. Une fois les n bits de la graine transmis, le LFSR fonctionne de manière autonome comme dans le cas d'une réinitialisation statique.

La détermination de ces graines repose, comme dans le cas précédent sur des résolutions algébriques de systèmes d'équations linéaires. Or ces équations dépendent des valeurs spécifiées dans les cubes de test et du polynôme caractéristique du LFSR. On remarquera que, comme dans le cas des décompresseurs linéaires combinatoires, l'architecture de test dépend des données de test et n'est plus valable pour une séquence de test qui devrait être modifiée.

2.1.3.3.4. Méthodes industrielles (Mentor)

On peut également trouver des méthodes basées sur des décompresseurs linéaires dans les outils commerciaux. On citera notamment celle commercialisée par Mentor Graphics®, baptisée TestKompress®. Il s'agit d'une suite complète d'outils de conception en vue du test intégrant leur technologie "Embedded Deterministic Test" (EDT), publiée dans [Rajs02]. La structure générale du décompresseur est donnée Figure 2.27. Il est constitué d'un générateur en anneau (ring generator) et d'un déphaseur (phase shifter).

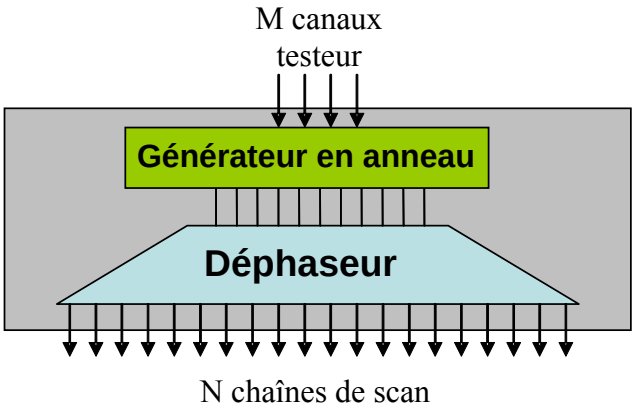


Figure 2.27 : Technologie EDT de Mentor Graphics®

Le générateur en anneau est constitué de portes XORs et de bascules. Il est obtenu à partir du polynôme caractéristique d'un LFSR, comme sur l'exemple de la Figure 2.28. L'avantage par rapport à un simple LFSR est de réduire les temps de chargement des valeurs d'initialisation grâce à une distribution régulière des entrées au sein de la structure (dans l'exemple, une entrée toutes les 4 bascules).

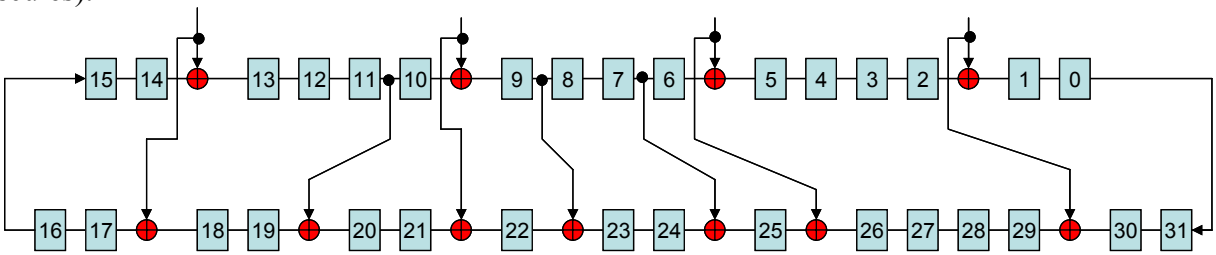


Figure 2.28 : Générateur en anneau implémenté à partir du polynôme $x^{32}+x^{18}+x^{14}+x^{9}+1$

Cette méthode, au demeurant très efficace en termes de réduction de temps de test et de volume de données, présente cependant les mêmes défauts vis-à-vis du test d'IP que ses concurrentes de Synopsys® et SynTest®. En effet, la génération de vecteurs de test pouvant être compressés par cette structure nécessite l'utilisation d'un générateur de vecteurs de test spécifiquement conçu pour cette structure. Or, comme on l'a déjà vu plus haut, le test de circuits IP implique de ne pas avoir accès au circuit proprement dit. Autrement dit, on ne peut appliquer aucune simulation de fautes ni aucun outil de génération de vecteurs de test, les données de test étant fournies par le concepteur de l'IP.

2.1.3.4. Méthode par dictionnaire

Enfin, la dernière famille de techniques de compressions horizontale des données de test consiste à stocker tout ou partie des vecteurs dans un dictionnaire (mémoire ROM ou RAM) intégré sur la puce. Dans le cas de circuits avec N chaînes de scan, le vecteur est découpé en slices de N bits. Les mots à stocker dans le dictionnaire sont ici les slices des vecteurs. Avec une telle structure, le testeur n'envoie pas le vecteur, mais son adresse dans la mémoire sur puce (Figure 2.29). La

compression est effective si l'adresse du mot, codée sur M bits, est plus petite que le mot lui-même. Cette condition est vérifiée si $M = \log_2(d) < N$, avec d le nombre de slices différentes et N le nombre de chaînes de scan. En d'autres termes, il suffit que toutes les différentes 2^N slices n'apparaissent pas dans la séquence de test. Cependant, même si cette condition est vérifiée, un autre problème se pose: la taille du dictionnaire sur puces. Cette double contrainte impose donc de chercher à réduire le nombre de mots de N bits à stocker dans le dictionnaire.

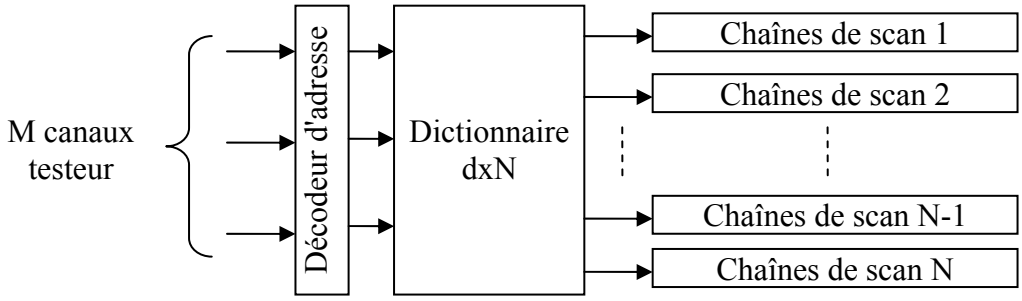


Figure 2.29 : Schéma général de compression par dictionnaire

Une première façon de réduire la taille du dictionnaire est de ne pas y intégrer toutes les slices de la séquence. Cette méthode est utilisée dans [Li03]. Seules les slices dont la fréquence d'apparition dans la séquence est la plus grande sont stockées dans le dictionnaire. Pour les autres, un système de bypass du dictionnaire est prévu de manière à ne pas pénaliser le taux de couverture de la séquence. Cela s'effectue bien évidemment au détriment du nombre de cycles de chargement ($\lceil N/M \rceil$ cycles pour chaque slice ne figurant pas dans le dictionnaire). L'idée est donc de minimiser le nombre de slices n'entrant pas dans le dictionnaire. Les auteurs proposent un algorithme exploitant la présence de X dans la séquence en les assignant de manière à réduire le nombre de ces slices.

Une variant de cette méthode a été présentée dans [Wurt04]. Cette fois aucun système de bypass n'est prévu. Cela implique de traiter toutes les slices de la séquence. Pour réduire la taille du dictionnaire, les auteurs proposent d'ajouter une structure permettant de modifier le mot sélectionné dans le dictionnaire. Cette technique, inspirée du "bit flipping", permet de modifier 1 bit de chaque mot du dictionnaire. Chaque mot du dictionnaire peut donc générer N mots différents (ceux dont la distance de Hamming du mot présent dans le dictionnaire vaut 1). Selon les auteurs, cette technique permet de réduire nettement le nombre de slices à stocker dans le dictionnaire.

Cependant, pour générer une slice avec cette technique, le testeur transmet au décodeur non seulement l'adresse du mot du dictionnaire, mais aussi la position du bit à corriger ($\log_2(N+1)$ bits pour coder une correction sur l'une des N chaînes de scan ainsi que pour coder l'absence de correction) . Les données envoyées par le testeur pour chaque slice sont de la forme décrite dans la Figure 2.30.

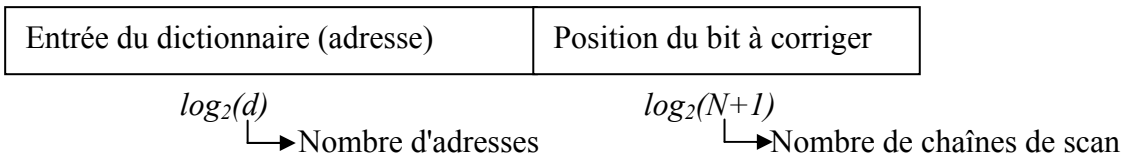


Figure 2.30 : Format des données du codage par dictionnaire avec correction

L'architecture de test est décrite Figure 2.31. On y retrouve le dictionnaire, comme dans la méthode précédente. Le correcteur, composé d'un décodeur et de N portes XOR, est commandé par les $\log_2(n+1)$ bits venant du testeur. Là encore, les valeurs non spécifiées des cubes de test sont assignées par un algorithme similaire au précédent, mais tenant compte cette fois de la compatibilité des vecteurs avec une distance de Hamming de 1, afin de minimiser la taille du dictionnaire.

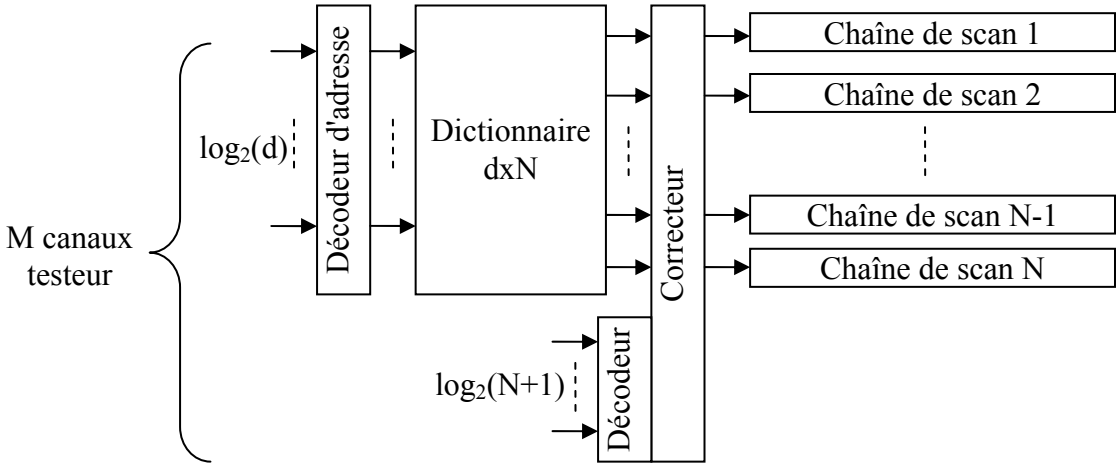


Figure 2.31 : Architecture de test pour un codage par dictionnaire avec une correction

Comparée à la méthode précédente, l'utilisation du système de correction nécessite un surplus de surface de silicium pour le correcteur. De plus le nombre de canaux testeur nécessaires au transfert des données est fixé par la taille du dictionnaire, et, par conséquent, par la séquence de test. D'un autre coté, la correction permet de toujours générer une slice à chaque cycle d'horloge, ce qui rend l'architecture régulière et parfaitement prévisible en termes de temps de test.

En revanche, le point commun à toutes les méthodes basées sur l'utilisation de dictionnaire, est le fait que le dictionnaire doit être construit en fonction de la séquence de test. Une fois de plus si la séquence de test est modifiée, le décompresseur risque de ne plus assurer son rôle correctement.

2.1.4. Adéquation Compression des données de test et Systèmes sur puces

Dans cet état de l'art, nous avons vu qu'il existe de très nombreuses méthodes de compression de données. Ces méthodes peuvent être classées dans trois grandes familles: Le BIST, la compression verticale et la compression horizontale.

Les techniques de BIST tout d'abord, consistent principalement à générer de manière pseudo aléatoirement les vecteurs au sein même du circuit. Cependant, le faible taux de couverture de fautes pouvant être atteint avec un nombre limité de vecteurs uniquement générés sur puce impose d'utiliser des ressources internes additionnelles dédiées au test (logique de pondération, bit fixing, bit flipping,…) et/ou une d'utiliser plus intensivement le testeur externe (broches de contrôle, bits de contrôle ou de configuration,…).

Concernant les techniques de compression verticale, les données de test sont compressées hors ligne, stockée sur le testeur et transmise à un décodeur intégré sur puce. L'idée est de réduire le nombre de donnée à transmettre sur un canal testeur en remplaçant des portions de données originale par des portions plus courtes. Pour cela, ces techniques tirent parti de la présence de bits non spécifiés (Xs) dans les séquences. Elles ne dépendent que des données test. Par contre ces méthodes engendrent une forte complexité au niveau du décodage et de la synchronisation, surtout dans le cas de circuits contenant plusieurs chaînes de scan.

Pour finir, les méthodes de compression horizontale visent à réduire le nombre de canaux testeur nécessaire à transmettre les données de test. Autrement dit, pour un nombre de canaux testeur donné, elles permettent d'augmenter le nombre de chaînes de scan interne et ainsi, comme il a été montré dans le Chapitre 1, de diminuer le temps de test. Ces méthodes exploitent aussi la possibilité d'assignation des Xs dans les séquences. Elles sont le plus souvent basées sur l'utilisation d'outils de test comme la simulation de fautes ou des générateurs de vecteurs. Les structures de décompressions peuvent prendre diverses formes. Certaines techniques ont même été commercialisées dans des suites d'outils d'aide à la conception de circuits intégrés.

2.1.4.1. Problèmes liés au test des IP

Actuellement, les circuits intégrés sont essentiellement des circuits séquentiels dont les bascules sont réorganisées en une ou plusieurs chaînes de scan. Cependant, il est de plus en plus fréquent que les industriels ne cherchent plus à concevoir intégralement leurs propres circuits, mais préfèrent acheter à d'autres fournisseurs des parties de circuits déjà prêtes à mettre en œuvre. Ces parties de circuits sont fournis sous la forme d'une boite noire – ou IP pour "Intellectual Property" – dont aucune information structurelle précise n'est fournie. On distinguera ici deux types d'IP: les "soft" IP qui sont fournies sous la forme d'une description fonctionnelle haut niveau et les "hard" IP dont la description, donnée par le layout niveau transistor, est figée pour une technologie visée. Le fournisseur d'IP ne communique que le format d'entrées-sorties du circuit IP ainsi que sa séquence de test.

On voit ici apparaître les contraintes liées au test de ce type de circuits. En effet, puisque aucune information structurelle n'est disponible (la liste des portes, les interconnections, bref le circuit en lui-même), toutes les techniques basée sur la simulation de fautes, l'exécution d'outils de génération de vecteurs, ou sur tout autre procédé nécessitant la connaissance de la structure du circuit, s'avèrent impossible à appliquer. Pour les mêmes raisons, on ne peut pas non plus modifier la structure du circuit, en particulier la structure des chaînes de scan. Le fournisseur a intégré une

structure de scan que l'intégrateur ne peut modifier, si ce n'est en reliant les entrées sorties de ces chaînes les unes aux autres (par exemple faire une seule chaîne de scan à partir de plusieurs en reliant les sorties des unes aux entrées des suivantes).

Ces techniques éliminent un très grand nombre des techniques de compressions de données de test, notamment les principales méthodes industrielles de Mentor®, SynTest® et Synopsys®.

2.1.4.2. Problèmes liés au test des SoCs

Le choix de l'utilisation d'IPs, plutôt que de circuits standards, est basé sur le principe de réutilisation ("design reuse"), qui contribue à augmenter la productivité et à diminuer les temps de conception ("time to market"). L'idée est de permettre aux intégrateurs de systèmes de choisir plusieurs blocs IP prédéfinis et de les assembler les uns aux autres pour former un système plus complexe. On trouve deux approches pour réaliser de tels systèmes : les systèmes sur carte ou les systèmes sur puces. La première consiste à regrouper plusieurs circuits intégrés, en boîtier, et à les interconnecter sur un circuit imprimé. Cette technique, bien que toujours utilisée (les cartes mère des micro-ordinateurs par exemple) se révèle très limitée (en performance et en volume physique du matériel) pour des systèmes très complexes. L'autre solution, celle qui nous intéresse dans ce mémoire, permet de regrouper sur le même morceau de silicium, plusieurs circuits préconçus, appelés cœurs dans la suite, afin de réaliser un système sur puce complexe (ou SoC pour "System-on-Chip") et très performant (les microprocesseurs actuels par exemple).

En ce qui concerne les tests de ces systèmes sur puces, contrairement aux systèmes sur carte, où chaque composant peut être testé indépendamment avant et après montage sur carte, les composants d'un système sur puce (cœurs et logique glue) sont difficilement testables puisqu'ils sont tous intégrés sur le même silicium. Plusieurs difficultés émergent de ce nouveau concept. Des problèmes apparaissent dès la conception de tels systèmes où les questions de testabilité doivent être prises en compte. Sur un système sur carte, chaque circuit est testé individuellement avant d'être connecté sur la carte, ensuite les interconnexions sont testées. Sur un système sur puce, il n'y a pas de test individuel avant l'assemblage. C'est à l'intégrateur de s'occuper du test des cœurs une fois ceux-ci intégrés sur la puce, et non au fournisseur de cœur. La conception du test devient alors complètement différente. Le test physique n'intervenant qu'après fabrication, l'intégrateur du système doit prévoir dès la conception des moyens d'accès qui permettront de tester chacun des composants intégrés sur la puce.

D'autre part, comme pour les circuits intégrés, la réduction du coût du test des SoCs est aussi devenue une préoccupation majeure chez les intégrateurs systèmes. Comme on l'a vu, la réduction de ces coûts passe entre autre par la réduction du temps de test des systèmes. Ce temps est directement lié au nombre de vecteurs de test à appliquer à chaque cœur et aux possibilités de parallélisme dans le système. Si les cœurs sont testés les uns après les autres, le temps de test du système est la somme des temps de test de tous les cœurs. Il peut rapidement devenir prohibitif dans le cas de systèmes comportant de nombreux cœurs. En mettant en parallèle les tests de plusieurs cœurs, le temps de test global peut être réduit. Cependant, les possibilités de parallélisation sont

limitées par plusieurs facteurs dont principalement le nombre de canaux testeurs requis pour tester un seul cœur. Un système ayant un nombre de broches d'entrées-sorties fonctionnelles fixé, et le nombre de canaux de test disponibles sur un testeur étant limité par leur coût, il est devenu très intéressant d'utiliser des techniques de compression horizontale afin de réduire le nombre de canaux testeur requis pour tester un cœur. Ainsi, la réduction de nombre de canaux testeur par cœur implique que plus de cœurs peuvent être testés en même temps. Le parallélisme est augmenté, et le temps de test global du système peut être réduit.

Dans ce cadre, la technique de compression à utiliser doit répondre à un certain nombre de contraintes. En particulier, le(s) décompresseur(s) intégré(s) au sein du système doivent pouvoir décompresser les données pour un ou plusieurs cœurs. Cette contrainte impose aux décompresseurs de ne pas être conçu spécifiquement pour un cœur en particulier. La technique de compression doit être générique et permettre aux décompresseurs de décompresser les données de n'importe quel cœur. En d'autres termes, la technique de compression doit être indépendante des circuits à tester et doit permettre le partage de décompresseur par plusieurs cœurs. Il n'existe pas à l'heure actuelle de telles méthodes répondant aux contraintes liées à la fois au test des IPs et à ceux des SoCs. Nous proposons donc dans la section suivante une nouvelle méthode de compression des données de test admettant ces contraintes.

2.2. Nouvelle méthode de compression des données de test.

2.2.1. Objectifs:

Dans le cadre de cette thèse, nous proposons une nouvelle technique de compression des données de test pour tenter de répondre aux critères énoncés précédemment. L'objectif principal est de concevoir une technique de compression adaptable au test des systèmes. Pour cela, la technique de compression et le décompresseur sur puce doivent répondre au cahier des charges suivant:

- Indépendance vis-à-vis des données de test : la technique de compression ne doit procéder à aucune analyse des données. Quelles que soient ces données, elle doit toujours pouvoir générer le vecteur décompressé désiré.
- Indépendance vis à vis de la netlist du circuit : dans le cadre du test de circuit IP, aucune connaissance structurelle du circuit sous test n'est disponible. La technique de compression ne doit pas reposer sur la connaissance de la netlist.
- Ne requiert pas d'outils de génération de vecteurs de test spécifique : du fait de l'absence de connaissance structurelle des circuits, le test des IP impose de ne pas utiliser d'outils de test spécifiques tels que des générateurs de vecteurs automatique ou de simulateurs de fautes.
- Décompresseur sur puce faible coût : comme toute les techniques de compression des données de test, notre technique devra être simple à intégrer et occuper le moins de place possible sur le silicium.

- Conservation du taux de couverture : bien évidemment, la séquence compressée générée par la technique de compression ne devra en aucun cas diminuer le nombre de fautes couvertes par la séquence originale.

Dans la suite, nous proposons un algorithme de compression des données de test associée à une architecture de décompression sur puce répondant à tous ces critères. Dans un premier temps, l'architecture du décompresseur sera détaillée. Nous donnerons ensuite les détails de l'algorithme de compression proprement dit. Pour finir, une analyse formelle de cette technique sera proposée.

2.2.2. Architecture proposée

2.2.2.1. Idée Initiale

Le principe général d'une architecture de compression-décompression horizontale pour un circuit avec N chaînes de scan est d'envoyer les slices de N bits au travers de M canaux testeur (avec M<N, i.e. moins de canaux testeur disponibles que de chaînes de scan) au décompresseur sur puce qui va régénérer exactement les slices originales. Les vecteurs de test préalablement compressés hors ligne sont stockés dans la mémoire du testeur.

Nous proposons ici une amélioration de la méthode présentée dans la thèse de Régis Poirier [Poir04]. L'idée principale de cette méthode est d'utiliser un décompresseur séquentiel constitué d'un additionneur et d'un registre à décalage (Figure 2.32).

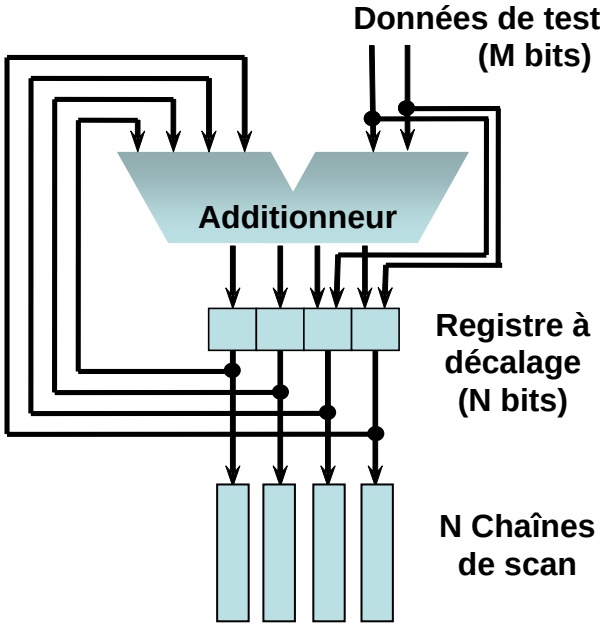


Figure 2.32 : Structure du décompresseur pour 4 chaînes de scan et 2 entrées testeur

Le principe général est d'envoyer la différence arithmétique D_i (codée sur M bits) entre deux slices S_i et S_{i+1} (codées sur N bits), plutôt que les slices de N bits elles-mêmes. L'exemple suivant permet d'illustrer le gain obtenu en termes de canaux testeurs. Soit le vecteur de 5 slices S_1à S_5 codés sur 5 bits donnée sur la Figure 2.33. Les différences entre slices successives, $D_1=S_2-S_1$, D_2,

D_3 et D_4 peuvent toutes être codées sur 3 bits. En stockant dans la mémoire testeur les différences à la place des slices initiales, le nombre de canaux testeur nécessaires à la transmission de ces données vers le circuit sous test est réduit de 5 à 3.

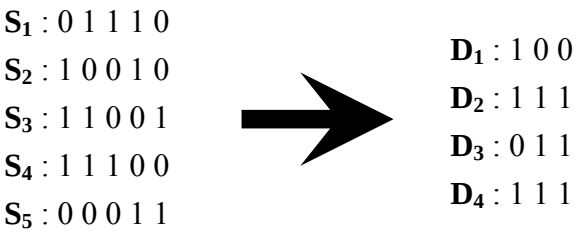


Figure 2.33 : Compression par codage des différences

De manière plus formelle, notons S_i une slice de la séquence de test et S_{i+1} la slice suivante. La slice compressée (la différence arithmétique) est notée Sci. Pour N chaînes de scan nous avons:

$S_i = a_{N-1}\ldots\ldots\ldots\ldots a_0$

$S_{i+1} = b_{N-1}\ldots\ldots\ldots\ldots b_0$

Pour M canaux testeur:

$Sc_i = c_{M-1}\ldots..c_0$ (avec les c_i positionnés sur les poids faibles).

Le décompresseur réalise donc l'opération en un seul cycle testeur:

$$\begin{array}{lll} & a_{N-1}\ldots\ldots\ldots\ldots a_0 & (S_i) \\ + & 0\ldots\ldots\ldots 0c_{M-1}c_{M-2}c_1c_0 & (Sc_i) \\ \hline & \mathbf{b}_{N-1}\ldots\ldots\ldots\ldots \mathbf{b}_0 & (S_{i+1}) \end{array}$$

Cependant, il est possible que cette différence arithmétique ne puisse pas être codée sur M bits. Dans ce cas, les slices sont sérialisées dans le registre à décalage. La transmission de chaque slice aux chaînes de scan requiert alors $\lceil N/M \rceil$ cycles testeur au lieu d'un seul cycle dans le cas ou la différence existe. Or lorsque N est bien supérieur à M, l'augmentation du temps de test d'une slice devient trop pénalisante. De plus, la probabilité que cette situation se produise dépend de l'écart e = N-M entre le nombre de chaînes de scan N et le nombre de canaux testeur M. Cette probabilité s'exprime ainsi: $p = 1-1/2^e$.

Nous avons donc une structure qui, dans le cas ou N >>M (i.e l'écart e est grand), a une forte probabilité de ne pas pouvoir fournir de slice compressée. L'augmentation du temps de test lors de ces situations pénalise trop fortement les performances de ce procédé.

2.2.2.2. Amélioration

Nous proposons donc ici une amélioration de cette structure. Les bits de la différence arithmétique ne sont plus positionnés uniquement sur les poids faibles mais répartis de manière

homogène sur la slice originale. Toujours sur l'exemple précédent, avec deux slices successives S_i et S_{i+1}, pour N chaînes de scan nous avons:

$S_i = a_{N-1}\ldots\ldots\ldots\ldots a_0$

$S_{i+1} = b_{N-1}\ldots\ldots\ldots\ldots b_0$

Pour M canaux testeur:

$Sc_i = c_{M-1}\ldots\ldots c_0$ (avec les c_i répartis sur la slice).

Le décompresseur réalise donc l'opération en un seul cycle testeur:

$$\begin{array}{lll} & a_{N-1}\cdots\cdots\cdots a_0 & (S_i) \\ + & 0\cdots 0c_{M-1}0\cdots 0c_{M-2}0\cdots\cdots 0c_1 0\cdots 0c_0 & (Sc_i) \\ & \hline & \\ & \mathbf{b_{N-1}}\cdots\cdots\cdots\mathbf{b_0} & (S_{i+1}) \end{array}$$

Le choix de répartir les bits c_i de manière homogène sur les entrées du décompresseur (donc sur les entrées de l'additionneur) repose sur la propriété suivante: la probabilité d'obtenir un bit b_i de S_{i+1} à partir d'un bit a_i de S_i est proportionnel à $1/2^d$, où d est la distance entre deux bits c_i et c_{i+1}, ce qui équivaut à d = N/M. Or, avec la structure présentée dans [Poir04], la probabilité est de $1/2^e$ où e = N-M. Notre structure est plus efficace (meilleure probabilité) si e > d, soit N-M > N/M ce qui correspond à la situation de N>>M. Une répartition homogène des bits c_i permet donc d'augmenter la probabilité de trouver une slice compressée, et donc de réduire le nombre de slices à charger en série.

2.2.2.3. Décompresseur sur puce

Avec une répartition homogène des bits de données, la structure du décompresseur est légèrement modifiée: lors de l'opération d'addition, le bit de propagation de retenue n'est pas transmis des lors qu'une entrée de donnée est présente sur l'additionneur. La structure de l'additionneur est donc ramenée à plusieurs blocs additionneurs concaténés les uns aux autres:

$$\left|\begin{array}{l} a_{N-1}\cdots a_{(M-1)*\lceil N/M\rceil} \\ + \ 0\cdots\cdots 0c_{M-1} \\ \hline b_{N-1}\cdots b_{(M-1)*\lceil N/M\rceil} \end{array}\right| \left|\begin{array}{l} a_{(M-1)*\lceil N/M\rceil-1}\cdots a_{(M-2)*\lceil N/M\rceil} \\ + \ 0\cdots\cdots\cdots\cdots 0c_{M-2} \\ \hline b_{(M-1)*\lceil N/M\rceil-1}\cdots b_{(M-2)*\lceil N/M\rceil} \end{array}\right| \cdots\cdots \left|\begin{array}{l} a_{2*\lceil N/M\rceil-1}\cdots a_{\lceil N/M\rceil} \\ + \ 0\cdots\cdots\cdots 0c_1 \\ \hline b_{2*\lceil N/M\rceil-1}\cdots b_{\lceil N/M\rceil} \end{array}\right| \left|\begin{array}{l} a_{\lceil N/M\rceil-1}\cdots a_0 \\ + \ 0\cdots\cdots\cdots 0c_0 \\ \hline b_{\lceil N/M\rceil-1}\cdots b_0 \end{array}\right| \begin{array}{l} (S_i) \\ (Sc_i) \\ \\ (S_{i+1}) \end{array}$$

Le décompresseur n'est donc pas un additionneur complet mais est composé de plusieurs additionneurs indépendants les uns des autres. Chaque bloc reçoit en poids faible la donnée de test qui lui est attachée. Il y a donc autant de blocs d'addition que d'entrées de test (M blocs pour M entrées) comme illustré sur la Figure 2.34. La longueur de chaque bloc d'addition est donc égale à N/M (certains ayant une longueur de N/M+1 dans le cas ou le rapport N/M n'est pas entier). C'est cette valeur qui fixe la distance d entre 2 entrées de test dans la structure du décompresseur.

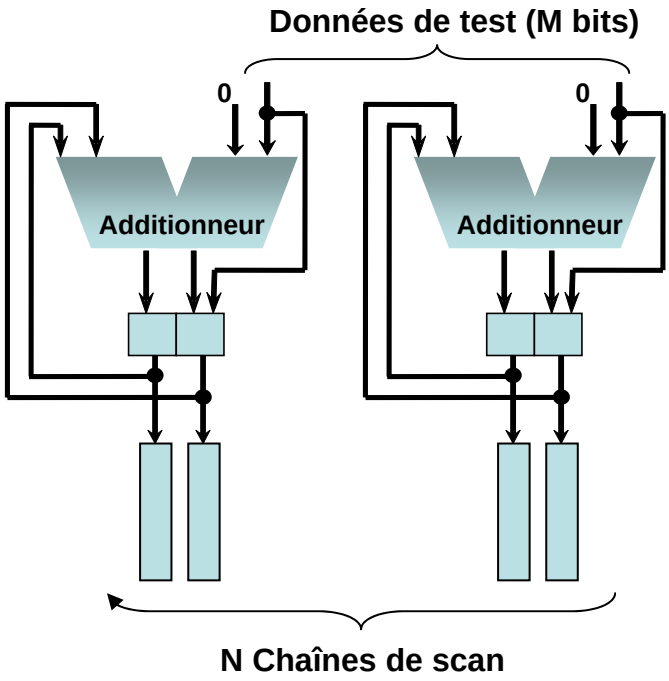


Figure 2.34 : Nouvelle structure du décompresseur sur puce pour 4 chaînes de scan et 2 entrées testeur

L'autre avantage majeur inhérent à notre structure par rapport à la structure précédente, réside dans la réduction du temps de propagation du résultat de l'addition. En effet, la structure précédente utilise un additionneur N bits, alors que notre structure est composée de plusieurs additionneurs N/M+1 bits. Notre structure permet donc d'avoir le résultat de l'addition plus rapidement et permet donc au décompresseur de fonctionner à une cadence plus élevée. Bien que le temps de propagation dans un additionneur puisse être réduit avec des structures d'additionneur plus complexe, l'augmentation de la surface de silicium impliquée par ces structures plus complexes ne correspond pas aux objectifs de décompresseur faible coût.

2.2.2.3.1. Synchronisation et commande

Il existe trois différents modes de fonctionnement de notre architecture de test: le mode normal (lors de l'application d'un vecteur, les données de test présentes dans les bascules sont appliquées à la logique combinatoire, puis sur le front d'horloge, les réponses sont récupérées dans les bascules), le mode test (chargement des chaînes de scan) dans le cas où la slice est compressée, et le mode test dans le cas où la slice n'est pas compressée. La sélection du mode de fonctionnement du décompresseur est effectuée à l'aide de deux bits additionnels. Le bit de "Scan Enable" permet la sélection entre mode normal et mode test. Ce bit est présent dans toutes les techniques de test externe scan. En mode test, le bit nommé "Add_Ctr" sur la Figure 2.35 permet quant à lui, de basculer le décompresseur du mode additionneur (slice compressée) au mode décalage (slice non compressée) selon la slice à transmettre.

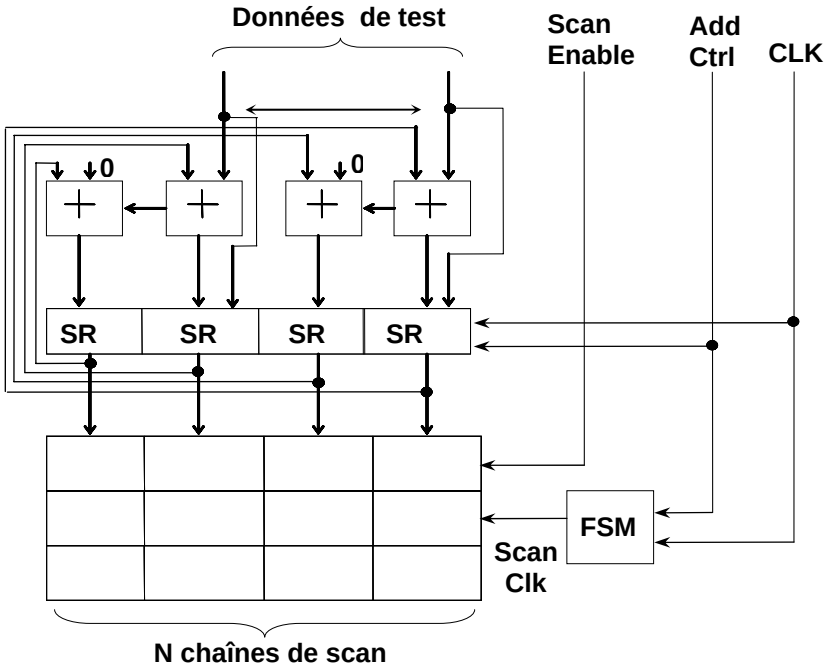


Figure 2.35 : Architecture détaillée de notre décompresseur pour 4 chaînes de scan et 2 entrées de données de test

Le rôle de la machine d'états finis ("FSM" sur la Figure 2.35) est d'inhiber le décalage dans les chaînes de scan lors du chargement série des slices non compressées. En effet, la slice compressée nécessite $\lceil N/M \rceil$ cycles pour être chargée dans le registre à décalage. Or pendant toute la durée de ce chargement, les chaînes de scan ne doivent pas être décalées puisque la slice complète n'est pas encore disponible dans le registre. En conséquence, en fonctionnement décalage en mode test, la machine d'états inhibe l'horloge pour les chaînes de scan mais pas pour le registre à décalage.

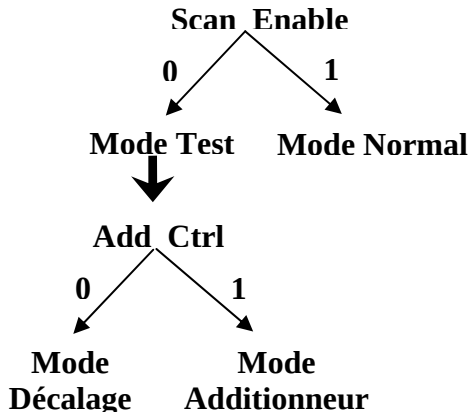
En résumé, trois modes de fonctionnement sont nécessaires au décompresseur (voir Figure 2.36) et sont commandés par 2 signaux externes (Scan_Enable et Add-Ctrl) à l'aide d'une machine d'états:

(1) Mode Normal (application et capture)

- Mode Test

(2) Mode Additionneur (slice compressée)

(3) Mode Décalage (slice non compressée)



Scan_Enable	Add_Ctrl	Mode	
0	0	Test	Décalage
0	1	Test	Additionneur
1	X	Normal	

Figure 2.36 : Modes de fonctionnements du décompresseur et signaux de contrôle

2.2.2.3.2. Fonctionnement

L'exemple suivant illustre le fonctionnement de la structure pour une configuration de 4 chaînes de scan et 2 canaux testeur. Un vecteur contenant 4 slices S1 à S4 de quatre bits doit être envoyé aux chaînes de scan. La première étape consiste à initialiser le décompresseur. Le registre à décalage du décompresseur ainsi que les chaînes de scan contiennent des valeurs indéterminées (Figure 2.37), il est donc impossible de commencer directement par une opération d'addition. La première slice S_1 est donc transférée en mode série via le registre à décalage du décompresseur en 2 cycles d'horloge (Figure 2.38 et Figure 2.39). Un dernier cycle est nécessaire au chargement de cette slice dans les chaînes de scan. (Figure 2.40).

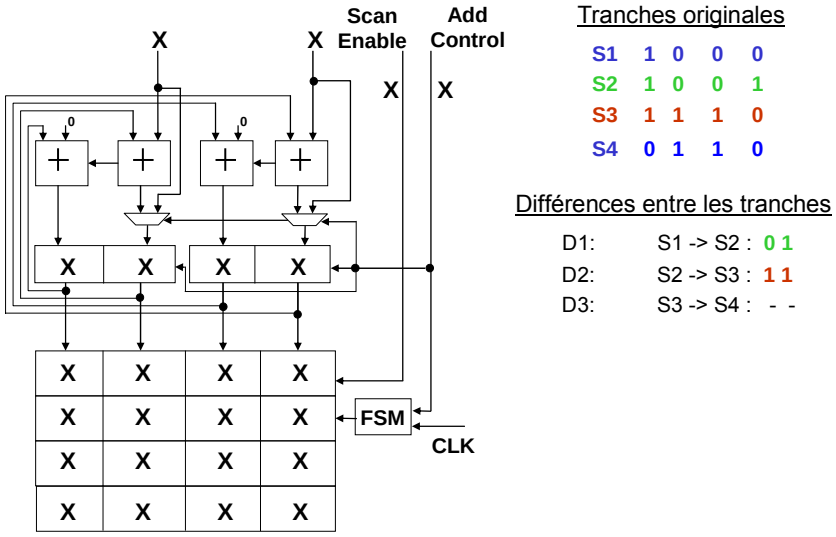


Figure 2.37 : Etat initial du décompresseur et des chaînes de scan

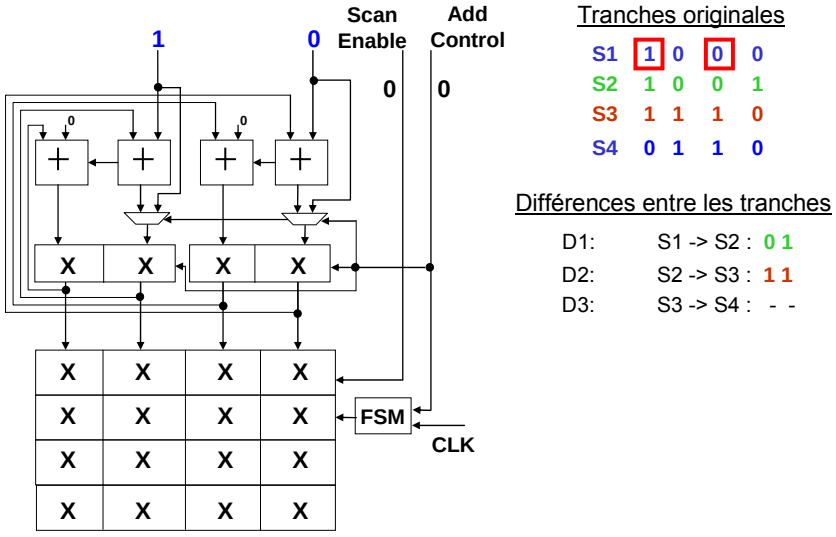


Figure 2.38 : Initialisation de la première slice S_1 (cycle 1)

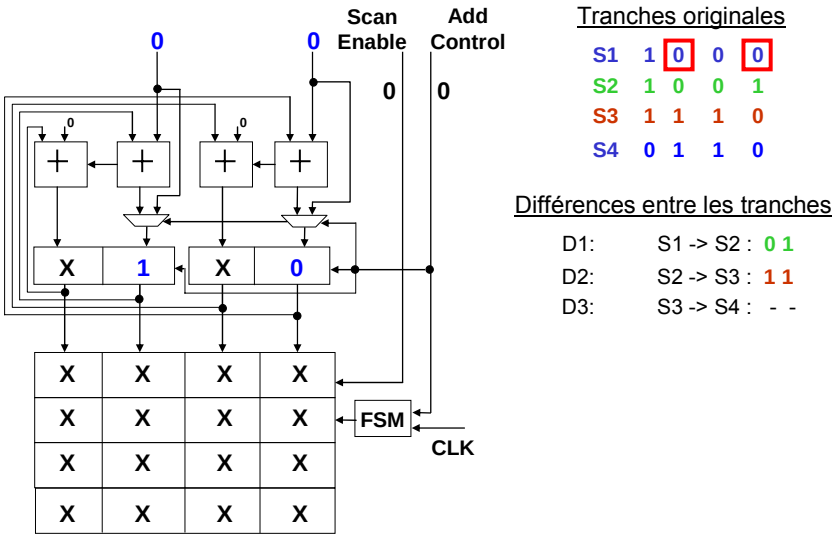


Figure 2.39 : Initialisation de la première slice S_1 (cycle 2)

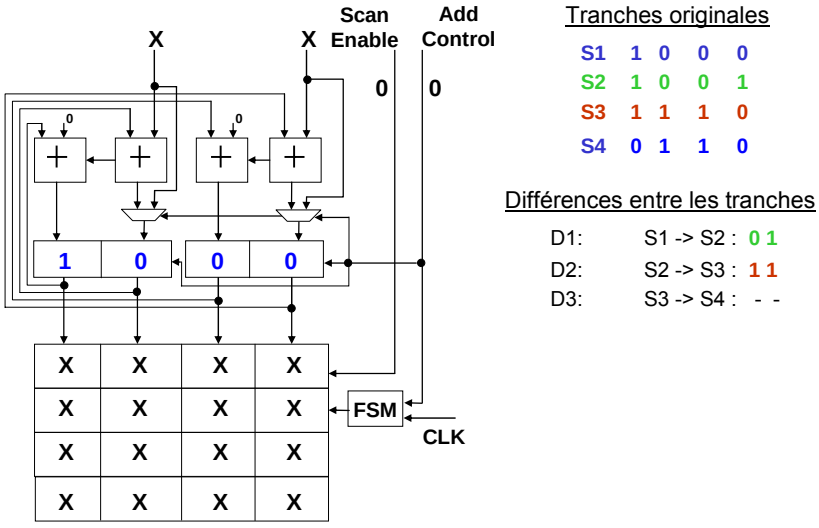


Figure 2.40 : Initialisation de la première slice S_1 (cycle 3)

Une fois le registre initialisé, la slice S_2 la slice S_2 est chargée dans le registre en envoyant le vecteur différence D_2 entre les slices S_1 et S_2 par le mode additionneur en 1 seul cycle (Figure 2.41). La différence D_2 est additionnée à S_1 pour recréer S_2. On répète cette opération d'addition pour la slice S_3 en additionnant la différence D_3 à la slice S_2 qui est à son tour chargée dans les chaînes de scan (Figure 2.42).

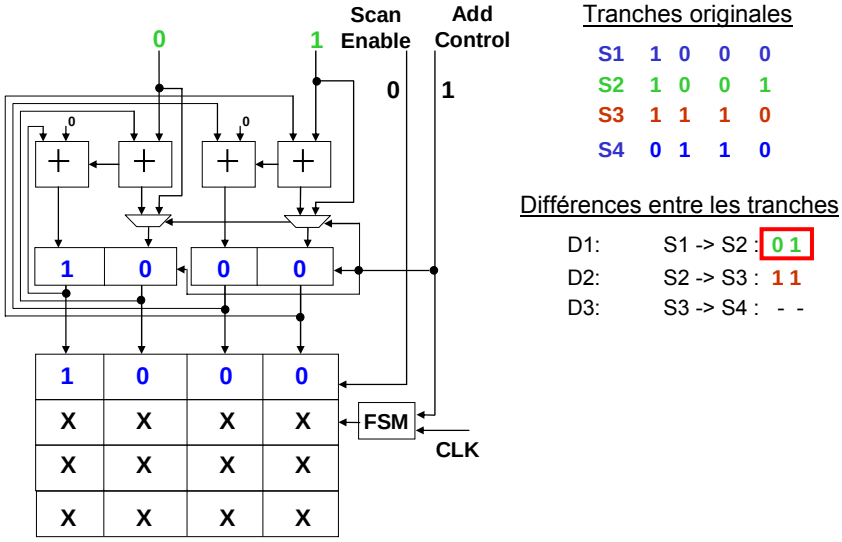


Figure 2.41 : Envoi de la seconde slice S_2 (Cycle 4)

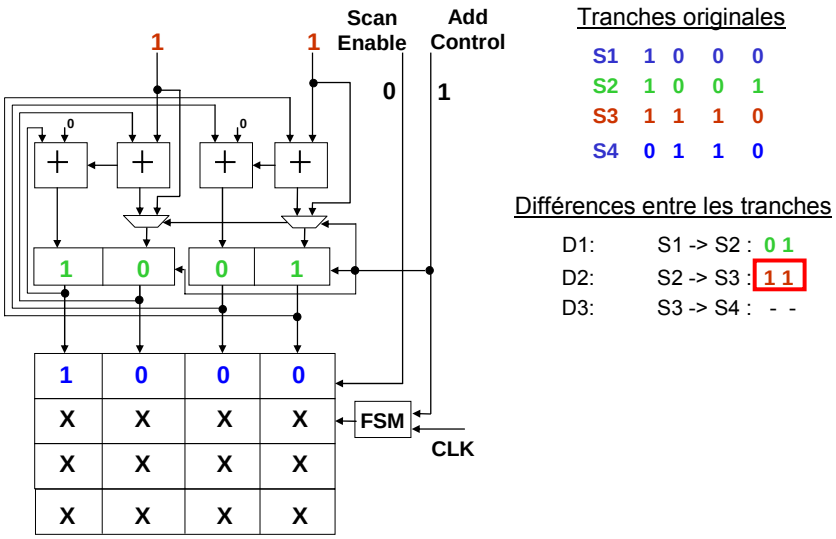


Figure 2.42 : Envoi de la troisième slice S_3 (Cycle 5)

En ce qui concerne la slice S_4, il n'existe pas de vecteur différence sur 2 bits permettant de passer de S_3 à S_4. Dans ce cas, la slice S_4 doit être envoyée par le mode série aux chaînes de scan en 2 cycles, de la même manière que lors de l'initialisation (Figure 2.43 et Figure 2.44). Là encore, un dernier cycle est nécessaire au chargement de la slice dans les chaînes de scan (Figure 2.45).

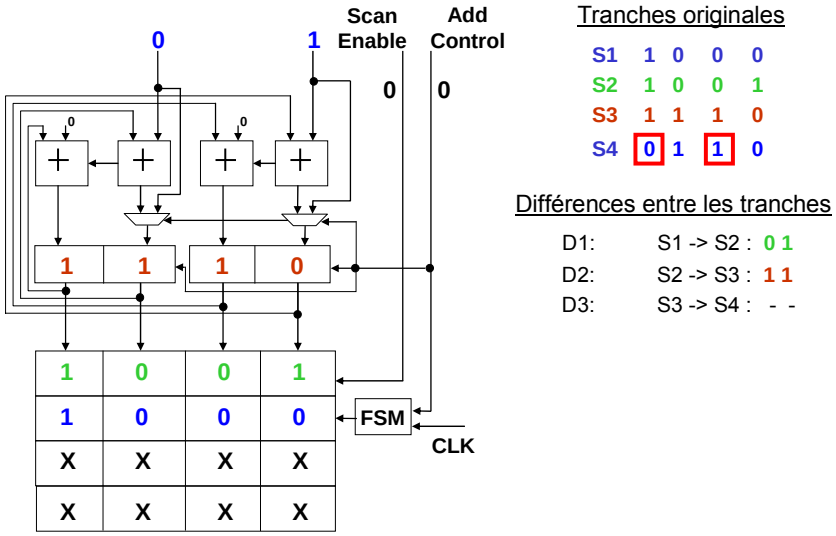


Figure 2.43 : Chargement de la slice S_4 (Cycle 6)

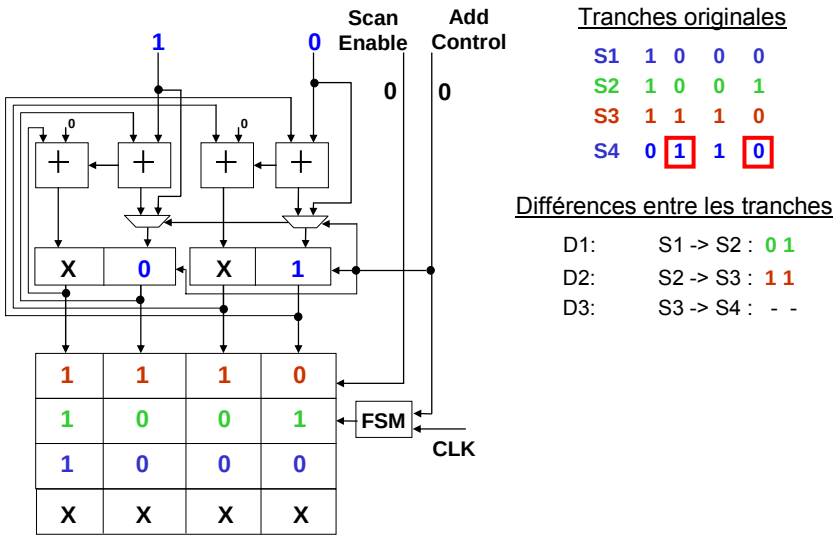


Figure 2.44 : Chargement de la slice S_4 (Cycle 7)

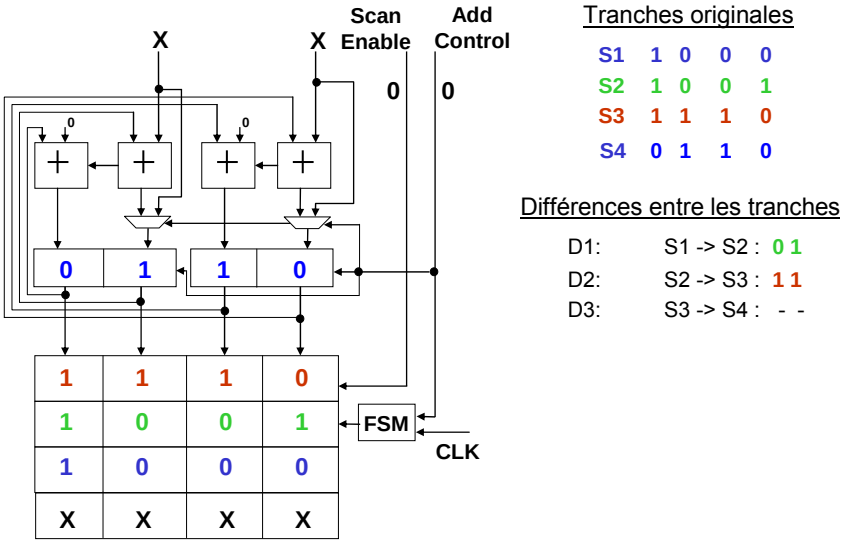


Figure 2.45 : Chargement de la slice S_4 (Cycle 8)

Une fois toutes les slices chargées dans les chaînes de scan (autrement dit, le vecteur est complètement chargé), le mode de fonctionnement normal est activé (Figure 2.46). Les chaînes de scan reçoivent alors les réponses du circuit (Figure 2.47). Ces réponses seront récupérées au fur et à mesure que les slices du nouveau vecteur seront envoyées (Figure 2.48).

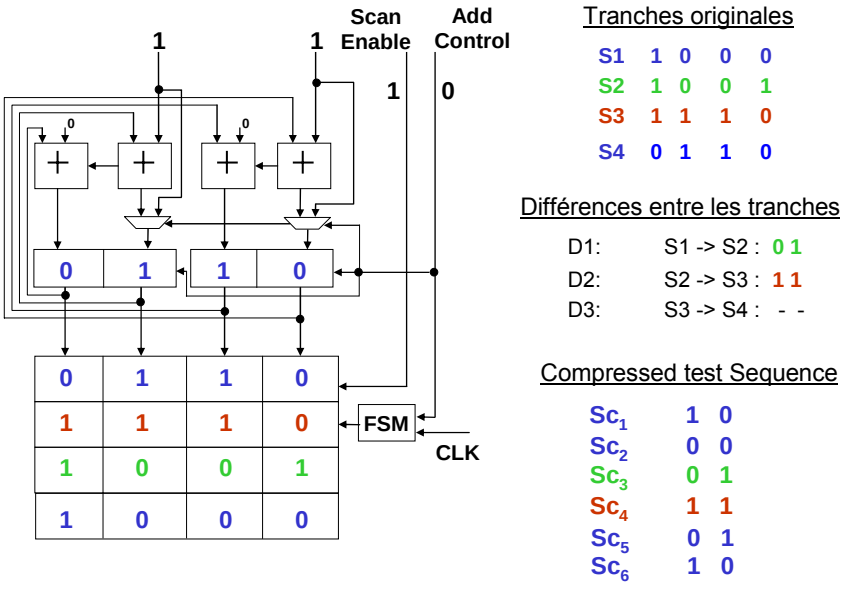


Figure 2.46 : Activation du circuit et capture des réponses (Cycle 9)

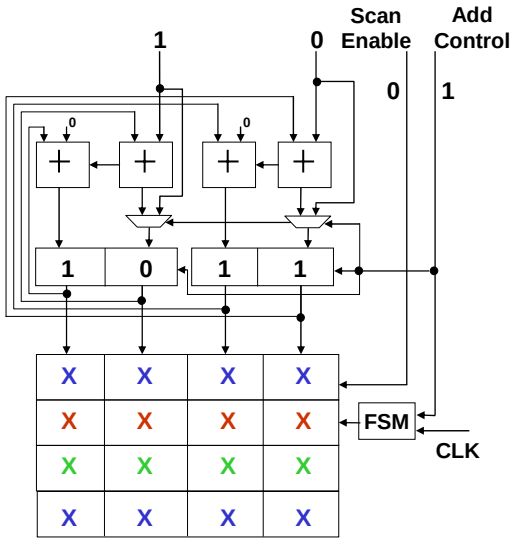


Figure 2.47 : Récupération des réponses dans les chaînes de scan

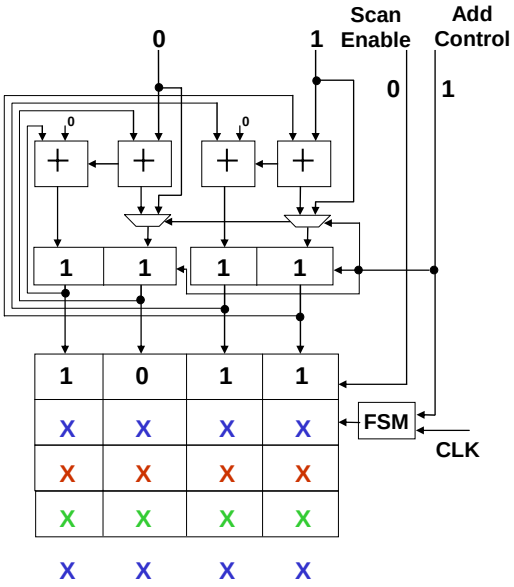


Figure 2.48 : Expulsion des réponses et chargement des nouvelles slices

2.2.3. Algorithme de compression

Dans l'exemple de la Figure 2.49, la séquence de test T est compressée en T' qui contient les slices compressées notées Sci (N=8 et M=2). T* représente la séquence de test qui sera concrètement appliquée au circuit après décompression. La première slice de T' n'est pas compressée mais chargée en série dans le registre en $\lceil N/M \rceil + 1$ cycles comme décrit précédemment.

T:	S_1:	x x 0 0	x 0 1 1	**T'**:	Sc_1:	1 1 0 0	1 0 1 1
	S_2:	x 1 0 x	1 x x 0		Sc_2:	1	1
	S_3:	1 x x 0	x x x 1		Sc_3:	1	1
	S_4:	x x 1 1	x 1 x x		Sc_4:	1	0
	S_5:	1 1 x x	x x 1 0		Sc_5:	0	1
	S_6:	x 0 x 0	1 x 1 x		Sc_6:	1	0

T*	S_1:	1 1 0 0	1 0 1 1
	S_2:	1 1 0 1	1 1 0 0
	S_3:	1 1 1 0	1 1 0 1
	S_4:	1 1 1 1	1 1 0 1
	S_5:	1 1 1 1	1 1 1 0
	S_6:	0 0 0 0	1 1 1 0

Figure 2.49 : Exemple de séquence de test compressée

Selon les valeurs de S_i et S_{i+1}, il est possible qu'aucune slice compressée Sc_i n'existe. Dans ce cas, le registre est initialisé avec la slice S_{i+1} en mode chargement série. La slice S_{i+1} est stockée dans le testeur en tant que $\lceil N/M \rceil$ mots de M bits qui sont envoyés en série, sans passer par l'additionneur du décompresseur. A chaque fois que cette situation ce produit, le temps d'application de la séquence de test est pénalisé de $\lceil N/M \rceil$ cycles.

Lors de la compression des données de test, le nombre d'occurrences de ces slices non compressibles doit donc être réduit en utilisant au mieux les degrés de liberté fournis par la présence de X dans la séquence de test originale. En effet, comme pour toutes les méthodes de compression, les séquences doivent contenir des X pour pouvoir être compressées. Cependant, lorsqu'elles sont appliquées au circuit, ces séquences sont complètement spécifiées. Les X doivent donc être assignés lors de la compression, et ce de façon à réduire le nombre de slices non compressibles. Par conséquent, le temps d'application du test et le volume des données compressées dépendent de cette assignation des X.

Cette assignation intervient lors de deux étapes de l'algorithme. Premièrement lors de la phase d'initialisation, il faut définir tous les bits de la première slice de la séquence ou, à chaque fois qu'il n'est pas possible de trouver une slice incompressible, il faut réinitialiser le processus de compression avec la slice suivante en la définissant totalement. Puis, lors de la phase de recherche des slices compressées, le choix qui est effectué sur chaque tranche compressée spécifie ainsi les valeurs qui sont concrètement transmises au circuit.

2.2.3.1. Assignation initiale

La Figure 2.50 illustre l'assignation lors de la phase d'initialisation d'une slice. Cette assignation est faite par recherche verticale dans la séquence de test jusqu'à la première valeur spécifiée. Ce choix est fait pour faciliter la compression puisque plus la distance de Hamming entre une slice et sa suivante est petite, plus il est facile de trouver une slice compressée permettant de passer de l'une à l'autre. Les bits b_7, b_6 et b_3 de la première slice S_1 sont donc respectivement initialisés avec les mêmes bits des slice S_3, S_2 et S_2. On obtient donc S_1 = 11001011 qui devient la slice d'initialisation Sc_1 des données compressées.

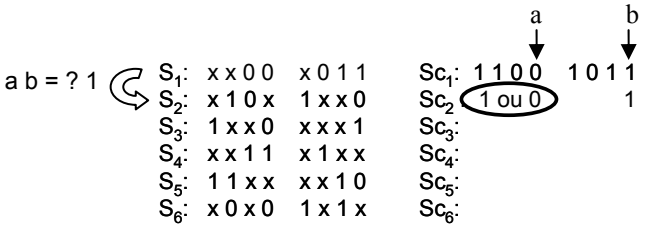
		b_7 b_6 b_5 b_4	b_3 b_2 b_1 b_0			b_7 b_6 b_5 b_4	b_3 b_2 b_1 b_0
T:	S_1:	x x 0 0	x 0 1 1	**T'**	Sc_1:	1 1 0 0	1 0 1 1
	S_2:	x 1 0 x	1 x x 0		Sc_2:	1	1
	S_3:	1 x x 0	x x x 1		Sc_3:	1	1
	S_4:	x x 1 1	x 1 x x		Sc_4:	1	0
	S_5:	1 1 x x	x x 1 0		Sc_5:	0	1
	S_6:	x 0 x 0	1 x 1 x		Sc_6:	1	0

Figure 2.50 : Assignation des X lors de l'initialisation d'une slice

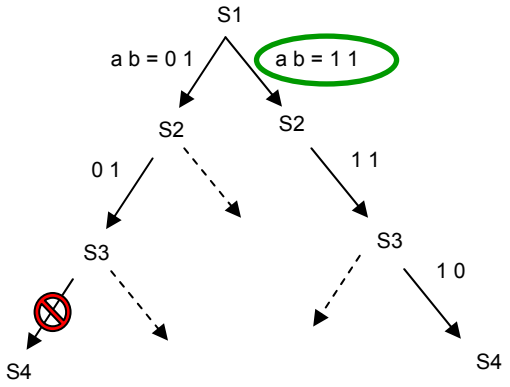
2.2.3.2. Assignations intermédiaires

Ensuite, il existe plusieurs possibilités pour passer de la slice S_1 à la slice S_2 par addition. Par exemple, il est possible d'ajouter soit a = 0 soit a = 1 à la première partie de la slice S_1 pour obtenir la partie correspondante de la slice S_2 grâce à la présence de X dans S_2, comme indiqué sur la Figure 2.51 (a). Pour effectuer ce choix, un arbre est parcouru afin de trouver la profondeur atteinte par chaque choix. La solution mémorisée est celle atteignant la plus grande profondeur sur l'arbre. Sur la Figure 2.51 (b), on peut remarquer que le choix a = 0 atteint sa profondeur maximale à la slice S_3 alors que le choix a = 1 permet de n'avoir aucune slice incompressible comme illustré sur la

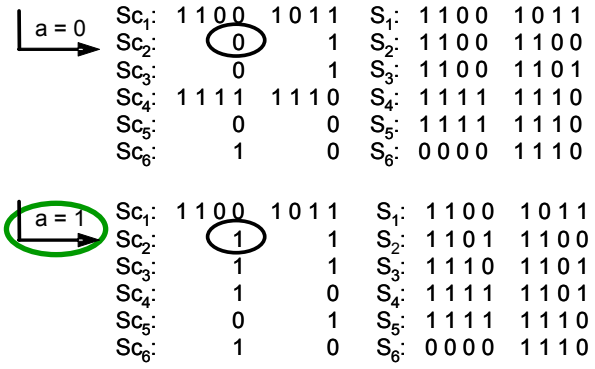
Figure 2.51 (c), et donc aucun chargement série supplémentaire (hormis celui de la slice initiale). La slice compressée Sc_2 est donc déterminé à Sc_2 = 11, ce qui assigne S_2 à 11001100.



(a) deux choix sont possibles pour le choix du bit a



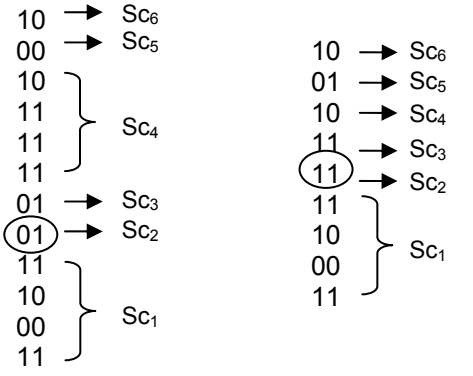
(b) avec a = 0, il n'existe pas de slice compressée permettant de passer de S_3 à S_4



(c) avec a = 1, toutes les slices sont compressibles

Figure 2.51 : Assignation des X lors du choix de la slice compressée

A la fin du processus de compression, toutes les valeurs indéterminées (X) ont été assignées lors des phases d'assignation de la compression dans le but d'améliorer la compressibilité des slices originales. La séquence compressée ainsi obtenue contient à la fois les slices compressée Sc_i, ainsi que les slices n'ayant pu être compressées sous la forme de $\lceil N/M \rceil$ slices de M bits. La Figure 2.52 illustre l'importance de l'assignation des X sur la séquence compressée obtenue. Un mauvais choix pénalise le volume des données à stocker (ici 12*2 = 24 bits dans le cas a contre 9*2 = 18 bits pour le cas b) ainsi que le temps de test (11 cycles pour les cas a contre 9 cycles pour le cas b).



(a) choix a = 0 (b) choix a = 1

Figure 2.52 : Impact de l'assignation des X sur la séquence compressée

2.2.3.3. Attribution des canaux testeur

Au préalable à l'exécution de l'algorithme, il faut définir auxquelles des N chaînes de scan vont être connectée les M canaux testeurs (dans le cadre de la compression horizontale, M < N). Ces choix sont faits avec pour objectif de réduire le nombre de slices non compressibles dans la séquence de test. Une fois la séquence mise sous cette forme de succession de slices de N bits, la plus grande cause d'occurrence de slices non compressibles est la succession de bits spécifiés dans les colonnes de cette séquence. Pour réduire ce phénomène, les chaînes de scan contenant le plus grand nombre de bits spécifiés successifs sont positionnées directement sur les broches d'entrée ou placées au plus proche comme décrit sur la Figure 2.53. Sur cet exemple, les chaînes C_2 et C_3 contiennent 2 bits spécifiés, la chaîne C_1 contient 1 bit spécifié et la chaîne C_0 n'en contient aucun. Les canaux testeurs sont donc reliés aux chaînes C_2 et C_3 et les autres chaînes sont placées au plus prés des canaux d'entrée dans l'ordre du nombre de bits spécifiés qu'elles contiennent décroissant. Il est désormais plus facile d'intervenir sur ces successions de bits spécifiés puisque la probabilité d'obtenir un bit désiré sur les canaux d'entrée est de 100% et décroît selon $1/2^d$, où d est la distance par rapport au canal d'entrée considéré.

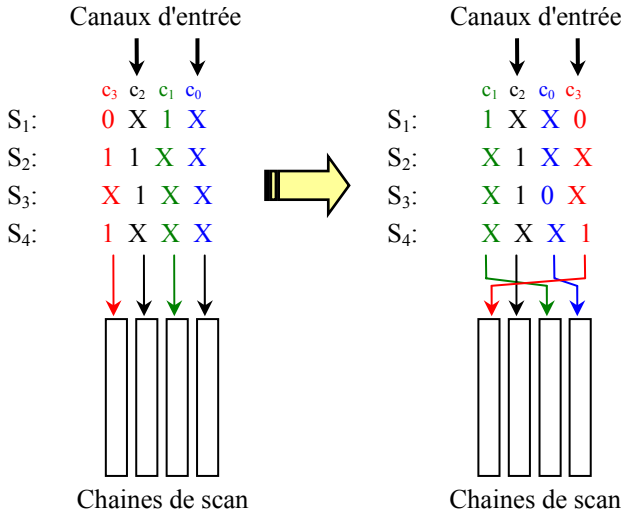


Figure 2.53 : Ordonnancement des colonnes

En pratique, il ne s'agit aucunement d'une modification de la séquence de test puisque le rétablissement de chaque colonne dans sa chaîne de scan correspondante est assuré par un simple câblage entre la cellule du registre à décalage du décompresseur et la chaîne de scan idoine.

2.2.3.4. Compression

Une fois la séquence réorganisée, l'algorithme de compression décrit Figure 2.54 est appliqué. La première étape consiste à assigner les X de la première slice pour l'initialiser comme décrit ci-dessus (Figure 2.50). Ensuite, pour chaque slice S_i de la séquence, l'algorithme de la Figure 2.54 recherche un si il existe une slice compressée Sc_i telle que $S_{i+1} = S_i + Sc_i$. Si différentes possibilités

existent, les valeurs indéterminées (X) sont assignées de manière à favoriser la compression des slices suivantes (Figure 2.51). La slice S_{i+1} complètement spécifiée est ensuite obtenue par addition.

Dans le cas où il n'existe pas de slice compressée Sc_i, la slice S_{i+1}est obtenue par la phase d'initialisation. Le processus de compression reprend à partir de cette slice et recherche s'il existe une slice compressée permettant de passer à la suivante.

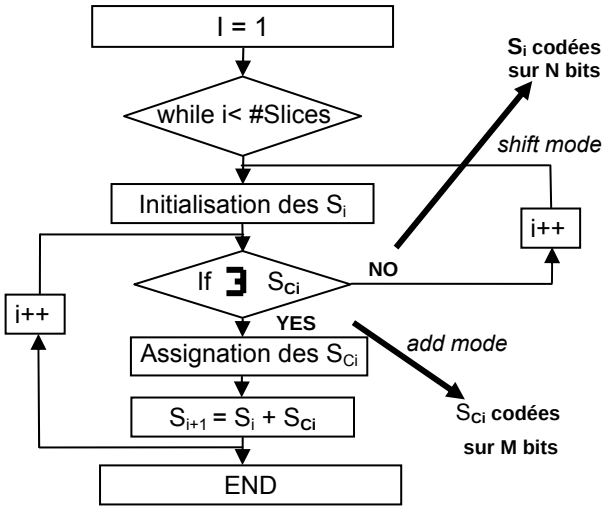


Figure 2.54 : Algorithme de compression

2.2.4. Performance

Dans cette section, les expressions théoriques du temps de test et du volume des données de test compressées sont décrites. Elles représentent le fonctionnement de la structure choisie dans ces travaux. Ces équations seront utilisées dans la section 2.3 pour comparer les résultats obtenus à l'aide de notre méthode de compression, avec ceux d'une méthode standard de sérialisation des données de test.

2.2.4.1. Equations temps de test

L'équation 2.1 donne l'expression du temps de test pour une séquence compressée avec notre méthode. N_{SLC} est le nombre de slices compressées, $N_{SL_{NC}}$ le nombre de slices non compressées, et P_2 le nombre de vecteurs dans la séquence originale. Notons que chaque slice compressée est chargée en un cycle dans les chaînes de scan. Une slice non compressée, elle, a besoin de $\lceil N/M \rceil+1$ cycles pour être chargée puisqu'elle est stockée en tant que $\lceil N/M \rceil$ mots dans la séquence compressée. Elle est ensuite envoyée en série au circuit. L cycles sont nécessaires pour récupérer les réponses et P_2 cycles pour appliquer le contenu des chaînes de scan au circuit (1 cycle par vecteur). On obtient l'équation suivante:

$$T_2 = N_{SI_C} + N_{SI_{NC}} \cdot \left(\left\lceil \frac{N}{M} \right\rceil + 1 \right) + L + P_2 \qquad \text{Équation 2.1}$$

2.2.4.2. Equation volume de données

L'équation 2.2 donne le volume de données d'une séquence compressée. Le volume V_2 est le produit du nombre total de mots stockés dans la mémoire du testeur (compressés: N_{SL_C} et non compressés: $N_{SL_{NC}} * \lceil N/M \rceil$) par leur taille M. L'expression est donc la suivante:

$$V_2 = M \cdot \left(N_{S_C} + N_{S_{NC}} \cdot \left\lceil \frac{N}{M} \right\rceil \right) \qquad \text{Équation 2.2}$$

On remarquera que ces équations sont toutes deux dominées par le terme relatif au nombre de slices non compressées $N_{S_{NC}}$. On retrouve ce qui a déjà été commenté précédemment, à savoir que plus le nombre de slices non compressées augmente, plus le temps de test et le volume des données à stocker augmentent à leur tour. Le besoin de réduire l'occurrence de ce cas de figure prend toute son importance à la vue de ces équations théoriques de performance de notre structure.

2.3. Résultats et comparaisons

Cette section présente les résultats expérimentaux obtenus à l'aide de notre technique de compression des données de test. Dans un premier temps, nous présentons la technique standard de chargement série des données. Une comparaison avec cette méthode nous permettra, dans une seconde partie, de mettre en avant les résultats obtenus par notre technique.

2.3.1. Comparaison avec une méthode standard de sérialisation des données

Nous avons choisi de comparer notre technique de compression à une technique de sérialisation des données pour deux raisons. La première vient du fait que cette technique répond parfaitement au cahier des charges que nous nous sommes fixé: indépendance vis-à-vis des données de test, non nécessité de la netlist du circuit, pas d'utilisation d'outils de test spécifique, et enfin un décompresseur sur puce faible coût.

La seconde raison tient du caractère générique de cette structure de test qui la rend analysable et facilement comparable à une autre structure. Pour une séquence de test donnée, cette structure se présente donc comme une référence à laquelle il est naturel de comparer les solutions de compression des données de test en termes de réduction du volume de données et du temps de test.

2.3.1.1. Principe et description

Comme son nom l'indique, cette technique est simplement basée sur un chargement série des données de test dans les chaînes de scan. Chaque slice de N bits est stockée dans le la mémoire du testeur sous la forme de $\lceil N/M \rceil$ mots de M bits, comme illustré sur la Figure 2.55. Il faut $\lceil N/M \rceil$ cycles d'horloges pour charger les données de la slice dans les M registres à décalage. Pour finir, la slice est chargée dans les chaînes de scan en un dernier cycle d'horloge. L'opération est répétée pour chaque slice. Une fois tout le vecteur chargé, le circuit passe en mode normal et le vecteur est appliqué au circuit. Les réponses sont ensuite récupérées au fur et à mesure que les slices suivantes sont chargées.

Notons ici que cette méthode n'est en rien une technique de compression. Il s'agit simplement de la plus simple façon de raccorder M canaux testeurs à N chaînes de scan. Les données ne sont

aucunement modifiées. De fait, à la différence des techniques de compression, la présence de valeurs indéterminées (X) ne peut être exploitée pour réduire le temps de test et/ou le volume des données.

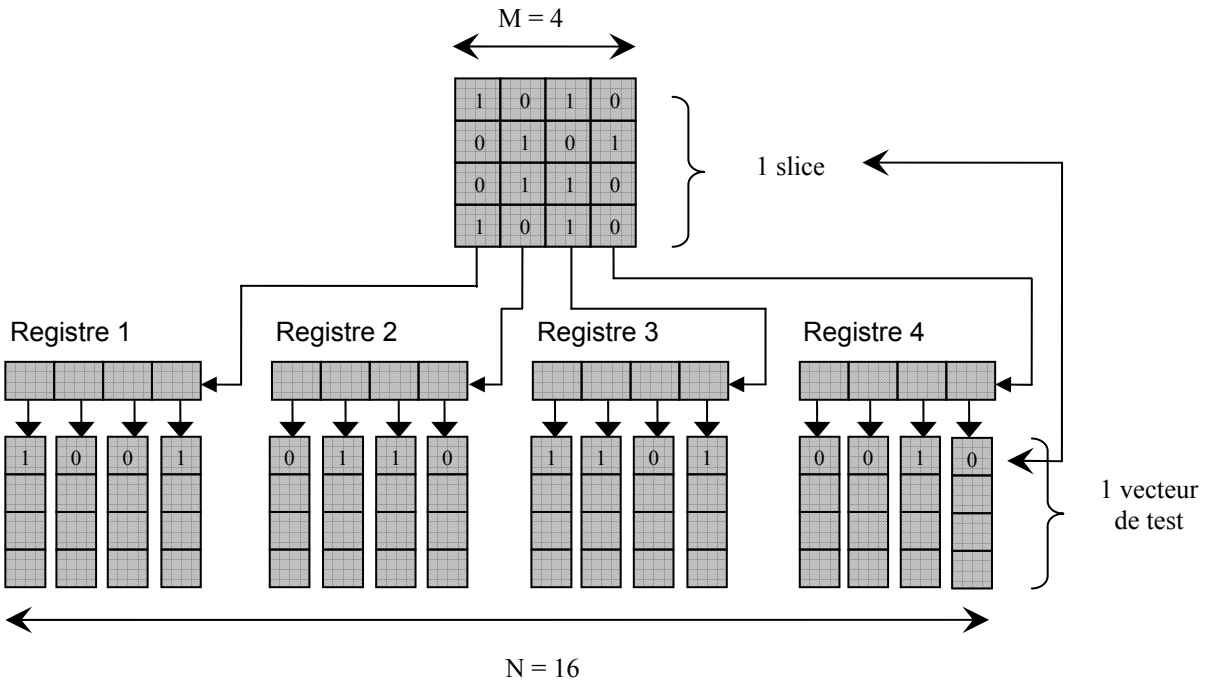


Figure 2.55 : Sérialisation des données de test

2.3.1.2. Performance

Pour la sérialisation, les équations 2.3 et 2.4 représentent les expressions du temps et du volume. P_1 représente le nombre de vecteurs. Dans l'équation 2.3, le premier terme correspond au chargement des vecteurs dans les chaînes de scan. Le second terme, lui, représente la récupération des réponses du circuit tandis que le troisième correspond aux cycles de capture de chaque vecteur. Dans l'équation 2.4, le volume total d'une séquence de test est le nombre final de slices multiplié par leur taille M.

$$T_1 = P_1 \cdot L \cdot \left(\left\lceil \frac{N}{M} \right\rceil + 1 \right) + L + P_1 \qquad \text{Équation 2.3}$$

$$V_1 = M \cdot L \cdot \left\lceil \frac{N}{M} \right\rceil \cdot P_1 \qquad \text{Équation 2.1}$$

2.3.2. Résultats expérimentaux

Dans cette section sont présentés les résultats expérimentaux obtenus avec notre algorithme de compression. Nous nous intéresserons en particulier au volume des données de test à stocker dans le testeur et surtout au temps d'application de test. Ces résultats sont comparés avec ceux obtenus par la structure de référence de sérialisation des données.

Dans un premier temps nous présenterons la procédure expérimentale, puis nous donnerons des résultats significatifs pour plusieurs configurations de chaînes de scan d'un circuit. Finalement, des résultats relatifs à d'autres circuits sera donnée dans une dernière partie.

2.3.2.1. Procédure expérimentale

Les résultats expérimentaux présentés ici sont basés sur des séquences de vecteurs de test relatives aux circuits de référence ISCAS'89 [Benc89] dont certains détails sont donnés en Annexe B. Ce sont des circuits conçus spécifiquement pour mettre à l'épreuve les différents travaux concernant le test des circuits intégrés séquentiels. Cependant, pour ces circuits, on ne dispose que des informations structurelles aussi appelées netlists. Il n'existe pas de séquence de test de référence qui permettrait à toutes les équipes travaillant sur la compression des données de test de pouvoir comparer leur techniques en toute équité. Nous avons donc généré nos propres séquences de test grâce à l'outil de génération de vecteur de test de Synopsis®, TETRAMAX® [Syno01].

Les séquences de vecteurs de test ont été générées suivant deux modes. Une première fois avec compaction des vecteurs et sans remplissage aléatoire des valeurs indéterminées. Dans la suite, ces séquences sont dénommées "type 2". Une seconde fois toujours avec compaction mais cette fois avec remplissage de ces X pour des séquences baptisées "type 1". Les séquences "type 2" seront compressées à l'aide de notre algorithme de compression, puis appliquées aux circuits par l'intermédiaire de notre structure de décompression. Elles contiennent, en moyenne, 80% de valeurs indéterminées. Les séquences dites "type 1", quant à elle, ne contiennent plus de X mais aussi moins de vecteurs. En effet, le remplissage aléatoire associé à la compaction des vecteurs permet de réduire nettement le nombre de vecteurs afin d'éviter une redondance dans la détection des fautes. Pour effectuer une comparaison honnête, ce sont ces séquences ("type 1) qui seront appliquées à la sérialisation. Il ne serait pas équitable de pénaliser cette structure avec des séquences plus longues contenant des X. Il est important de noter ici que ces deux types de séquences assurent strictement le même taux de couverture de fautes

En résumé, les séquences "type 2", contenant des X et plus longue en nombre de vecteurs, seront utilisées avec notre méthode alors que les séquences "type 1", sans X et plus courte" seront utilisées pour la structure de sérialisation des données de test.

2.3.2.2. Analyse des résultats pour un ou 2 circuits complets

Le Tableau 2.5 présente les résultats correspondant aux séquences générées pour le circuit s5378 pour trois configurations de chaînes de scan. Les 214 bascules du circuit sont organisées respectivement en N=8 chaînes de scan de L=27 bascules, puis en N=16 chaînes de L=14 et finalement en N=32 chaînes de L=7. Les résultats pour M variant de 5% à 50% de N sont présentés dans ce tableau. La valeur "%X" représente le pourcentage de X dans la séquence, "volume (bits)" le volume de données en bits stockées dans la mémoire testeur, "taux comp." est le taux de compression atteint et "temps (cycles)" est le nombre de cycles testeurs nécessaires à l'application de la séquence au circuit. Les deux dernières colonnes de la table représentent les gains en temps et

volume atteints par notre compression avec les séquences de type 1 par rapport à la sérialisation des séquences de type 2. Rappelons que les séquences de type 2 comportent 217 vecteurs pour en moyenne 82% de valeurs indéterminées, alors que les séquences de type 1 ne comportent que 160 vecteurs ne contenant aucune valeur indéterminée.

			Compression - 217 vecteurs						Sérialisation - 160 vecteurs		Gain	
L	N	M	% X	volume original (bits)	volume comp (bits)	taux comp %	Nsl_c	Temps (cycles)	Volume (bits)	Temps (cycles)	Volume %	Temps %
27	8	1	82,57	46872	13181	63,37	4813	14471	34560	39067	61,87	62,96
		2			17670	59,41	4867	10071	34560	21787	48,88	53,78
		4			25472	44,63	5350	7121	34560	13147	26,3	45,84
14	16	1	82,78	48608	13988	56,41	2308	14949	35840	38254	60,98	60,93
		2			15792	55,2	2344	8821	35840	20334	55,94	56,62
		4			19172	30,87	2453	5609	35840	11374	46,51	50,69
		8			27328	19,39	2660	4025	35840	6894	23,75	41,62
7	32	2	83,19	48608	16928	47,91	1056	9151	35840	19207	52,77	52,36
		3			18177	47,15	1065	6737	36960	13607	50,82	50,49
		8			21992	45,17	1109	3383	35840	5767	38,64	41,34
		16			29904	33,14	1169	2443	35840	3527	16,57	30,74

Tableau 2.5 : Résultats obtenus pour le circuit s5378 pour trois configurations de chaînes de scan

La Figure 2.56 montre que le volume de données de test augmente quand M se rapproche de N. On constate que les mots compressés prennent plus de place mémoire quand M est grand et que, même si la compression est plus efficace avec un M grand (plus de slices compressibles), elle ne parvient pas à compenser cette augmentation. Deuxièmement, comme on vient de le dire, la compression est plus efficace à mesure que M augmente, on observe donc que temps de test diminue en conséquence.

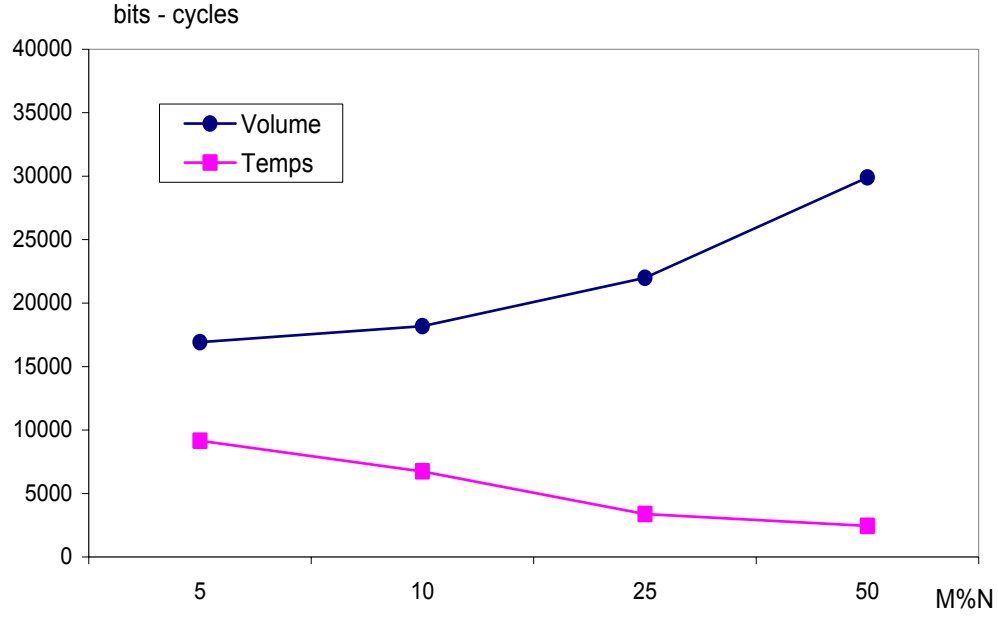


Figure 2.56 : Résultats de la compression pour la configuration 32 chaînes de scan

En ce qui concerne les résultats de la sérialisation, on peut remarquer sur la Figure 2.57 et sur le Tableau 2.5 que le volume des données reste constant (la petite variation est due à la mise sous forme de slices de $\lceil N/M \rceil$ mots de M bits). Le temps de test quant à lui diminue à mesure que M augmente. Cela s'explique par le fait que plus les slices sont grandes et que le volume de test ne change pas, il y a moins de slices à charger, donc moins de cycles de test.

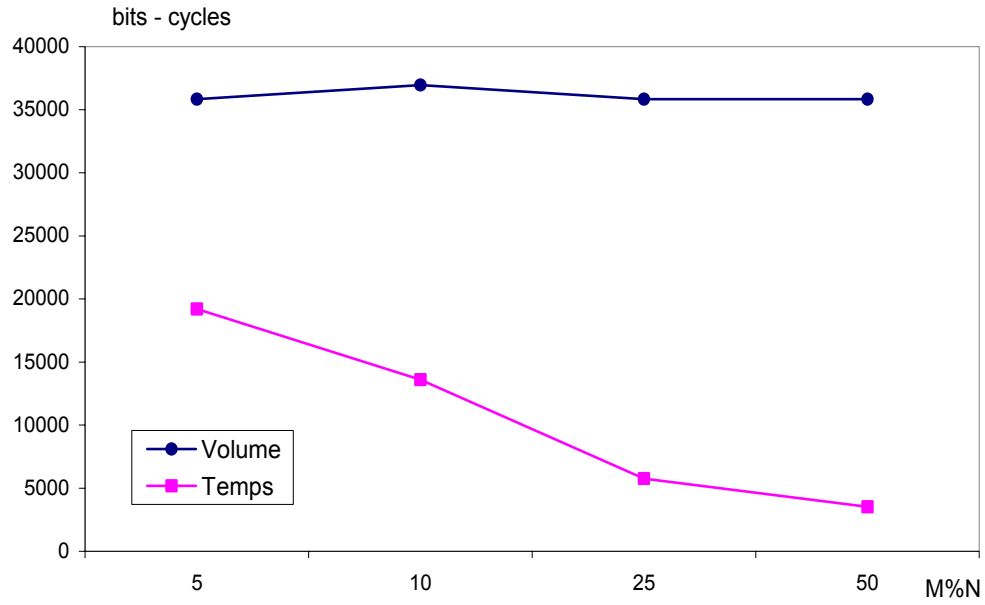


Figure 2.57 : Résultats de la sérialisation pour la configuration 32 chaînes de scan

Les valeurs numériques de gains de notre méthode par rapport à la sérialisation sont données dans les deux dernières colonnes du Tableau 2.5 ainsi que sur la Figure 2.58. En ce qui concerne le gain en volume, on constate qu'il décroît rapidement à mesure que le nombre de canaux testeur (M) augmente. Ceci est du au fait que le volume de données des séquences utilisées pour la sérialisation reste constant quelque soit la configuration choisie tandis que le volume des données utilisées avec notre technique de compression augmente avec M. Quant au gain en temps de test, il diminue également à mesure que M augmente. En effet, pour les deux techniques, plus le nombre de canaux testeurs est grand, moins il y a de slices à transmettre aux chaînes de scan et plus la compression est facile. Cela dit, comme les séquences de "type 2" appliquées à la sérialisation des données sont plus courtes, l'impact de l'augmentation de M est plus fort sur cette dernière, d'où la diminution du gain.

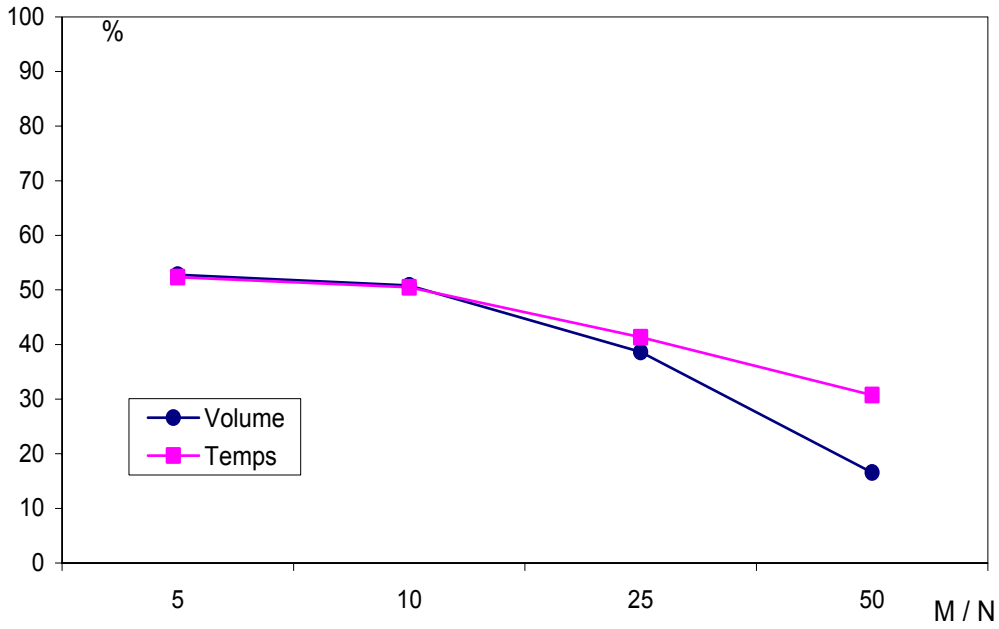


Figure 2.58 : Gains en temps de test et volume pour la configuration 32 chaînes de scan

La diminution des gains avec l'augmentation du rapport M/N est logique, puisque, dans le cas extrême où aucune compression horizontale n'est appliquée (M = N ou M/N = 100%), notre technique et la sérialisation sont équivalentes (chaque canal testeur alimente chaque chaîne de scan). Or les séquences appliquées à la sérialisation sont plus courte que celles utilisées pour notre technique de compression. Les résultats sont alors en faveur de la sérialisation. Il y a donc un seuil dans l'augmentation du nombre de canaux testeur où la sérialisation devient plus efficace. Ce seuil dépend d'une part de l'écart en nombre de vecteurs entre les deux séquences utilisées, et d'autre part du nombre de slice non compressées par notre technique.

Cependant, la tendance actuelle est d'augmenter le nombre de chaînes de scan N pour diminuer leur longueur et ainsi réduire le temps de test. Donc, pour un nombre de canaux testeur fixé, le rapport M/N a tendance à diminuer (M << N). Dans ces situations, notre technique est toujours nettement plus performante que la sérialisation avec des gains compris entre 40 et 50% en temps de test et volume de données.

2.3.2.3. Généralisations aux autres circuits

De très nombreuses autres expériences ont été effectuées sur d'autres circuits de références ISCAS'89. Le Tableau 2.6 présente quelques résultats tirés de ces expérimentations. Les valeurs P_1 et P_2 représentent respectivement de nombre de vecteurs des séquences "type 1" (compaction et remplissage aléatoire des X) pour la sérialisation et des séquences "type 2" (compaction uniquement) pour la compression. A l'instar du circuit s5378, on constate que les séquences de "type 1" contiennent toujours moins de vecteurs que celles de type 2" (P2 < P1)

			compression							sérialisation			Gain	
Circuit	FF	L'	P2	% X	volume original (bits)	volume comp. (bits)	taux comp %	Nsl_c	Temps (cycles)	P1	Volume (bits)	Temps (cycles)	Volume %	Temps %
s5378	214	7	217	83,19	48608	21992	54.76	1109	3383	160	35840	5767	38,64	41,34
s9234	247	8	215	77,3	55040	30176	45.17	1036	4679	186	47616	7634	36,63	38,71
s13207	699	22	272	93	191488	65296	65.9	5258	9182	250	176000	27772	62,9	66,94
s15850	611	20	142	84	90880	42520	53.21	2015	6302	126	80640	12746	47,27	50,56
s35932	1763	56	26	50	46592	28256	39.35	764	4306	25	44800	7081	36,93	39,19
s38417	1664	52	440	93,83	732160	244192	66.65	20332	33564	145	241280	37897	-1,2	11,43
s38580	1464	46	189	85,49	278208	119448	57.07	6615	17245	142	209024	32848	42,85	47,5

Tableau 2.6 : Résultats d'autres circuits ISCAS'89 pour 32 chaînes de scan et 8 canaux testeurs

On remarque que notre méthode permet une réduction du volume des données moyenne de 37% et du temps de test moyen de l'ordre de 42%. On notera tout de même une exception sur les résultats du circuit s38417. Ceci est dû au fait que le nombre de slice compressé (noté Nsl_c sur le Tableau 2.6) n'est pas assez important pour combler l'écart en nombre de vecteurs entre les séquences "type 1" (440 vecteurs) et "type 2" (145 vecteurs). Cela dit, cette contre performance de notre méthode n'est apparue que pour cette configuration. Dans les autres configurations, notre technique produit un plus grand nombres de slices compressées, suffisant pour combler l'écart en nombre de vecteurs entre les deux séquences, et ainsi avoir des gains en volumes et temps de test du même ordre de grandeur que pour les autres circuits.

2.4. Conclusion

Dans ce chapitre, une méthode simple de compression horizontale des données de test permettant de réduire le volume des données et le temps de test a été présentée. Cette méthode répond parfaitement au cahier des charges fixé dans la section 2.2.1. Premièrement, elle est indépendante des séquences de test et des netlists des circuits dans la mesure où la structure de compression-décompression n'a besoin d'aucunes informations structurelles sur le circuit pour être appliqué à la séquence de test. Deuxièmement, aucun outil de conception spécifique n'est requis : en effet notre technique est parfaitement générique est s'applique directement à la séquence de test, peu importe la manière dont elle a été obtenue. Troisièmement, le décompresseur sur puce associé à notre algorithme de compression ne requiert que très peu de surface additionnelle : uniquement un additionneur et un registre N bits. Et Enfin, notre méthode ne pénalise en rien le taux de couverture de fautes d'une séquence de test donnée : elle peut même, dans le cas où le taux de couverture de la séquence originale n'est pas de 100%, augmenter ce taux de couverture grâce au remplissage des valeurs indéterminées (X)

L'objectif de la compression horizontale des données est de réduire le volume des données et le temps de test dans le cadre de circuits dont le nombre de chaînes de scan est nettement supérieur au nombre de canaux disponibles sur les testeurs. Or pour les gros circuits actuels, et selon la loi de Moore, cette tendance se maintiendra dans le futur, le nombre de chaînes de scan est choisir de plus

en plus grand de façon à minimiser le temps de chargement de ces chaînes. Dans ce cadre, notre méthode a été comparée à la technique standard de sérialisation des données de test. Il a été observé lors de très nombreuses expériences que, même pénalisée par des séquences de test plus longues à cause de la présence de valeurs indéterminées, notre technique atteint des gains de temps entre 25 et 50% et des gains en volume entre 15 et 60 % par rapport à la sérialisation.

Notre technique est donc largement avantageuse pour réduire le volume de données et le temps de test dans le cadre du test de circuits dont le nombre de canaux testeur est nettement inférieur au nombre de chaînes de scan, et cela tout en répondant strictement au cahier des charges fixés par l'objectif du test des IP et des systèmes sur puces (SOC).

2.5. Références bibliographiques

[Bard87] P.H. Bardell, W.H. McAnney, J. Savir: Built-in test for VLSI: pseudorandom techniques. John Wiley and sons, 1987.

[Bard82] P.H. Bardell, W.H. McAnney: Self-testing of multichip logic modules. Proceedings of the International Test Conference, ITC 1982, pp. 200-2004.

[Press99] M. Pressly, D. Das, C. Hunter: LBIST for PowerPC™ Embedded Core Microprocessors: Feasible or Not?. International Workshop on Microprocessor Test and Verification, 1999.

[Das00] D. Das, N.A. Touba: Reducing test data volume using external/lbist hybrid test patterns. Proceedings of the International Test Conference, ITC 2000, pp. 115-122.

[Pome93] I. Pomeranz, S. Sudhakar, M. Reddy: 3-weight pseudo-random test generation based on a deterministic test set for combinational and sequential circuits. IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems, Vol 12, N° 7, 1993, pp. 1050-1058.

[Jas01] A. Jas, C.V. Krishna, N.A. Touba: Hybrid BIST based on weighted pseudo-random testing: a new test resource partitioning scheme. Proceedings of the 19th VLSI Test Symposium, VTS 2001, pp. 2-8.

[Wund96] H-J. Wunderlich, G. Kiefer: Bit Flipping BIST. Proceedings of the International Conference on Computer-Aided Design, ICCAD 1996, pp. 337-343.

[Hell00] S. Hellebrand, H. Liang, H-J. Wunderlich: A mixed mode BIST scheme based on reseeding of folding counters. Proceedings of the International Test Conference, ITC 2000, pp. 778-784.

[Kay01] D. Kay, S. Mourad: Compression technique for interactive BIST application. Proceedings of the 19th VLSI Test Symposium, VTS 2001, pp. 9-14.

[Dors98] R. Dorsch, H-J. Wunderlich: Accumulator based deterministic BIST. Proceedings of the International Test Conference, ITC 1998, pp. 412-421.

[Doul07] M. Doulcier, M-L. Flottes, B. Rouzeyre: AES vs LFSR Based Test Pattern Generation: A Comparative Study. Proceedings of the 8th Latin-American Test Workshop, LATW 2007, pp. 314-321.

[Jas99] A. Jas, J. Ghosh-Dastidar, N.A. Touba: Scan vector compression decompression using statistical coding. Proceedings of the 17th VLSI Test Symposium, VTS 1999, pp. 114-120.

[Jas03] A. Jas, J. Ghosh-Dastidar, M-E. Ng, N.A. Touba: An efficient test vector compression scheme using selective Huffman coding. IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems, Vol. 22, N° 6, 2003, pp. 797-806.

[Jas98] A. Jas, K. Mohanram, N.A. Touba: An embedded core DFT scheme to obtain highly compressed test sets. Proceedings of the 8th Asian Test Symposium, ATS 1998, pp. 275-280.

[Chan01] A. Chandra, K. Chakrabarty: Frequency-directed run-length (FDR) codes with application to system-on-a-chip test data compression. Proceedings of the 19th VLSI Test Symposium, VTS 2001, pp. 42-47.

[Chan03] A. Chandra, K. Chakrabarty: Test data compression and test resource partitioning for system-on-a-chip using frequency-directed run-length (FDR) codes. IEEE Transaction on Computer, Vol. 52, N° 8, 2003, pp. 1076-1088.

[Chan00] A. Chandra, K. Chakrabarty: Test data compression for system-on-chip using Golomb code. Proceedings of the 18th VLSI Test Symposium, VTS 2000, pp 113-120.

[Chak01] A. Chandra, K. Chakrabarty: Efficient test data compression and decompression for system-on-chip using internal scan chain and Golomb coding. Proceedings of the Conference on Design, Automation and Test in Europe, DATE 2001, pp. 145-150.

[Toub06] N.A. Touba: Survey of Test Vector Compression Techniques. Design & Test Vol. 23, N° 4, 2006, pp. 294-303.

[Tang03] H. Tang, S.M. Reddy, I. Pomeranz: On reducing test data volume and test application time for multiple scan chain designs. Proceedings of the International Test Conference, ITC 2003, pp. 1079-1087.

[Patt90] D.A. Patterson, J.L. Hennessy: Computer architecture: a quantitative approach. Morgan Kaufmann Publishers Inc., 1990, pp. 583-584.

[Orai04] B. Arslan, A. Orailoglu: CircularScan: a scan architecture for test cost reduction. Proceedings of the Conference on Design, Automation and Test in Europe, DATE 2004, pp. 1290-1296.

[Arsl04] B. Arslan, A. Orailoglu: Test cost reduction through a reconfigurable scan architecture. Proceedings of the International Test Conference, ITC 2004, pp. 945-952.

[Hamz99] I. Hamzaoglu, J.H. Patel: Reducing test application time for full scan embedded cores. Proceedings of the twenty-ninth annual international Symposium on Fault-Tolerant Computing, FTCS 1999, pp. 260-267.

[Pate02] A. Pandey, J. Patel: An incremental algorithm for test generation in Illinois scan architecture based designs. Proceedings of the Conference on Design, Automation and Test in Europe, DATE 2002, pp. 368-375.

[Pand02] A. Pandey, J. Patel: Reconfiguration technique for reducing test time and test data volume in Illinois scan architecture based designs. Proceedings of the 20th VLSI Test Symposium, VTS 2002, pp. 9-15.

[Chan07] A. Chandra, H. Yan, H., R. Kapur: Multimode Illinois scan architecture for test application time and test data volume reduction. Proceedings of the 25th VLSI Test Symmposium, VTS 2007, pp. 84-89.

[Sama03] S. Samaranayake, E. Gizdarski, N. Sitchinava, F. Neuveux, R. Kapur, T.W. Williams: A reconfigurable shared scan-in architecture. Proceedings of the 21st VLSI Test Symposium, VTS 2003, pp. 9-14.

[Sitc04] N. Sitchinava, S. Samaranayake, R. Kapur, E. Gizdarski, F. Neuveux, T.W. Williams: Changing the scan enable during shift. Proceedings of the 22nd VLSI Test Symposium, VTS 2004, pp. 73-78.

[Wang04] L. Wang, K.S. Abdel-Hafez, S. Wu, X. Wen, H. Furukawa, F. Hsu, S. Lin, S. Tsai: VirtualScan: A new compressed scan technology for test cost reduction. Proceedings of the International Test Conference; ITC 2004, pp. 916-925.

[Bayr01] I. Bayraktaroglu, A. Orailoglu: Test volume and application time reduction through scan chain concealment. Proceedings of the 38th Annual Design Automation Conference, DAC 2001, pp. 151-155.

[Bala05] K.J. Balakrishnan, N.A. Touba: Reconfigurable linear decompressors using symbolic gaussian elimination. Proceedings of the Conference on Design, Automation and Test in Europe, DATE 2005, pp. 1130-1135.

[Köne91] B. Könemann: LFSR-coded test pattern for scan designs. Proceeding of the European Test Conference, ETC 1991, pp. 237-242.

[Kris02] C.V. Krishna, N.A. Touba: Reducing test data volume using LFSR reseeding with seed compression. Proceedings of the International Test Conference, ITC 2002, pp. 321-330.

[Kris01] C.V. Krishna, A. Jas, N.A. Touba: Test vector encoding using partial LFSR reseeding. Proceedings of the International Test Conference, ITC 2001, pp. 885-893.

[Rajs02] J. Rajki, J. Tyzer, M. Kassab, N. Mukherjee, R. Thompson, K-H. Tsai, A. Hertwig, N. Tamarapalli, G. Mrugalski, G. Eide, J. Qian: Embedded Deterministic Test for low-cost manufacturing test. Proceedings of the International Test Conference, ITC 2002, pp. 301-310.

[Li03] L. Li, K. Chakrabarty, N.A. Touba: Test data compression using dictionaries with selective entries and fixed-length indices. ACM Transaction on Design Automation of Electronic Systems, Vol. 8, N° 4, 2003, pp. 470-490.

[Wurt04] A. Wurtenberger, C.S. Tautermann, S. Hellebrand: Data compression for multiple scan chains using dictionaries with corrections. Proceedings of the International Test Conference, ITC 2004, pp. 926-935.

[Poir04] R. Poirier: Compression de données pour le test des circuits intégrés. Thèse de doctorat, LIRMM – Université de Montpellier II, 2004.

[Benc89] F. Brglez, D. Bryant and K. Kozminski: Combinational profiles of sequential benchmark circuits. Proceedings of International Symposium on Circuits And Systems, ISCAS 1989, pp. 1929-1934.

http://www.cbl.ncsu.edu/benchmarks/Benchmarks-upto-1996.html

[Syno01] http://www.synopsys.com/tools/implementation/rtlsynthesis/pages/tetramaxatpg.aspx

Chapitre III:

Compression des données de test et SoCs

3.1. Test des SoCs : principe 101

3.1.1. Architecture de test : la norme 1500 101

3.1.2. Définition de l'ordonnancement du test des cœurs 102

3.1.3. Problèmes induits par le test multi cœurs 105

3.2. Etat de l'art 105

3.2.1. Méthode d'optimisation 105

3.2.1.1.1. Conception des wrappers 105

3.2.1.1.2. Conception du TAM 106

3.2.2. Introduction de la compression 107

3.3. Méthode proposée 108

3.3.1. Présentation 108

3.3.1.1. Compression et temps de test 108

3.3.1.2. Compression et test en parallèle 109

3.3.1.3. Compression et architecture de test 110

3.3.2. Problème et notations 111
3.3.3. Algorithmes 112
3.3.3.1. Temps de test des cœurs 113
3.3.3.2. Exploration des partitions 115
3.3.3.3. Assignation des cœurs et Ordonnancement 116
3.3.3.4. Illustration 118
3.4. Résultats 121
3.4.1. Procédure expérimentale 121
3.4.2. Résultats et analyse 123
3.4.2.1. Wrappers fixés. 123
3.4.2.2. Wrappers libres. 125
3.5. Conclusion 126
3.6. Références bibliographiques 128

Dans les chapitres précédents, nous avons vu comment le test des circuits devenait de plus en plus complexe et donc, de plus en plus cher. Différentes approches permettant de réduire ce coût ont été passées en revue: intégration des moyens de test directement sur le circuit, codage des données de test, ou encore augmentation du nombre de chaînes de scan sans augmenter le nombre de canaux testeur.

Dans ce chapitre, nous présentons dans un premier temps le principe du test des systèmes sur puce (ou SoC pour System-on-Chip) ainsi que les différentes approches d'optimisation proposées dans la littérature. Dans un second temps, nous proposons une méthode utilisant une technique de compression horizontale pour réduire le temps de test des systèmes et par conséquent leur coût. L'idée générale est d'utiliser la compression horizontale pour augmenter le nombre de cœurs pouvant être testés simultanément. Toutefois, on ne peut pas avoir plus de chaînes de scan pour tester les cœurs que de canaux disponibles sur le testeur externe. La compression permet justement d'augmenter le nombre de chaînes de scan sans augmentation du nombre de canaux testeur.

Comme nous l'avons vu dans le chapitre 2.1.4, la plupart des techniques de compression ne sont pas applicable au test des SoC et des IP. Par simplicité, nous utiliserons ici la technique de compression qui a été présentée dans le Chapitre 2:, mais toute autre méthode compatible avec le test des systèmes pourrait être utilisée à la place. L'analyse des résultats apportés par notre méthode conclue ce chapitre.

3.1. Test des SoCs : principe

Un système est un assemblage de cœurs et/ou d'IP (Figure 3.1) reliés entre eux par des circuits de logique glue. La tâche principale du test du système consiste à tester les différents circuits du système ainsi que la logique glue connectant les cœurs entre eux. La principale difficulté dans ce cadre, réside dans l'acheminement d'une séquence de test au cœur auquel elle est destinée.

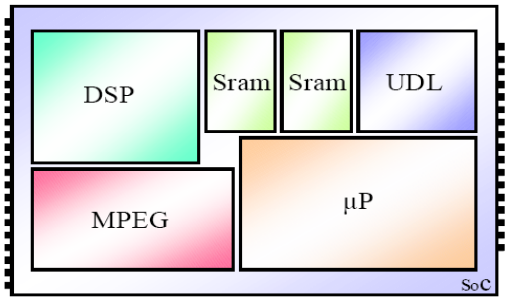


Figure 3.1 : Exemple de système sur puces contenant 6 cœurs

3.1.1. Architecture de test : la norme 1500

Pour répondre aux besoins du test des systèmes, une architecture permettant d'acheminer les données de test aux différents cœurs a été proposée par l'organisation IEEE, définie sous la norme **IEEE 1500** [IEEE05]. Elle consiste principalement en un bus d'acheminement des données de test (ou **TAM** pour Test Acces Mechanism) et d'un anneau d'isolation autour des cœurs (ou **wrappers**)

comme illustré sur la Figure 3.2. Le TAM relie les entrées/sorties de test du système aux cœurs, le wrapper servant d'interface entre le cœur proprement dit et les entrées du système. En mode test, le wrapper connecte le cœur aux entrées de test.

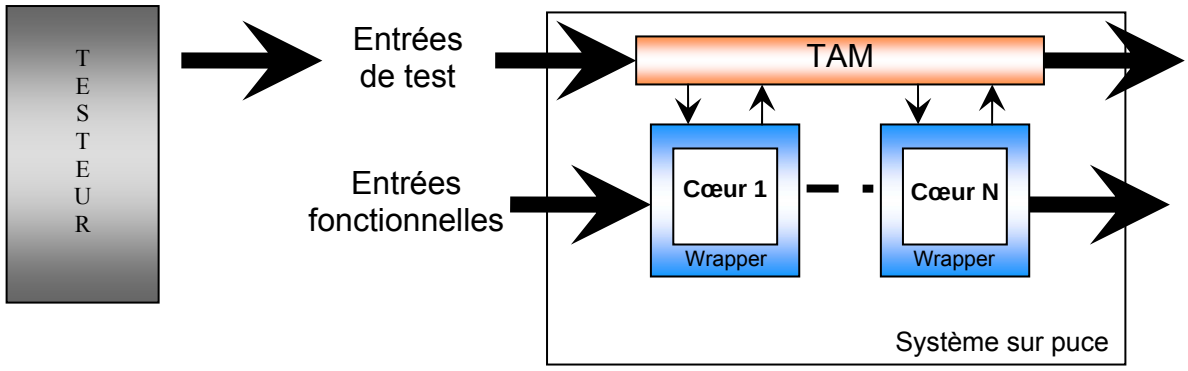


Figure 3.2 : Architecture de test des systèmes IEEE 1500

Cette architecture permet donc de rendre chaque cœur visible par le testeur et ainsi de pouvoir transmettre et récupérer leurs données de test respectives.

3.1.2. Définition de l'ordonnancement du test des cœurs

Dans la suite de ce chapitre, nous utiliserons les notations suivantes : soit W_{TAM} la taille du bus de test (TAM) en bits, Wc_i le nombre de bits nécessaire à la connexion du cœur i sur le TAM, t_i le temps de test du cœur i et Ttot le temps de test du système complet. Les temps de test sont exprimés en nombre de cycles.

Une première approche du test du système consiste à tester les cœurs les uns à la suite des autres. Le temps de test du système complet Ttot est égal à la somme des temps de test individuels des cœurs : $Ttot = \sum_{i=1}^{n} t_i$.

Un exemple de cette approche est donné sur la Figure 3.3 : soit un système de 4 cœurs C1, C2, C3 et C4 ayant respectivement pour caractéristiques lors du test (Wc_i , t_i) : (2,4), (2,3), (1,5) et (3,5). Un TAM de 4 bits est utilisé pour acheminer les données de test.

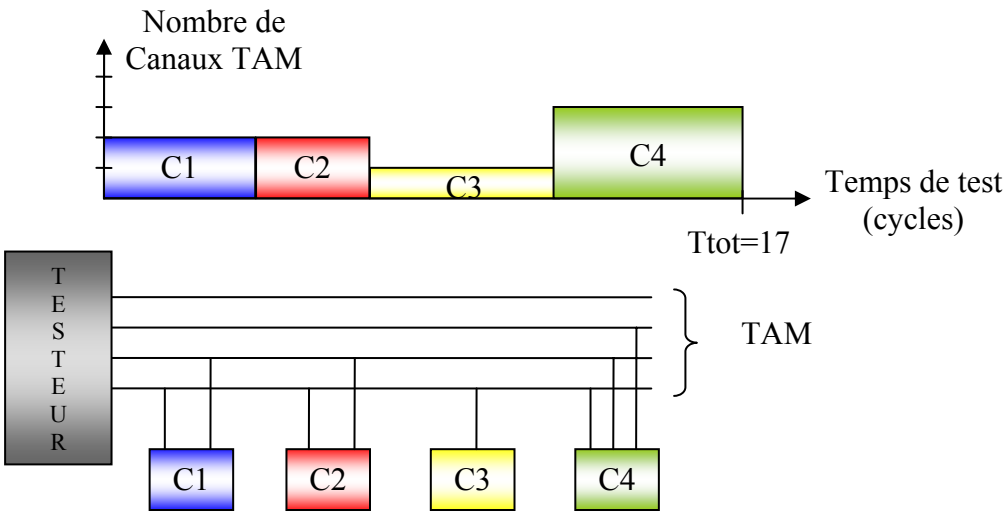


Figure 3.3 : Test en série des cœurs d'un système

Il apparaît évident que cette solution peut être optimisée en termes de temps de test. Le test simultané de plusieurs cœurs réduit de manière significative le temps de test total du système (Figure 3.4). Il faut noter ici que l'implantation d'un contrôleur dans le système est nécessaire à la réalisation et au bon déroulement de ce type de test. La surface additionnelle de silicium due à ce contrôleur est négligeable devant la taille du système lui-même puisque il se résume principalement à des compteurs et une machine d'état gérant les signaux d'activation du test de chaque cœur.

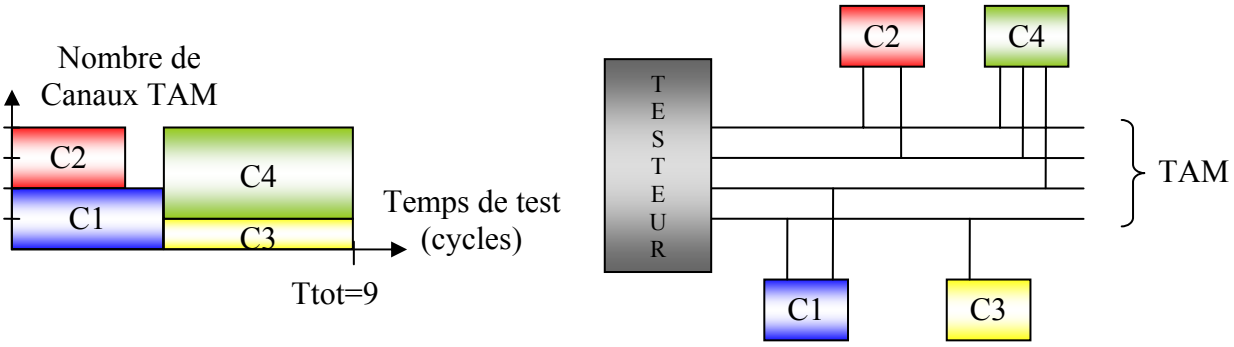


Figure 3.4 : Test en parallèle des cœurs d'un système

L'ordonnancement du test des cœurs dans le cadre du test des systèmes consiste à paralléliser le test des cœurs. Cet ordonnancement doit cependant être effectué dans la limite de la taille du bus de test W_{TAM} et de la taille des wrappers Wc_i. D'autres contraintes peuvent venir s'ajouter à cela telles que l'augmentation de consommation électrique entraînée par le test simultané de plusieurs cœurs [Lars02] [Iyen02] par exemple. Par souci de clarté, cette dernière contrainte ne sera pas prise en compte dans la suite de ce chapitre, mais il serait tout à fait envisageable de l'intégrer à la méthode proposée dans la suite.

Trois principales approches d'ordonnancement ont été proposées dans la littérature. La première utilise des "sessions de test de longueur fixe" [Chou97]. Une illustration est donnée sur la Figure 3.5 (a). On considère ici que le temps de test du cœur le plus long fixe la longueur des sessions (le temps T_1 du cœur 1 sur l'exemple). Le temps de test total est un multiple du temps de session, Ttot = 3*T_1 dans l'exemple. Le temps de test est optimal si les longueurs de test des cœurs sont similaires. Cependant, dès que les cœurs ont des temps de test différents, il apparaît que cette solution introduit des temps d'inactivité. Le principal intérêt de cette méthode est la simplicité du contrôleur de test.

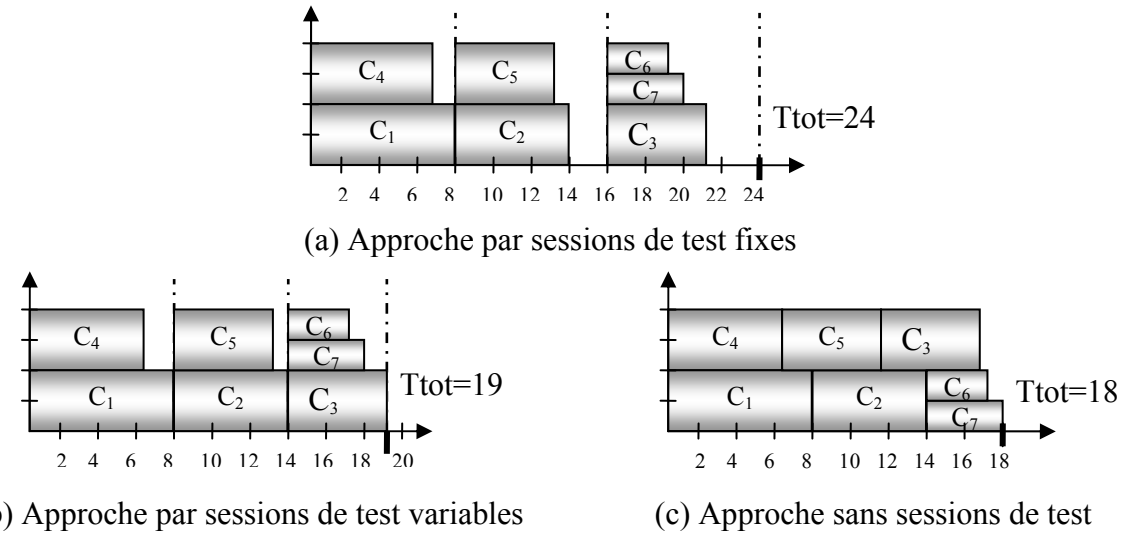


Figure 3.5 : Différentes approches de l'ordonnancement du test

Une seconde approche visant à éliminer ces temps "inutiles" et par conséquent à réduire le temps de test total a été proposé [Mure00] [Lars01]. Elle utilise cette fois le principe de "sessions de test variables" dont le principe est illustré sur la Figure 3.5 (b). Cette approche vise à optimiser le temps de test, perdu lorsque les temps de test des différents cœurs sur le système sont différents. La longueur d'une session est calée sur la durée du test le plus long dans la session considérée (respectivement T_1, T_2 et T_3). Le temps total est alors la somme des temps de session (Ttot = T_1+T_2+T_3). L'espace des solutions couvert par cette méthode est nettement plus important et permet d'améliorer le temps par rapport à l'approche précédente. Cependant cette approche requiert, d'une part, plus de temps de calcul pour l'obtention d'une solution, et d'autre part, plus de logique de contrôle sur le circuit.

La dernière approche consiste à ne considérer aucune session de test [Iyen01] et démarrer le test d'un cœur dès que les contraintes le permettent, comme illustré sur la Figure 3.5 (c). Cela nécessite d'une part, que le contrôleur central ait plus de signaux à gérer, mais aussi d'avoir une machine d'état plus complexe car le contrôleur attend la fin du test de chaque cœur pour lancer le test des cœurs suivants. Il est évident que cette approche obtient les meilleurs résultats en termes de temps de test total des systèmes. Cependant une attention particulière doit être portée sur le temps de calcul nécessaire à l'obtention d'une solution optimale dans le cas de systèmes contenant

beaucoup de cœur. Des approches heuristiques ont été proposées, notamment dans [Flot01] afin de fournir des solutions, certes sans garanties d'optimalité mais néanmoins très performantes, dans des temps raisonnables.

3.1.3. Problèmes induits par le test multi cœurs

Nous avons vu que la parallélisation du test des cœurs d'un système peut réduire de manière significative le temps de test du système et donc le coût du test de ce système. Cependant, cela peut rendre la synchronisation de ces différents tests simultanés nettement plus complexe. En effet, chaque canal du bus de test peut être affecté au transport des données de test d'un cœur à un autre à tout moment de manière indépendante. Il s'agit donc de synchroniser chaque ligne du TAM avec chaque cœur, ce qui peut s'avérer coûteux en termes de données supplémentaires et de surface de silicium sur le système.

D'autre part, plus on veut pouvoir tester de cœur en même temps, autrement dit augmenter le parallélisme, plus la taille du bus de test doit augmenter. Or ce bus de test est directement connecté au testeur. La taille du bus (W_{TAM}) est donc limitée par le nombre de canaux disponibles sur le testeur. Il est donc intéressant de pouvoir augmenter le parallélisme des tests des cœurs sans pour cela augmenter les besoins en canaux testeur.

Dans la suite, nous passons en revue quelques méthodes d'optimisation de l'architecture de test basée sur la norme IEEE 1500 ayant pour objectif de réduire le temps de test. Nous proposons ensuite une méthode orientée dans ce même sens utilisant la compression des données de test.

3.2. Etat de l'art

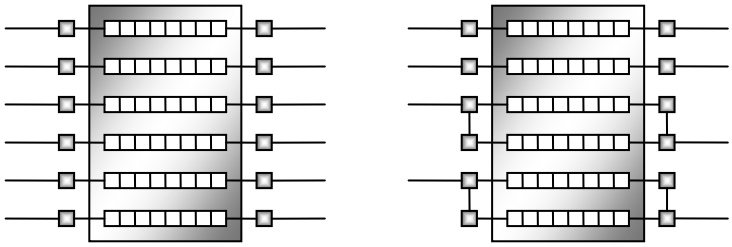
3.2.1. Méthode d'optimisation

Au sein de la norme IEEE 1500, à partir de la structure générale TAM/wrapper, l'intégrateur système a la charge de concevoir les détails de l'architecture de test du système sur puce. En effet, le temps de test du système dépend entre autre de l'architecture du bus de test TAM. En ce qui concerne le temps de test d'un cœur, il va lui dépendre entre autre de la structure de son wrapper. Deux situation sont envisageables pour l'implantation du wrapper : soit le cœur est fournit avec un wrapper prêt à l'emploi, auquel cas l'intégrateur ne peut pas le modifier, soit le cœur est nu, et c'est à l'intégrateur de le concevoir. La conception du bus de test est, elle, complètement à la charge de ce dernier. Il doit en déterminer la taille est les connexions avec les cœurs (ou plutôt avec les wrappers des cœurs). Un rapide aperçu des techniques d'optimisation de wrapper et de TAM susceptibles d'être employées pour réduire le temps de test du système est donné dans cette section.

3.2.1.1.1. Conception des wrappers

Rappelons que le wrapper est l'interface reliant les entrées/sorties de test du circuit aux canaux du TAM auquel il est relié. Si l'on parallélise toutes les entrées/sorties de test, c'est à dire que

chaque entrée ou sortie de test du cœur est reliée à un bit du TAM, le temps de test est minimal car on applique une slice de test à chaque coup d'horloge comme illustré sur la Figure 3.6 (a).



(a) 12 canaux TAM (b) 8 canaux TAM

Figure 3.6 : Conception du wrapper

Cependant, il est possible de réduire le nombre de canaux pour connecter un cœur au TAM. Dans cette situation, les cellules du wrapper de ce cœur doivent être mises en série comme illustré sur la Figure 3.6 (b). Cette sérialisation s'accompagne inévitablement d'une augmentation du temps de test du cœur puisque plus de cycles sont nécessaires pour charger complètement une slice (2 fois plus sur l'exemple). En contrepartie, un effet second est d'augmenter les possibilités de parallélisation des tests au niveau système puisque moins de canaux TAM sont nécessaires pour fournir au cœur les données de test.

Dans [Chak01], les auteurs proposent une méthode qui, en fonction du nombre de canaux TAM disponibles pour connecter un cœur, propose une architecture du wrapper correspondant au temps de test le plus faible pour ce cœur. Cette architecture vise à équilibrer au mieux les chaînes de scan en les sérialisant si nécessaire. En effet, le temps de test d'un cœur est en majeure partie déterminé par la plus longue chaîne de scan. Cependant cette approche ne consiste qu'à chaîner les chaînes de scan internes et ne considère pas la possibilité de chaîner également les cellules du wrapper. Une autre méthode a été proposée dans [Mari00].Elle présente l'avantage par rapport à la précédente de pouvoir chaîner aussi les cellules du wrapper.

Ces méthodes permettent de réduire le temps de test d'un cœur du système pour un nombre de canaux TAM donné. Pour un cœur i, ce nombre W_{TAM_i} sera appelé dans la suite "taille du wrapper" puisqu'il fait référence aux nombre de connexion du wrapper sur le TAM. Le temps de test t_i d'un cœur i est donc une fonction discrète décroissante de ce nombre : $t_i = f(W_{TAM_i})$.

3.2.1.1.2. Conception du TAM

Une fois les wrappers définis, qu'ils soient fournis avec le cœur ou conçus par l'intégrateur système, l'étape suivante consiste à définir l'architecture du bus de test TAM. Les 2 principales variables qui doivent être déterminées lors de cette étape sont la taille du bus (le nombre de canaux du TAM, sachant que chaque canal TAM requiert de la surface silicium sur le système) et la façon dont sont connectés les wrappers à ces canaux. La recherche de solution peut être orientée selon plusieurs axes comme la minimisation de la surface additionnelle de silicium par exemple, ou ce qui nous intéresse plus particulièrement ici, la réduction du temps de test total.

Dans [Chak01] les auteurs proposent une méthode orientée sur la co-optimisation surface/temps de test. Dans cette approche, on considère que le bus de test est divisé en plusieurs sous-bus, un cœur n'étant connecté qu'à un seul sous-bus. Les auteurs proposent un algorithme basé sur une modélisation ILP (Integer Linear Programming) permettant de trouver un compromis entre temps de test et surface additionnelle avec un temps de calcul raisonnable. Toutefois, la contrainte limitant l'exploration des solutions aux seules architectures à base de bus partitionnés peut empêcher la découverte d'une meilleure solution.

Une autre solution a été proposée dans [Poug02]. Une modélisation mathématique du problème à base d'hypergraphe d'incompatibilité afin de pouvoir appliquer une méthode exhaustive qui permet de trouver la solution optimale pour l'implantation de ce bus de test sur le système. Cette méthode permet d'obtenir en sortie l'assignation des cœurs au bus de test, le temps de test minimal du système et la largeur utile minimum pour le bus global de test. Le nombre de solutions parcourues est réduit par l'insertion de conditions en divers points de l'algorithme ce qui permet d'économiser du temps CPU dans des boucles. Les solutions équivalentes et les solutions dont le temps de test total est prohibitif ne sont donc pas générées. Cependant, dans le cas de système contenant beaucoup de cœurs, cette méthode montre ses limites et le temps de calcul devient prohibitif. Les solutions trouvées ne sont donc pas optimales car elles sont intermédiaires.

3.2.2. Introduction de la compression

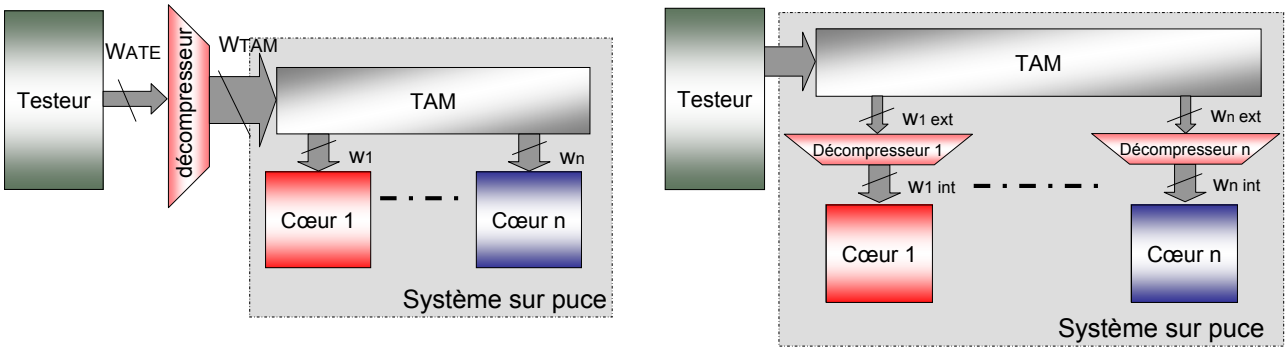
Nous venons de voir que l'on peut réduire le temps de test du système complet simplement en optimisant la conception de toute l'infrastructure de test (wrappers et TAM) dans ce sens. On a également vu que la réduction du temps de test du système passe principalement par une augmentation du parallélisme du test des cœurs qui le compose. Il est donc intéressant de trouver un moyen d'augmenter ce parallélisme sans pour autant augmenter la taille du bus de test, ce qui serait non seulement couteux en termes de surface de silicium, mais également en termes de canaux testeur additionnels.

Pour ce faire, quelques méthodes utilisant la compression des données de test ont été publiées dans la littérature. On y trouve un certains nombre d'approches utilisant des techniques de compressions de flux des données (compression verticale) [Chan03] [Gonc03] [Lars07] afin de réduire la profondeur mémoire nécessaire sur le testeur ou sur la diffusion d'un même vecteur de test sur plusieurs cœurs [Lars05]. Or nous avons vu dans la section 2.1.2 les problèmes liés à la synchronisation des données de test, difficultés qui vont être amplifiées dans le cadre des systèmes sur puce.

D'autre part, des solutions basées la réduction des besoins en canaux testeurs d'un cœur (compression horizontale) ont été publiées dans [Iyen03] [Gonc04] [Gonc05]. Cependant, ces méthodes sont basées sur des techniques de décompression nécessitant un outil de test spécifique, ou du moins reposant sur la connaissance des netlists des cœurs à tester. Or cette condition n'est que rarement vérifiée dans le cadre des systèmes sur puces. De plus, l'ensemble des solutions

architecturales n'est pas exploré puisque ces approches se concentrent sur des structures figées du type :

- un décompresseur en tête du système qui fournit les données de test à tous les cœurs (Figure 3.7 (a) comme par exemple dans [Lars08])
- un décompresseur pour chaque cœur du système, ou au mieux pour plusieurs récurrences du même cœur (Figure 3.7 (b) comme par exemple dans [Wang07])



(a) 1 seul décompresseur en tête (b) 1 décompresseur par cœur

Figure 3.7 : Positions des décompresseurs

Nous présentons dans la suite une solution intégrant une technique de compression horizontale dans la recherche d'architecture de test amenant un temps de test du système complet minimal. Cependant, la compression rajoute au problème de recherche d'architecture des variables liées à l'insertion de décompresseur dans la chaîne de transmission des données de test. Pour parvenir aux meilleures solutions, la compression ne doit pas avoir été prédéfinie préalablement à la recherche d'architecture, mais doit être complètement adaptée et intégrée lors de cette recherche d'architecture de test.

3.3. Méthode proposée

3.3.1. Présentation

L'objectif de la méthode proposée est d'augmenter la bande passante du TAM grâce à la compression de données de test sans augmenter les coûts en équipement de test (canaux testeur, profondeur mémoire…). Pour ce faire, nous proposons d'utiliser une technique de compression horizontale adaptée aux systèmes et compatible avec le test des IP. Notons ici que n'importe quelle technique de compaction des réponses telles que [Mitr02] ou [Rajs05] peut être librement utilisée en sortie du système. Nous ne traiterons pas de cette partie dans cette étude.

3.3.1.1. Compression et temps de test

Comme cela a été dit dans le Chapitre 2: dans le cadre de la compression horizontale, et ce quelque soit la technique employée, il n'est pas toujours possible de compresser un vecteur de test. La conséquence est que, si on veut conserver le taux de couverture de la séquence de test, les

vecteurs non compressibles doivent être transmis au cœur par une autre voie, typiquement de façon série ce qui entraîne une augmentation du temps de test du cœur En résumé, pour un nombre de chaînes de scan donné (ou une taille de wrapper donnée dans le cas qui nous intéresse ici), le temps de test d'un cœur augmente à mesure que le nombre d'entrées du décompresseur diminue. Ce phénomène peut être vérifié dans le Tableau 2.5 où, pour le circuit ISCAS'89 s5378 [Benc89], pour 16 chaînes de scan (N sur le tableau), on passe d'un temps de test de 4025 cycles pour 8 entrées décompresseur (M sur le tableau) à un temps de 8821 cycles pour seulement 2 entrées. La compression horizontale permet donc dans cette situation de réduire le nombre de canaux nécessaires au transfert des données de test vers un cœur mais au prix d'une légère augmentation du temps de test de ce cœur. Il faut donc s'assurer que l'augmentation du parallélisme induite par la compression permette de contrebalancer l'augmentation du temps de test des cœurs individuels afin d'atteindre l'objectif de réduction du temps de test du système complet qui nous intéresse ici.

3.3.1.2. Compression et test en parallèle

Lors du test simultanée de plusieurs cœurs, les données de test des différents cœurs sont concaténées en transmises aux cœurs via le bus de test. L'ajout de la compression dans ce cadre implique d'utiliser le même décompresseur pour tous les cœurs. Ceci demande une attention particulière lors de la recherche d'une architecture minimisant le temps de test du système.

Par l'exemple de la Figure 3.8, pour deux cœurs C1 et C2 testés en parallèle sans compression, possédant chacun deux chaînes de scan, la séquence de test T12 est composée de la concaténation des deux séquences de test T1 et T2 des cœurs. Avec une compression de rapport ½ (1 canal testeur pour 2 chaînes de scan), la séquence T1 est compressée en T1* qui nécessite 5 cycles testeur : 2 cycles pour l'initialisation avec la première slice chargée en série et 3 cycles pour charger les 3 différences arithmétiques constituant les vecteurs compressés. La séquence T2 est compressée en T2* qui, elle, nécessite 7 cycles : les 3 premières slices n'étant pas compressibles (il n'existe pas de différences arithmétique entre les slices), ces 3 slices sont chargées en séries en 6 cycles et la dernière slice est chargée avec sa slice compressée correspondante en 1 cycle. Dans le cas du test simultanée des deux cœurs C1 et C2 avec un même décompresseur, la séquence T12* a donc besoin de 7 cycles testeur pour être envoyée aux deux cœurs.

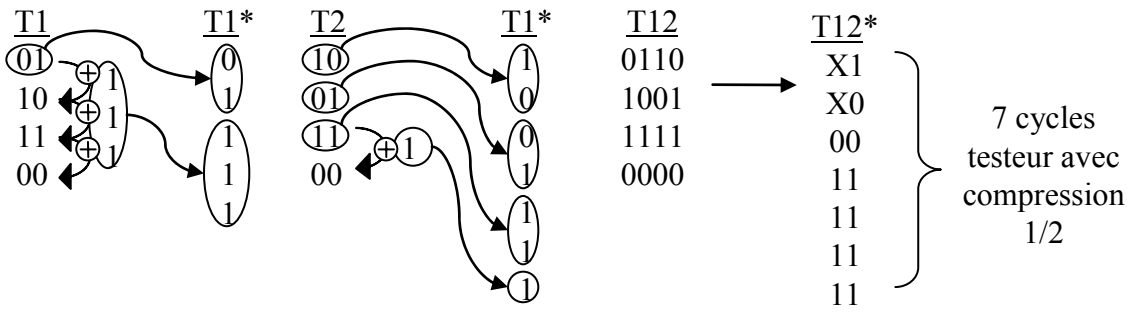


Figure 3.8 : Concaténation de 2 séquences de test T1 et T2 pour le test en parallèle

Sur un autre exemple (Figure 3.9), il s'agit cette fois de tester en parallèle le même cœur C1 mais cette fois avec un autre cœur C3 et avec le même rapport de compression de ½. La séquence T1* seule a toujours besoin de 5 cycles et la séquence T3* requiert 5 cycles également. Le test simultanée des cœurs C1 et C3 avec un même décompresseur nécessitera donc cette fois 5 cycles testeurs.

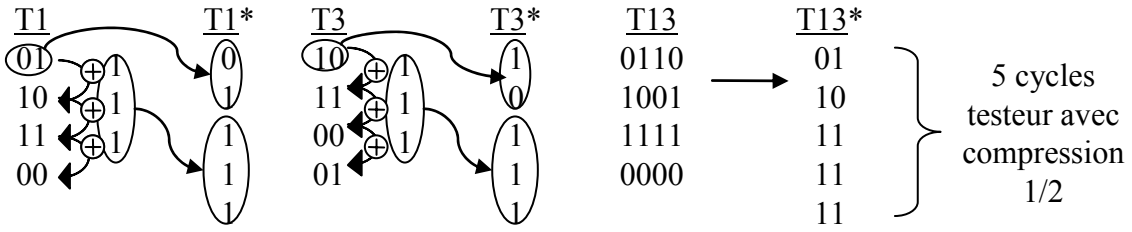


Figure 3.9 : Concaténation de 2 séquences de test T1 et T3 pour le test en parallèle

On voit donc que le temps de test de C1 est de 7 ou 5 cycles selon le cœur qui est testé en même temps que lui. L'utilisation d'un même décompresseur pour deux cœurs testés en parallèle pose donc le problème de la détermination du temps de test de ces cœurs et donc de l'évaluation du temps de test complet du système. Pour connaître le temps de test total, il est nécessaire de connaître le temps de test de chaque cœur, et pour connaître le temps de test d'un cœur on a besoin de savoir quels cœurs sont testés en parallèle ce qui relève de l'ordonnancement. C'est un cercle vicieux qui empêche tout algorithme de pouvoir obtenir un ordonnancement optimal en termes de temps de test.

Une autre façon de procéder pour tester plusieurs cœurs en même temps avec un même décompresseur serait de réaliser l'assignation des cœurs au bus de test avec l'ordonnancement des tests associé sans tenir compte de la compression, puis de compresser la séquence ainsi obtenue à la fin de la procédure. Mais cette méthode ne peut prétendre à l'obtention d'un temps de test total optimal puisque l'ordonnancement est obtenu à partir du temps de test des séquences non compressées qui sont, comme on l'a vu plus haut, différents des temps de test avec la compression.

En conclusion, un algorithme d'ordonnancement ne peut pas être établi que si les cœurs alimentés par un même décompresseur sont testés en série.

3.3.1.3. Compression et architecture de test

Nous venons de voir dans les deux sections précédentes que l'utilisation de la compression implique, d'une part, que le temps de test d'un cœur dépend du rapport de compression utilisé et d'autre part, que l'utilisation d'un seul décompresseur pour alimenter plusieurs cœurs testés en même temps n'est pas compatible avec l'utilisation d'un algorithme d'ordonnancement visant à réduire le temps de test du système.

Ces remarques imposent des contraintes quant au type d'architecture à choisir et sur la façon d'explorer l'ensemble des solutions. En effet, il apparaît évident que le ou les rapports de

compression ne peuvent être établis ni avant l'exécution de l'ordonnancement du test du système puisqu'un grand nombre de solutions ne seraient pas explorées, ni après puisque la recherche serait faussée par des temps de test des cœurs erronés. Le rapport de compression devient dès lors une variable à part entière de l'algorithme qui devra déterminer l'architecture de test et l'ordonnancement qui y sera associé.

D'autre part, l'impossibilité de tester en parallèle sur un même décompresseur implique l'utilisation de plusieurs décompresseurs sur des sous-bus indépendants. Ces sous-bus peuvent évidemment alimenter plusieurs cœurs, mais sur chacun de ces sous-bus, les cœurs sont testés en série. Le temps de test de chaque bus est donc la somme des temps de test individuels des cœurs qui y sont assignés. Le temps de test total du système est par conséquent le plus grand temps de test des sous-bus.

Sur l'exemple de la Figure 3.10, le TAM est scindé en trois sous-bus de test indépendants (TAM_1, TAM_2 et TAM_3), chacun alimenté par un décompresseur. De la même manière, l'ensemble des canaux testeur va être découpé en 3 voies (ATE_1, ATE_2 et ATE_3). Sur chaque sous-bus peuvent se trouver plusieurs cœurs testés en série, mais les trois sous-bus fonctionnent en parallèle. Dans la suite, nous appelons ligne de test i l'ensemble {sous-bus TAM_i, sous ensemble canaux testeur ATE_i, décompresseur dont le rapport est défini par W_{ATE_i} / W_{TAM_i}}.

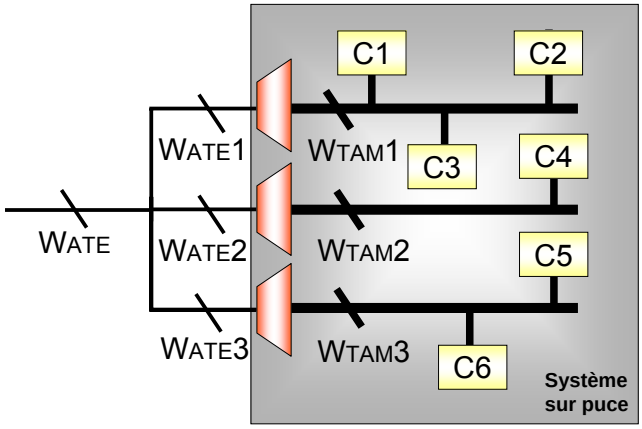


Figure 3.10 : Exemple d'architecture de test avec compression

3.3.2. Problème et notations

La recherche de l'architecture de test optimale et l'ordonnancement de test associé peut être formulée comme un problème d'optimisation. Les données d'entrée sont le nombre de canaux ATE disponibles sur le testeur, la largeur maximale du bus de test TAM intégrable sur le système, ainsi que la liste des cœurs implantés dans le système. Chaque cœur est supposé fourni avec sa propre séquence de test. L'objectif est de déterminer la meilleure partition de l'architecture en un nombre p de sous-bus à déterminer ainsi que les connexions des cœurs à ces sous-bus afin de minimiser le temps de test total du système.

Dans ce chapitre, nous utilisons les notations suivantes : soient n le nombre de cœurs du système, w_j la taille du wrapper du cœur j (en réalité, le nombre de bits requis pour la connexion du cœur n sur le TAM), p est le nombre de ligne (nombre de sous-bus et nombre de décompresseurs). W_{ATE_i} et W_{TAM_i} pour i variant de 1 à p représentent respectivement le nombre de canaux ATE et la taille du TAM de la ligne i et $r_i = W_{ATE_i} / W_{TAM_i}$ est le rapport de compression de la ligne i. Le temps de test d'un cœur j avec un wrapper de w_j pour un rapport de compression r_i est noté $t^j_{wj,ri}$, et le temps total de test du système est noté TAT (pout Temps d'application du Test).

Le problème consiste à déterminer (l'ensemble des variables à déterminer sont présentées sur la Figure 3.11) :

- Le nombre de ligne p
- Les dimensions W_{ATE_i} et W_{TAM_i} pour chaque ligne i variant de 1 à p.
- L'assignation des cœurs aux lignes
- La taille du wrapper wc_j de chaque cœur[1]
- Un ordonnancement du test avec un temps de test total minimal

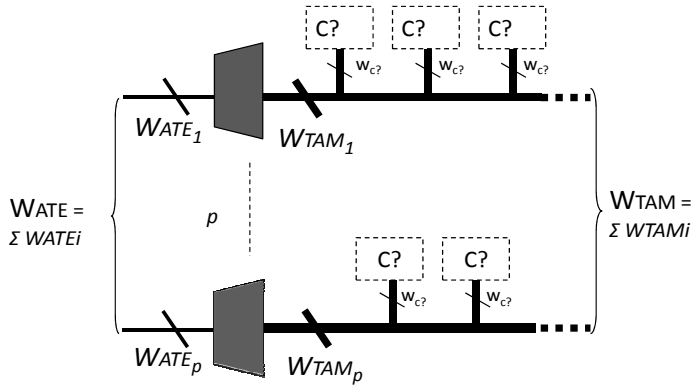


Figure 3.11 : Architecture vise et variables à déterminer

Les contraintes suivantes doivent être respectées :

$$W_{ATE} = \sum_{i=1,..,p} W_{ATE_i}$$ Contrainte 3.1

$$W_{TAM} \geq \sum_{i=1,..,p} W_{TAM_i}$$ Contrainte 3.2

$W_{TAM_i} \geq W_{ATE_i}$ pour i de 1 à p Contrainte 3.3

$wc_j \leq W_{TAM_i}$ si le cœur j est connecté à la ligne i. Contrainte 3.4

3.3.3. Algorithmes

[1] En ce qui concerne les wrappers, deux situations peuvent se présenter: soit les cœurs sont fournis avec leur wrapper auquel cas wc_n n'est plus à déterminer, soit les cœurs sont donnés bruts et c'est au systémier de concevoir les wrappers. Dans ce cas, il faut noter que la valeur de wc_j est comprise entre $1 \leq wc_j \leq \max(\#PIs,\#POs) + \#SCs$ (avec #PIs = nombre d'entrées primaires, #POs = nombre de sorties primaires et #SCs = le nombre de chaînes de scan du cœur). Nous ne reviendrons pas sur le design du wrappers qui a déjà été traité dans la section 3.2.1.1.1.

L'algorithme général de recherche d'architecture est donnée Figure 3.12. Il se compose principalement de deux boucles imbriquées.

1. Pour toute les partitions valides ATE / TAM
2. Définir l'assignation des n cœurs au p lignes qui minimise le temps de test total du système pour cette configuration ATE / TAM.
   - Si le temps de test total obtenu est meilleur, mémoriser l'assignation des cœurs et l'architecture ATE / TAM correspondante

Figure 3.12 : Algorithme général de recherche d'architecture de test

La première boucle (ligne 1) consiste à parcourir l'ensemble des partitions valides de l'ensemble bus ATE / bus TAM en p lignes. Une partition est dite valide si pour toutes les i lignes, $W_{TAM_i} \geq W_{ATE_i} > 0$. La deuxième partie (ligne 2) est une recherche d'assignation/ordonnancement des cœurs sur l'une des lignes de test pour une configuration ATE / TAM donnée afin de minimiser le temps de test total du système. Lors du parcours des configurations valides, à chaque fois qu'une architecture réduit le temps de test total du système, elle est mémorisée. Les deux parties de cet algorithme sont décrites plus en détail dans la suite.

3.3.3.1. Temps de test des cœurs

Les temps de test $t^j_{wj,ri}$ de chaque cœur, pour chaque rapport de compression r_i et pour chaque taille de wrapper w_j sont des données d'entrées de l'algorithme. Ces données sont nécessaires au calcul du temps de test total du système TAT dont nous rappelons la formule dans l'équation 3.1 :

$$TAT = \max\left(TAT_i, i = 1..p\right)$$

$$\text{avec} TAT_i = \sum_{\text{cœurs sur la ligne i}} t^j_{w_j, r_i} \qquad \text{Équation 3.1}$$

La connaissance de ces temps de test est donc nécessaire avant d'effectuer l'ordonnancement du test du système. Or pour connaître ces temps, il faut utiliser l'algorithme de compression pour chaque cœur, pour chaque rapport de compression, et pour chaque taille de wrapper. Ceci peut se révéler très coûteux en termes de temps de calcul selon la technique de compression utilisée. Nous proposons ici une alternative au calcul exhaustif de toutes ces valeurs.

Tout d'abord, si les wrappers ne sont pas fournis avec les cœurs, rappelons que, en l'absence de compression (rapport de compression $r_i = 1$), le temps de test d'un cœur est une fonction décroissante en palier de la taille du wrapper. De plus ce temps dépend plus du nombre de vecteurs de test que des vecteurs eux-mêmes. Le temps de test en fonction de la taille du wrapper pour le dixième cœur du système de référence D695 tiré de ITC'02 [Benc02] est donné Figure 3.13). De manière générale, le nombre de paliers est assez restreint (15 paliers pour ce cœur pour 64 taille de wrapper). Il n'est donc pas nécessaire de connaître les temps pour chaque taille de wrapper mais

uniquement pour chaque palier, ce qui réduit le nombre de configurations à couvrir lors de l'ordonnancement.

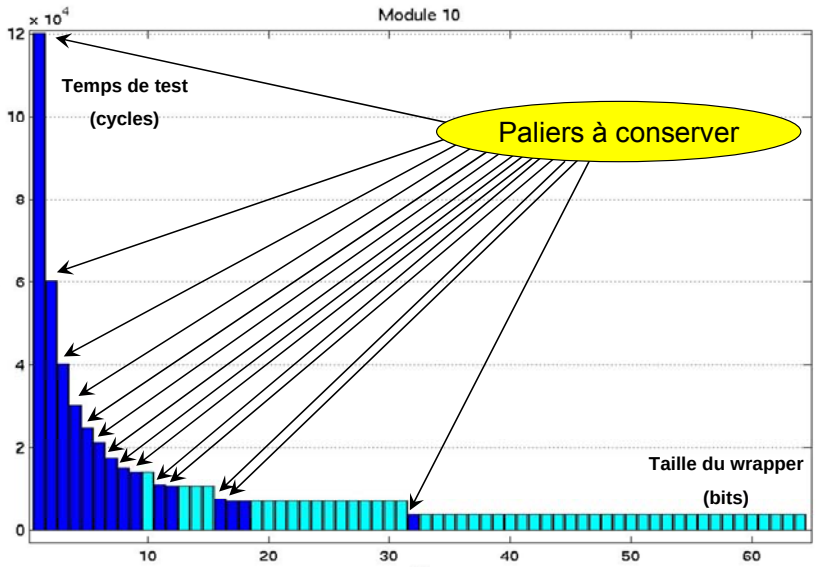


Figure 3.13 : Temps de test vs. taille du wrapper du 10ème cœur du système D695 (ITC'02)

Deuxièmement, quelle que soit la technique de compression utilisée, on peut observer le même comportement du temps de test en fonction du rapport de compression r (pour un nombre de chaînes de scan ou une taille de wrapper w_j fixé). Le temps de test d'un cœur peut être approximé par la fonction suivante :

$$t^{j}_{w_j,r} = \frac{a}{r} + b \qquad \text{Équation 3.2}$$

Deux valeurs du temps de test sont suffisantes pour identifier les coefficients a et b. On peut donc obtenir une estimation des temps de test $t^{j}_{wj,r}$ pour tous les rapports de compression à partir de seulement deux valeurs calculées. Il n'est donc nécessaire d'effectuer la compression pour calculer le temps de test d'un cœur que pour deux valeurs du rapport de compression r. Pour plus de précision, il faut choisir deux pas de calcul éloignés l'un de l'autre. On voit sur l'exemple donné Figure 3.14 que les deux dernières valeurs de calculs ne sont pas prises en comptes dans l'estimation des temps de calculs. En effet lorsque la taille du wrapper est égale au nombre de bits du TAM (pas de compression ⇔ $w_j = W_{TAM_i}$) et pour $w_j = W_{TAM_i}-1$, toutes les slices sont compressibles (1slices => 1 cycle d'horloge). Or cette situation peut fausser l'identification des temps de test.

On choisit donc les valeurs de $1/w_j$ (1 bit TAM pour w_j chaînes de scan) et $(w_j-2)/ w_j$ (2 entrées du décompresseur en moins par rapport au nombre de chaînes de scan) pour réaliser l'estimation des temps de test. Cette propriété a été vérifiée avec la technique de compression décrite dans le Chapitre 2: Sur la Figure 3.14, les valeurs calculées et celles estimées sont reportées pour le circuit s38417 des circuits de références ISCAS'89 [Benc89] pour 16 chaines de scan (autrement dit pour un wrapper de 16 bits). L'erreur maximum entre les valeurs calculées et celles

estimées est inférieure à 1%. Des résultats similaires ont été obtenus pour tous les circuits ISCAS'89 pour différentes tailles de wrappers.

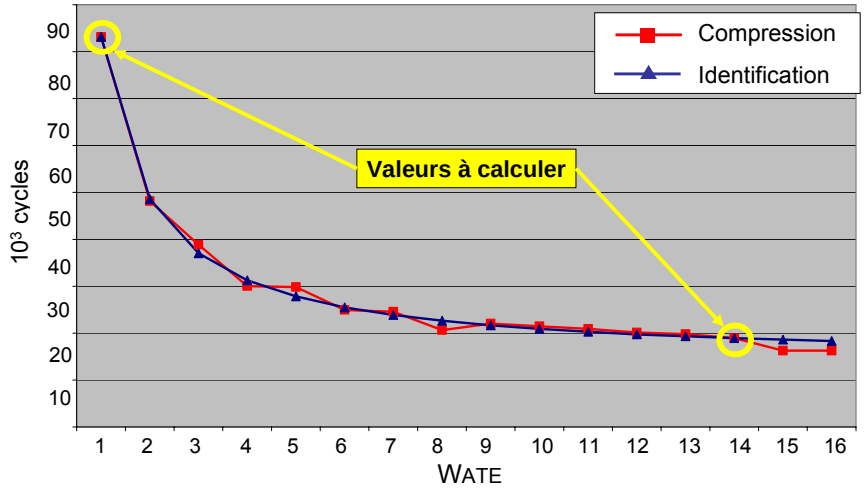


Figure 3.14 : Comparaison entre calcul et estimation du temps de test du circuit S38417 (ISCAS'89) pour un wrappers de 16 bits

3.3.3.2. Exploration des partitions

Les décompositions valides d'un bus en sous-bus, du coté ATE et du coté TAM sont calculées en tenant compte de certaines contraintes (ligne 1 de la Figure 3.12) :

- la taille d'un bus ne peut pas être nulle
- la taille d'un bus ATE est inférieure ou égale à la taille du bus TAM correspondant (une égalité correspond à une absence de décompresseur)
- l'ordre n'a pas d'importance (ex : **3-1**-2/**6-2**-4 est équivalent à **1-3**-2/**2-6**-4)

L'algorithme de recherche des partitions est finalement assez simple. Il s'agit de trouver l'ensemble des découpages possibles du bus ATE en p parties puis, pour chaque découpage, trouver l'ensemble des découpages possibles du bus TAM satisfaisant les contraintes ci-dessus. Un exemple de paire de découpage ATE et TAM est donné Figure 3.15.

Le découpage des canaux en p parties peut être facilement déterminé par l'application de la formule de partition d'un nombre entier. Le nombre X(n,p) de partition d'un ensemble en p partie est donné par :

$$X(n,p)=\sum_{k=1}^{p} X(n-p,k) \text{ with } X(n,n)=X(n,1)=1 \quad \text{and } X(n,p)=0 \text{ si } p>n$$

Équation 3.3

Par exemple, 10 canaux ATE peuvent être découpé en 3 parties de X(10,3)=8 façon différentes (1+1+8, 1+2+7, 1+3+6, etc…).

Ensuite, pour chaque découpage du bus ATE, on cherche tous les découpages compatibles possibles pour le bus TAM. Un découpage compatible du TAM est un découpage qui respecte les contraintes 2 et 3 soit $W_{TAM} \geq \sum_{i=1,..,p} W_{TAM_i}$ et $W_{TAM_i} \geq W_{ATE_i}$ pour toutes les lignes.

Un exemple d'architecture testeur / bus de test est donné sur la Figure 3.15. Pour une architecture à 3 lignes, les 6 canaux testeurs sont découpés en 3 sous ensembles de 3, 1 et 2 canaux et les 12 voies du bus de test TAM sont découpées en 3 sous-bus de 6, 2 et 4 bits. Cette architecture contient donc 3 décompresseurs dont les rapports de décompression sont de 3/5, 1/3 et 1/2.

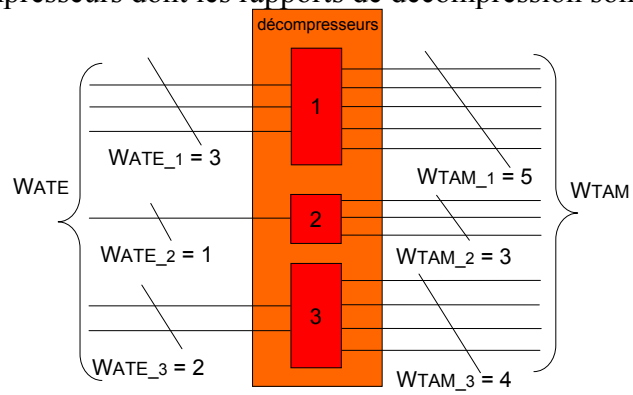


Figure 3.15 : Exemple de découpage ATE / TAM

Notons que si le wrapper est fourni avec le cœur et donc non modifiable, la contrainte 4 peut réduire l'ensemble des découpages à retenir comme valide. En effet, le plus petit sous-bus TAM doit être suffisamment grand pour que le plus petit wrapper puisse y être connecté. De même, il doit exister un sous-bus TAM assez grand pour accueillir le cœur avec le wrapper le plus grand On remarque aussi que dans le cas où $W_{TAM_i} = W_{ATE_i}$, le taux de compression est égal à 1 et donc cette ligne ne comporte pas de décompresseur.

3.3.3.3. Assignation des cœurs et Ordonnancement

Pour chaque paire de découpage ATE / TAM, les cœurs doivent êtres assignés sur les sous-bus TAM et un ordonnancement effectué afin d'obtenir le temps de test total minimum de cette configuration (ligne 2 de la Figure 3.12). L'ordonnancement des tests et l'assignation des cœurs doivent être menés conjointement pour obtenir une architecture optimale en termes de temps de test total du système. Malheureusement, la recherche de l'ensemble assignation/ordonnancement optimal est un problème NP complet. Nous proposons ici une heuristique permettant d'obtenir une solution performante. L'algorithme est donné Figure 3.16.

La première étape détermine une solution initiale de l'architecture, c'est à dire une assignation initiale des cœurs au sous-bus. Il faut distinguer deux cas : soit le wrapper est conçu et fixé lors de la conception du cœur, soit c'est au systémier de le concevoir et la taille du wrapper est un degré de liberté de plus. Les cœurs successifs sont placés de telle sorte que le temps de test total du système soit le plus petit à chaque itération.

Dans le premier cas (wrapper fixé), les cœurs sont positionnés sur le plus petit sous-bus TAM possible pouvant l'accueillir (contrainte 3.4), c'est à dire, sur le sous-bus où le maximum de bits du TAM est utilisé. Par exemple, dans un cas avec 3 sous-bus TAM, chacun respectivement de 5, 7 et 10 bits, un cœur dont le wrapper est préalablement fixé à 6 bits sera assigné au sous-bus TAM de 7 bits. En effet, le premier sous-bus TAM de 5 bits est trop petit pour y connecter le cœur (contrainte 3.4). Le deuxième bus est préféré au troisième puisque, a priori, il est avantageux de réserver les bus les plus larges aux cœurs avec les plus gros wrappers.

Dans le second cas (wrapper libre), chaque cœur est positionné sur le sous-bus TAM le plus large et son wrapper est défini d'après la contrainte 4.

La seconde étape est la partie principale de l'algorithme. Il s'agit d'améliorer cette solution initiale. Pour cela, les cœurs sont déplacés d'un sous-bus TAM à l'autre avec pour objectif la réduction du temps de test total du système. L'idée est de retirer un cœur de la ligne dont le temps de test est le plus grand de façon a réduire le plus possible le temps de test total. Ainsi, tous les cœurs de la ligne considérée sont virtuellement déplacés dur les autres lignes. A chaque fois le temps de test total est recalculé. Le déplacement qui réduit le plus le temps de test est conservé. En cas d'égalité, l'algorithme choisi le déplacement (cœur j sur ligne i) qui minimise le nombre de bits inutilisé sur un sous-bus TAM. ($W_{TAM_i} - w_j$ est minimal), ceci afin de se laisser plus de liberté pour déplacer les cœurs avec les wrappers les plus large (dans le cas de wrappers prédéfini) sur les sous-bus les plus larges dans les pas suivants de l'algorithme.

2.1 // Solution initiale
- Trier les cœurs par volume des données de test décroissant
- Assigner chaque cœur à un sous bus tel que le temps de test total augmente le moins possible.

2.2 // Amélioration de la solution
- Tant que le temps total est réduit
  - Trouver la ligne i avec le temps le plus grand
  - Pour chaque cœur j de la ligne i,
    - Pour toutes les autres ligne k ($k \neq i$)
      - Déplacer le cœur c de la ligne i à la ligne k
      - Si le temps total est réduit, calculer le nouveau temps et mémoriser i, k, j et le nouveau temps
      - Ramener le cœur j sur la ligne k
  - Déplacer le cœur c de la ligne i à la ligne k tel que:
    1) le temps le plus petit a été obtenu
    2) le nombre de bits inutilisés sur k est minimisé

Figure 3.16 : algorithme d'assignation des cœurs et d'ordonnancement du test

Cet algorithme ne tient pas comptes de contraintes telles que la consommation électrique, la précédence des tests, etc… Cependant, ces contraintes peuvent aisément être intégrées. En ce qui concerne la consommation, deux niveaux d'optimisation peuvent être envisagées. D'une part, au

niveau de la compression des données de test du cœur, les bits non assignés (X) lors de la compression peuvent être assignés de manière à limiter la consommation lors du décalage des vecteurs dans les chaînes de scan. D'autre part, lors de l'ordonnancement, l'ordre dans lequel les cœurs assignés sur un même bus sont testés n'est pour le moment pas questionné. Sur l'exemple de la Figure 3.10, les cœurs de la première ligne peuvent être testé dans six ordres différents (C1-C2-C3, C1-C3-C2, etc…). L'ordre du test peut être déterminé par des contraintes de consommation avec lors du test en parallèle des cœurs des autres lignes (C4, et C5/C6)

3.3.3.4. Illustration

Soit un système contenant 16 cœurs dont les wrappers sont fixés et une architecture de test de 5 lignes. A partir d'une solution initiale (Figure 3.17), l'algorithme déplace les cœurs d'une ligne à l'autre pour atteindre au final, la solution avec le temps de test total du système le plus petit possible.

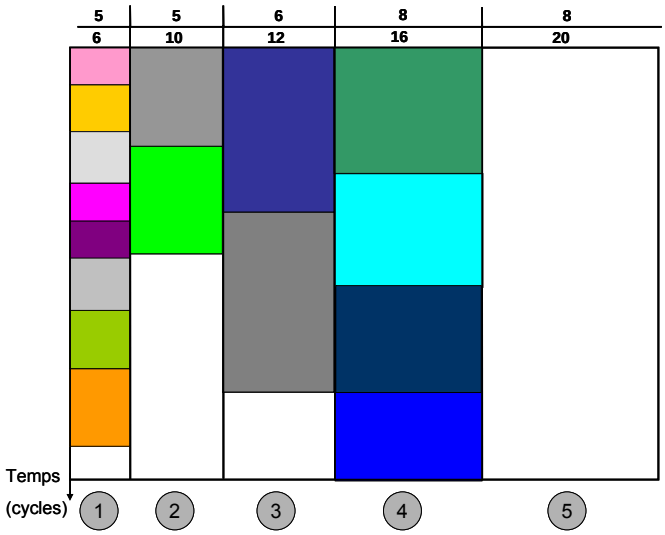


Figure 3.17 : Solution initiale

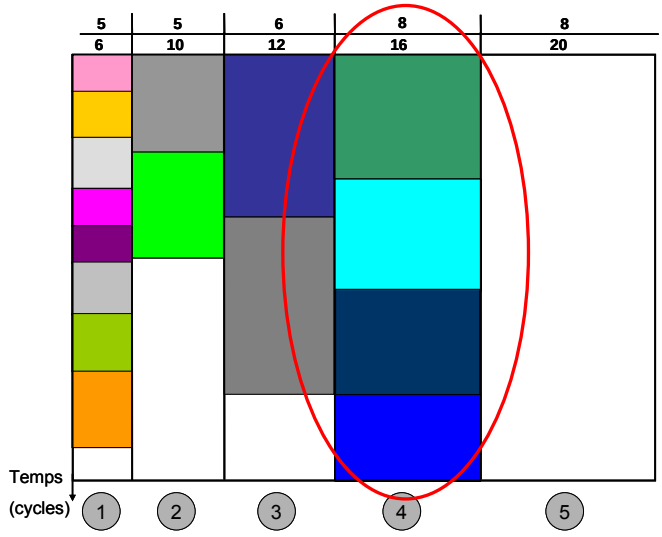


Figure 3.18 : Recherche du bus avec le temps de test le plus grand

A partir de cette solution initiale, l'algorithme cherche le bus dont le temps test des cœurs en série sera le plus long. Sur la Figure 3.18, il s'agit du bus n° 4. L'algorithme déplace successivement tous les cœurs de ce bus sur tous les autres bus, c'est-à-dire sur les bus n° 1, 2, 3 et 5 (Figure 3.19 et Figure 3.20, l'opération est répétée sur les troisième et quatrième cœurs, mais non représenté sur les figures pour ne pas surcharger la lecture).

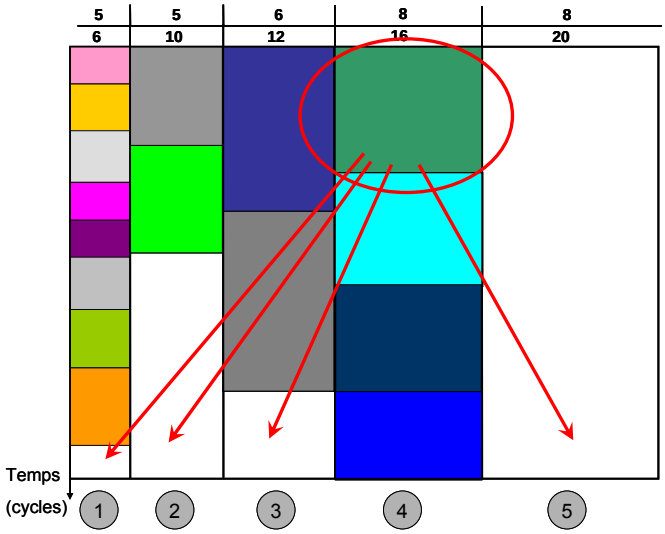


Figure 3.19 : Déplacement virtuel du premier cœur sur les autres bus

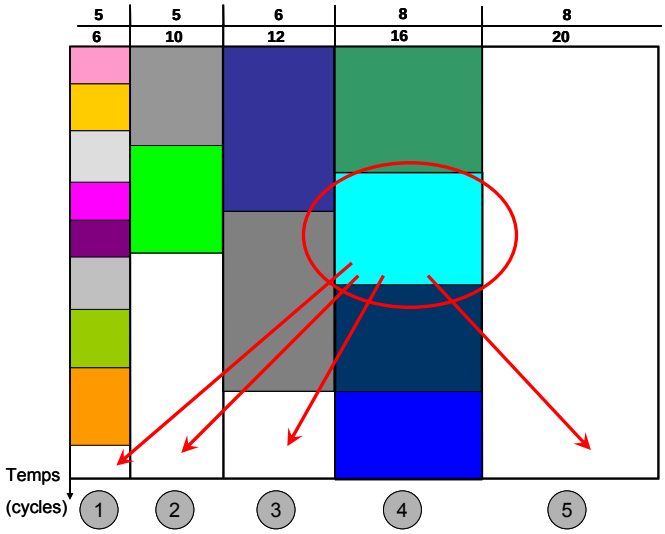


Figure 3.20 : Déplacement virtuel du deuxième cœur sur les autres bus

Une fois que tous les déplacements virtuels ont été effectués et les temps de test de chaque bus calculé pour chaque déplacement, l'algorithme mémorise le choix (cœur+déplacement+temps de test des bus) qui, dans l'ordre, minimise le plus le temps de test total du système, laisse le moins de bits inutilisé sur le bus d'arrivée. Dans notre exemple, sur la Figure 3.21, le quatrième cœur est déplacé de la ligne n°4 à la ligne n° 5.

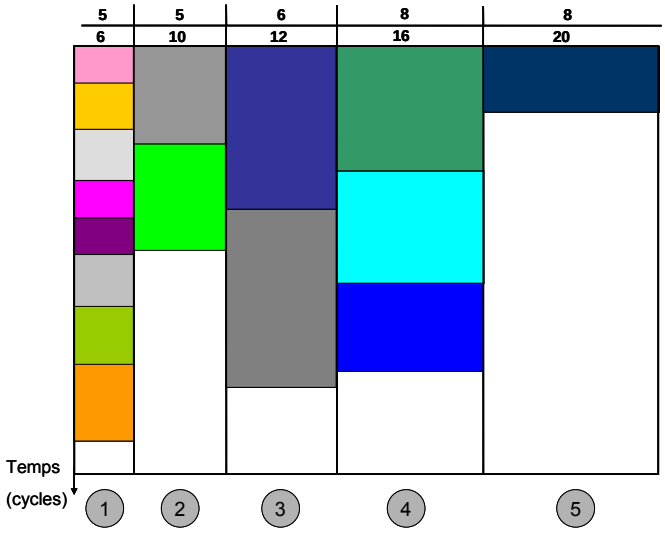


Figure 3.21 : Mémorisation du choix donnant le meilleurs temps de test total

Cette boucle est répétée tant que le temps de test total du système est réduit: recherche du bus dont le temps de test est le plus grand (le bus n°1 sur la Figure 3.22), déplacement virtuel des cœurs de ce bus (Figure 3.23), etc… jusqu'à obtenir la solution finale qui fourni le temps de test total du système le plus petit possible.

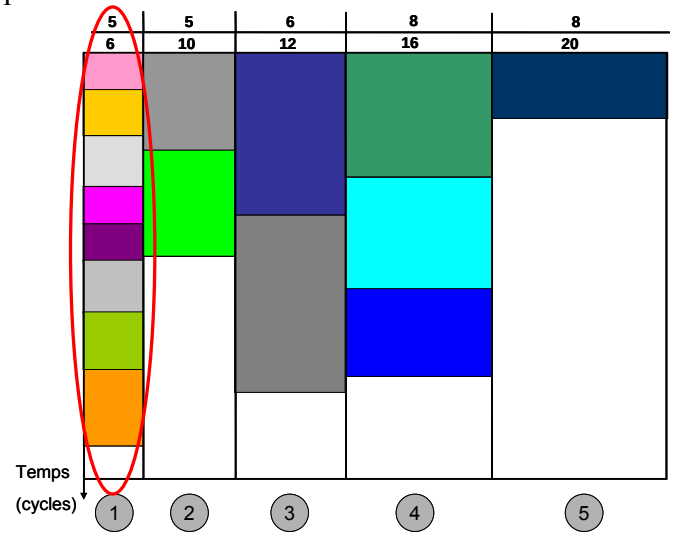


Figure 3.22 : Nouvelle recherche du bus avec le temps de test le plus grand

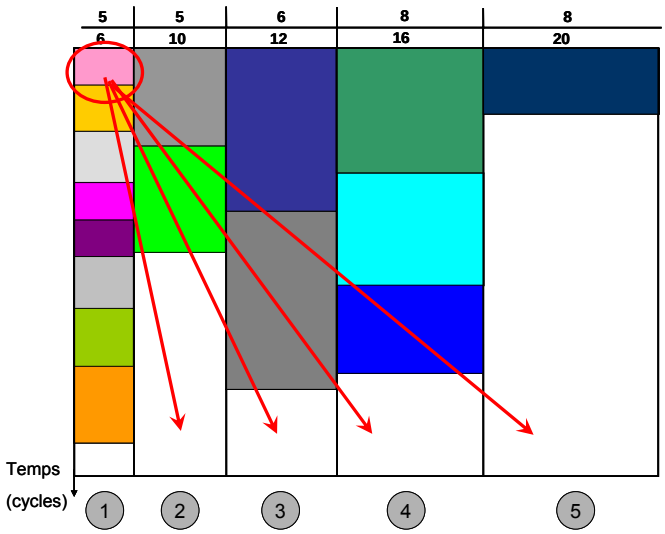


Figure 3.23 : Déplacement virtuel du premier cœur sur les autres bus

3.4. Résultats

3.4.1. Procédure expérimentale

Notre méthode de conception d'architecture de test des systèmes avec compression a été appliquée sur deux exemples de systèmes sur puce. Un premier système, conçu à partir de 16 circuits de références ISCAS'89 [Benc89] a été publié dans [Bayr01] [Iyen03]. Il est présenté sur la Figure 3.24. Les circuits ISCAS'89, dont certains détails sont donnés en Annexe B, sont ici les cœurs du système sur puce et nous assumons ici qu'ils sont fournis avec leur wrappers prédéfinis. Pour chaque cœur, la taille des wrappers est égale au nombre de chaînes de scan de ce cœur.

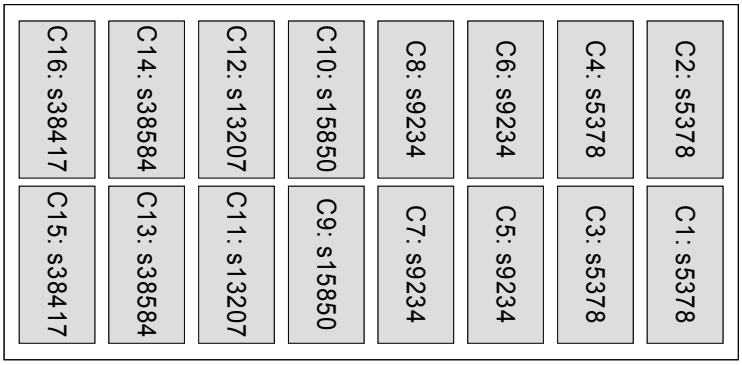


Figure 3.24 : Système n°1 à base de cœurs ISCAS 89

Les séquences de tests des circuits ont été obtenues, comme dans le Chapitre 2: avec le générateur de vecteur TETRAMAX® de Synopsis® [Syno01]. Par soucis de simplicité, la compression de ces séquences a été effectuée avec la technique présentée dans le Chapitre 2, mais rappelons que n'importe quelle méthode satisfaisant les contraintes décrites plus haut sur le test des IP et des systèmes sur puce peut également être utilisée. Les caractéristiques des cœurs avec leurs séquences de test sont données sur le Tableau 3.1.

Cœurs	Nombre de chaînes de scan (w_j)	Temps de test (cycles)
1 à 4	5	9331
5 à 8	6	9030
9, 10	10	8804
11,12	12	16048
13,14	14	19845
15,16	16	45760

Tableau 3.1 : Caractéristiques des cœurs du système n°1

Un deuxième système a été utilisé pour valider notre méthode. Il s'agit du système de références g1023 tiré de ITC'02 [Benc02] dont la description est donnée sur le Tableau 3.2. Ce système comporte 14 cœurs, malheureusement, ni la netlist de ces cœurs, ni leur séquence de test ne sont fournies. Or au moins une de ces deux informations est nécessaire pour effectuer une compression, quelle que soit la technique employée. Pour l'ensemble des systèmes de la suite ITC'02 dont certains détails sont donnés en Annexe B, seul le nombre de vecteurs de test est précisé. Pour pouvoir réaliser des expérimentations sur ces systèmes, nous avons choisi de générer de manière aléatoire les séquences de test (en accord avec le nombre de vecteurs donné). Pour coller au plus prés de ce que pourraient être ces séquences pour des cœurs industriels, les vecteurs ont été générés de telle manière à contenir 80% de bits non spécifiés (X). La plupart des auteurs référencent plutôt un taux de 95% à 99%, mais, par soucis d'honnêteté, nous avons délibérément choisis un taux plus bas pour ne pas favoriser l'intervention de la compression dans notre procédure.

Cœurs	Nombre de bascules	Nombre de vecteurs de test	Nombre de chaînes de scan
1	592	134	14
2	167	74	2
3	53	57	1
4	216	268	4
5	127	51	4
6	94	36	2
7	94	34	2
8	104	31	2
9	64	68	1
10	13	29	1
11	9	15	1
12	13	16	1
13	0	512	0
14	0	1024	0

Tableau 3.2 : Caractéristiques des cœurs du système n°2

La technique de compression utilisée ici est celle présentée dans le Chapitre 2. Rappelons néanmoins que n'importe quelle méthode répondant aux contraintes du test des Systèmes sur puces peut être utilisée de la même façon.

3.4.2. Résultats et analyse

Comme il a été expliqué plus haut, notre méthode peut traiter des systèmes dont les cœurs sont fournis avec un wrapper prédéfini, ou avec des cœurs dont la conception des wrappers est à la charge du systémier. Nous présentons dans la suite des résultats sur ces deux possibilités.

3.4.2.1. Wrappers fixés.

Nous assumons dans un premier temps que les wrappers des cœurs sont fixés. Ils ont été intégrés en accord avec la norme IEEE 1500 par le concepteur du cœur, et ne sont plus modifiables par le systémier. Dans cet exemple, la taille des wrappers est égale au nombre de chaînes de scan d'un cœur.

Nous appliquons notre méthode au système n°1 décrit dans la section précédente. Le nombre de canaux ATE est fixé à 32, et la taille maximale du bus de test TAM à 64 bits. L'algorithme explore les architectures allant de 2 à 6 lignes. Les résultats sont présentés dans le Tableau 3.3.

Nombre de lignes	Nombre de configurations	TAT	Détail des lignes (W_{ATE_i} / W_{TAM_i})	Nombre de bits utilisés sur le TAM
2	522	127413	(16,16) / (16, 48)	30
3	44639	90457	(8,9,15) / (14,16,34)	42
4	1345142	68361	(5,7,8,12) / (7,14,16,27)	53
5	18605924	57941	(5,5,7,7,8) / (6,12,14,16,16)	64
6	142238520	57941	(1,4,5,7,7,8) / (1,5,12,14,16,16)	63

Tableau 3.3 : Résultats pour le système n°1, 32 canaux ATE, TAM 64 bits, wrappers fixés

La deuxième colonne du Tableau 3.3 indique le nombre de configuration architecturale sur les bus ATE / TAM qui ont été explorées. La troisième colonne reporte le temps de test total du système obtenu avec l'architecture retenue. Les détails de cette architecture sont donnés dans la quatrième colonne. La dernière colonne précise le nombre réel de bits utilisés sur le bus TAM.

Par exemple, pour l'architecture avec 3 lignes, 44639 configurations ont été explorées. La configuration retenue pour cette architecture entraîne un temps de test total du système de 90457 cycles. Cette architecture impose l'utilisation de trois décompresseurs tels que :

$$(W_{ATE_1} / W_{TAM_1}) = (8 / 14)$$
$$(W_{ATE_2} / W_{TAM_2}) = (9 / 16)$$
$$(W_{ATE_3} / W_{TAM_3}) = (15 / 34)$$

Sur ces résultats, on remarque tout d'abord que toues les architectures potentielles sont explorées, y compris celle ne nécessitant pas de décompresseur. Par exemple, pour la configuration

utilisant 2 lignes, l'architecture proposée par l'algorithme n'incluse pas de décompresseurs sur la première ligne : $W_{ATE_1} = W_{TAM_1} = 16$.

Deuxièmement, alors que la taille maximale du bus TAM a été fixée à 64 bits, pour certaines architectures (celles avec 2, 3, 4 ou 6 lignes), tous les bits ne sont pas utilisés. Ceci est principalement dû au choix de la taille des wrappers pour chaque cœur. Au final, le bus de test TAM sera conçu avec la taille réelle (respectivement 30, 42, 53 et 63 bits), et non avec la taille maximale de 64 bits donnée au départ.

Parmi toutes les architectures explorées, le meilleur temps de test total du système TAT est obtenu pour une configuration de 5 lignes. L'architecture retenue et l'ordonnancement associé sont donnés sur les Figure 3.25 et Figure 3.26.

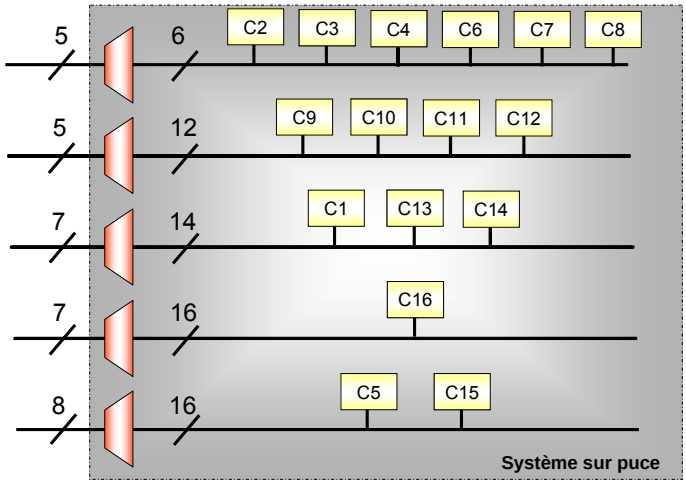


Figure 3.25 : Architecture retenue pour le système n°1,
pour 5 lignes, 32 canaux ATE, TAM 64 bits, wrappers fixés

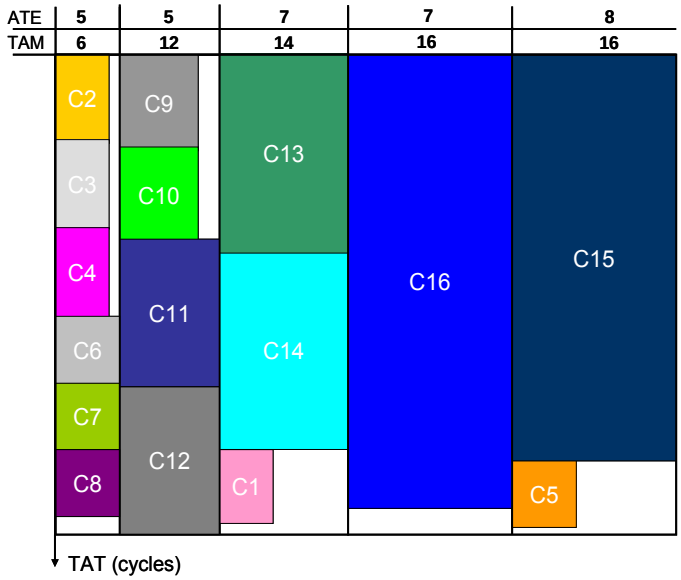


Figure 3.26 : Ordonnancement du test pour le système n°1,
pour 5 lignes, 32 canaux ATE, TAM 64 bits, wrappers fixés

Pour un plus grand nombre de ligne de test, les expérimentations ont montré que le temps de test total du système TAT augmente. Ceci est expliqué par le fait que la taille des wrappers soit fixée à l'avance. En effet, le plus petit des bus doit au moins pouvoir contenir le wrapper le plus grand. Or, augmenter le nombre de ligne avec cette contrainte freine les possibilités de parallélisations.

Le Tableau 3.4 présente une comparaison entre les résultats obtenus par notre méthode avec ceux des architectures standards, c'est-à-dire sans compression, mais dans les mêmes conditions. Dans un premier cas, la même limite sur le nombre de canaux ATE est respectée (32 canaux ATE donc un bus TAM de 32 bits). Dans le second cas, la même taille de bus TAM est observée (TAM de 64 bits, donc 64 canaux ATE maximum). Les résultats sont reportés dans le Tableau 3.5.

	Architecture			
Nombre de lignes	Compression 32 -> 64 bits		Standard 32 bits	Standard 64 bits
2	127413	30	127413	127413
3	90457	42	127413	90457
4	68361	53	127413	68361
5	57941	64	131210	57941
6	57941	63	131210	55738

Tableau 3.4 : Comparaison avec les architectures standards

Pour les configurations standards 32 bits sans compression, le temps de test total TAT est de 127413 cycles pour 2 à 4 lignes et de 131210 cycles pour 5 et 6 lignes. Le meilleur résultat sans compression de 131210 cycles est comparé au meilleur résultat avec notre méthode qui est de 57941 cycles. L'utilisation de la compression de données de test dans le cadre du test des systèmes sur puces conduit donc à un réduction de 54% du temps de test total sur cet exemple (certes au prix d'une augmentation de la surface de silicium dédiée au test: bus TAM plus large, décompresseurs, etx…).

En ce qui concerne les architectures standard 64 bits sans compression, sauf pour la configuration à 2 lignes de test, les résultats sont toujours meilleurs sans compressions. Cependant, l'architecture 64 bits implique aussi de disposer de 64 canaux ATE au lieu de 32 canaux avec la compression. Dans cet exemple, notre méthode permet donc de diviser par 2 le nombre de canaux ATE requis pour le test du système au prix d'une augmentation du temps de test total de seulement 4%.

3.4.2.2. Wrappers libres.

Une deuxième série d'expérimentation a été menée. Elle concerne cette fois le cas où les wrappers n'ont pas été intégrés par le concepteur des cœurs. Les cœurs sont fournis brut, avec leurs entrées/sorties et la circuiterie de test. Il revient alors au systémier de construire les wrappers autour

des cœurs. Cela accorde un degré de liberté supplémentaire pour l'optimisation du test du système, mais ce degré de liberté supplémentaire augmente considérablement la complexité des algorithmes et donc le temps de calcul nécessaire à l'obtention de la solution. On rappelle que toutes les tailles de wrappers ne sont pas utilisées puisque, comme cela a été expliqué dans la section 3.3.3.1, seuls quelques paliers sont intéressants en termes de rapport taille du wrapper / temps de test.

Les résultats obtenus sur le système n°1 sont donnés sur le Tableau 3.5. On remarque ici le même comportement général que dans le cas précédent. Un gain en temps de test avec la même contrainte de 32 canaux de test, ou un gain en canaux de test pour une très faible augmentation du temps de test pour l'architecture 64 bits.

On peut toutefois remarquer que, par rapport aux résultats obtenus avec les wrappers fixés, ces résultats sont meilleurs en termes de temps de test. Ceci peut s'expliquer par une meilleure utilisation des bus TAM grâce à une plus grande flexibilité de taille des wrappers ce qui se traduit par de meilleurs temps de test total du système.

	Architecture		
Nombre de ligne	Compression 32 -> 64 bits	Standard 64 bits	Standard 32 bits
2	66596	52953	97216
3	61277	49814	96624
4	57337	49129	96736
5	55101	48592	96624
6	54140	48517	96563

Tableau 3.5 : Comparaison avec les architectures standards pour le système n°1

On observe le même comportement sur le système n°2. Les résultats sont donnés dans le Tableau 3.6. Deux architectures ont été utilisées pour ce système: une première utilisant 32 canaux ATE pour un TAM de 64 bits (comme pour le système n°1) et une architecture utilisant 26 canaux ATE pour un TAM de 32 bits. Une fois de plus, selon les contraintes, on obtient soit un gain en temps de test total du système, soit un gain sur le nombre de canaux requis pour effectuer le test.

	Architecture avec compression		Architecture Standard		
Nombre de ligne	32→64	16→32	64 bits	32 bits	16 bits
2	17492	26256	15153	19633	33952
3	14185	23084	11274	17892	33718
4	12996	21409	11274	17235	33824
5	12399	20719	11274	17215	33824
6	12138	20667	11274	17235	33824

Tableau 3.6 : Comparaison avec les architectures standards pour le système n°2

3.5. Conclusion

Dans ce chapitre nous avons montré l'apport de l'utilisation de la compression des données de test dans le cadre du test des systèmes sur puces. Nous avons vérifié que l'augmentation du parallélisme du test des cœurs du système apporté par la compression apporte un réel bénéfice en termes de temps de test total du système. En effet, bien que la compression des données de test implique une légère augmentation du temps de test individuel d'un cœur, cet effet est largement compensé par l'augmentation du parallélisme du test de ces cœurs.

Dans un premier temps, nous avons étudié comment la compression pouvait s'intégrer dans une architecture de test des systèmes en accord avec la norme IEEE 1500. L'utilisation de lignes de test indépendantes, chacune possédant son propre décompresseur, permet de palier les contraintes liées à l'utilisation de la compression des données de test lors du test en parallèle de plusieurs cœurs. Nous avons ensuite proposé une méthode qui explore l'ensemble des solutions architecturales, depuis un seul décompresseur pour tout le système jusqu'à l'utilisation d'un décompresseur par cœur. Un balayage sur le nombre de ligne de test à utiliser permet d'obtenir une solution architecturale complète fixant toutes les variables du problème afin d'obtenir le meilleur temps de test total du système.

De nombreuses expérimentations ont été menées sur deux systèmes de références. Un premier basé sur des circuits ISCAS'89, et un second proposé par ITC'02. Dans le premier cas, les données de test ont été obtenue, comme dans le Chapitre 2, par un générateur de vecteurs de test industriel, alors que, dans le second cas, aucune donnée sur les cœurs n'étant disponibles, nous avons utilisé des séquences aléatoires comprenant 80% de valeurs non spécifiées. Ces expérimentations ont principalement montré qu'avec notre méthode, un gain en temps de test total du système de 50% avait été atteint. Elles ont également montré que, pour un temps de test total équivalent, notre méthode permet de réduire de moitié le besoin en canaux testeur. Remarquons toutefois que, bien que nos expérimentations utilisent la technique de compression présentée dans le Chapitre 2, tout autre technique peut être utilisée à la condition qu'elle respecte les contraintes liées au test des IP et des systèmes sur puces.

3.6. Références bibliographiques

[IEEE05] http://grouper.ieee.org/groups/1500/

[Lars02] E. Larsson, H. Fujiwara: Power constrained preemptive TAM scheduling. Proceedings of the 7th European Test Workshop, ETW 2002, pp. 411-416.

[Iyen02] V. Iyengar, K. Chakrabarty, E.J. Marinissen: Wrapper/TAM co-optimization, constraint-driven test scheduling, and tester data volume reduction for SOCs. Proceedings of the 39th annual Design Automation Conference, DAC 2002, pp. 685-690.

[Chou97] R. Chou, K. Saluja, V. Agrawal: Scheduling tests for VLSI systems under power constraints. IEEE Transaction on VLSI Systems, Vol. 5, N° 2, 1997, pp. 175- 185.

[Mure00] V. Muresan, X. Wang, M. Vladutiu: A comparison of classical scheduling approaches in power-constrained block-test scheduling. Proceedings of the International Test Conference, ITC 2000, pp. 882-891.

[Lars01] E. Larsson, Z. Peng: An integrated system-on-chip test framework. Proceedings of the Conference on Design, Automation and Test in Europe, DATE 2001, pp. 138-144.

[Iyen01] V. Iyengar, K. Chakrabarty: Precedence-based, preemptive, and power-constrained test scheduling for system-on-a-chip. Proceedings of the 19th VLSI Test Symposium, VTS 2001, pp. 368-374.

[Flot01] M.L. Flottes, J. Pouget, B. Rouzeyre: Sessionless test scheme: power-constrained test scheduling for system-on-a-chip. Proceedings of the International Conference on Very Large Scale Integration, VLSI-SoC 2001, pp.105-110.

[Chak01] K. Chakrabarty, V. Iyengar, E.J. Marinissen: Test wrapper and test access mechanism co-optimization for system-on-chip. Proceedings of the International Test Conference, ITC 2001, pp. 1023-1032.

[Mari00] E.J. Marinissen, S.G. Goël, M. Lousberg: Wrapper design for embedded core test. Proceedings of the International Test Conference, ITC 2000, pp. 911-920.

[Poug02] J. Pouget: Test des systèmes sur puce : Ordonnancement et exploration de l'espace des solutions architecturales. Thèse de doctorat, LIRMM – Université de Montpellier II, 2002.

[Chan03] A. Chandra, K. Chakrabarty: Test data compression and test resource partitioning for system-on-a-chip using frequency-directed run-length (FDR) codes. IEEE Transaction on Computers, Vol. 52, N° 8, 2003, pp. 1076-1088.

[Gonc03] P.T. Gonciari, B.M. Al-Hashimi, N. Nicolici: Variable-length input Huffman coding for system-on-a-chip test. IEEE Transaction on Computer-Aided Design, Vol. 22, N° 6, 2003, pp. 783-796.

[Lars07] A. Larsson, E. Larsson, P. Eles, Z. Peng: Optimized integration of test compression and sharing for SOC testing. Proceedings of the Conference on Design, Automation and Test in Europe, DATE 2007, pp. 207-212.

[Lars05] A. Larsson, E. Larsson, P. Eles, Z. Peng: SOC test scheduling with test set sharing and broadcasting. Proceedings of the 14th Asian Test Symposium, ATS 2005, pp. 162-169.

[Iyen03] V. Iyengar, A. Chandra: A unified SOC test approach based on test data compression and TAM design. Proceedings of 18th International Symposium on Defect and Fault Tolerance in VLSI Systems, DFT 2003, pp. 511-518

[Gonc04] P.T. Gonciari, B.M. Al-Hashimi: A compression-driven test access mechanism design approach. Proceedings of the European Test Symposium, ETS 2004, pp. 100-105.

[Gonc05] P.T. Gonciari, B.M. Al-Hashimi, N. Nicolici: Integrated test data decompression and core wrapper design for low-cost system-on-a-chip testing. Proceedings of the International Test Conference, ITC 2002, pp. 64-70.

[Lars08] A. Larsson, E. Larsson, K. Chakrabarty, P. Eles, Z. Peng: Test-architecture optimization and test scheduling for SOCs with core-level expansion of compressed test patterns. Proceedings of the Conference on Design, Automation and Test in Europe, DATE 2008, pp. 188-193.

[Wang07] Z. Wang, K. Chakrabarty, S. Wang: SoC testing using LFSR reseeding, and scan-slice-based TAM optimization and test scheduling. Proceedings of the Conference on Design, Automation and Test in Europe, DATE 2007, pp. 201-206.

[Mitr02] S. Mitra, K.S. Kim: X-compact, an efficient response compaction technique for test cost reduction. Proceedings of the International Test Conference, ITC 2002, pp: 311-320.

[Rajs05] J. Rajski, J. Tyszer, C. Wang, S.M. Reddy: Finite memory test response compactors for embedded test applications. IEEE Transaction on Computer-Aided Design, Vol. 24, N° 4, 2005, pp. 622- 634.

[Benc89] F. Brglez, D. Bryant and K. Kozminski: Combinational profiles of sequential benchmark circuits. Proceedings of International Symposium on Circuits And Systems, ISCAS 1989, pp. 1929-1934.

http://www.cbl.ncsu.edu/benchmarks/Benchmarks-upto-1996.html

[Benc02] E.J. Marinissen, V. Iyengar, K. Chakrabarty: A set of benchmarks for modular testing of SOCs. Proceedings of the International Test Conference, ITC 2002, pp. 519-528.
http://itc02socbenchm.pratt.duke.edu/

[Bayr01] I. Bayraktaroglu, A. Orailoglu: Test volume application time reduction through scan chain concealment. Proceedings of the 38th annual Design Automation Conference, DAC 2001, pp. 151-155.

[Syno01] http://www.synopsys.com/tools/implementation/rtlsynthesis/pages/tetramaxatpg.aspx

Chapitre IV:

Compression des données de test et NoCs

4.1. Réutilisation du NoC pour le test 135

4.1.1. Réseau sur puce: définition 135

4.1.2. Pourquoi réutiliser le NoC 136

4.1.3. Réutilisation du NoC pour le test 136

4.1.3.1. Modifications structurelles 136

4.1.3.2. Format des données 137

4.1.4. Limitation du test avec un NoC : les ports d'entrée 138

4.1.5. Réduction du temps de test 138

4.2. Etat de l'art 140

4.2.1. Optimisation de l'architecture 140

4.2.2. Test préemptif 140

4.2.3. Test non-préemptif 141

4.3. Méthode proposée 141

4.3.1. Architecture de Test 142

4.3.2. Impact de la compression dur les données de test 143
4.3.3. Problème et notations 144
4.3.4. Test non-préemptif et compression 145
4.3.4.1. Exploration architecturale 145
4.3.4.2. Ordonnancement du test non-préemptif 146
4.3.4.3. Exemple applicatif 147
4.4. Résultats 150
4.4.1. Procédure expérimentale 150
4.4.2. Résultats et analyse 151
4.5. Conclusion 155
4.6. Références bibliographiques 156

Un des principaux challenges lors de la conception de systèmes sur puce est la communication entre les différents modules composant le système. En effet, un tel système n'est au final qu'un assemblage hétéroclite de cœurs provenant de sources multiples, qui n'ont pas été spécifiquement conçus pour fonctionner ensemble. Faire communiquer entre eux ces différents éléments pose parfois des problèmes de protocoles et de synchronisation. Jusqu'à présent, l'interconnexion entre ces blocs était réalisée au moyen de bus de données traditionnels.

Ces architectures de bus sur puce deviennent un goulot d'étrangement essentiellement pour deux raisons. D'abord, les architectures de bus doivent évoluer en permanence pour suivre la complexité croissante des systèmes sur puce contenant un milliard de transistors ou plus, organisés en dizaines, voire en centaines de blocs IP, ce qui suppose un allongement des délais de mise sur le marché des nouveaux systèmes sur puce. De plus, au lieu de suivre la même évolution que la loi de Moore pour les transistors dont la taille diminue de plus en plus, les interconnexions se complexifient à chaque génération puisqu'elles doivent connecter de plus en plus de blocs fonctionnels sur la puce. Par conséquent, le bus influe de plus en plus sur les facteurs de coût/performance, comme la surface de silicium, la vitesse de communication sur le circuit et la consommation globale.

Une solution alternative, est d'opter pour une infrastructure de communication configurable préétablie. Depuis quelques années, on a vu émerger un nouveau type d'architecture d'interconnexion des cœurs dans les systèmes sur puces [Beni02]. Cette architecture baptisée "Réseau sur Puce" (ou NoC pour Network-on-Chip) se présente sous une forme semblable en tout point aux réseaux de télécommunications modernes que nous utilisons chaque jour. Ces réseaux présentent plusieurs avantages: ils ont une topologie générique, ils sont adaptables à la taille du système (nombre de cœurs, éloignement entre les cœurs…) et autorisent un parallélisme plus grand que celui des architectures à bus. Il apparaît évident que le recours à ces réseaux sur puce révèle particulièrement toute son efficacité pour les circuits de grande dimension. De plus, les réseaux sur puces sont compatibles avec l'approche "Globalement Asynchrone, Localement Synchrone" (GALS) [Chap84] autorisant la présence d'horloges différentes au sein d'un même système

Cependant, on retrouve avec les réseaux sur puce la même problématique de test qu'avec les systèmes sur puce basés sur les bus. Il s'agit de pouvoir tester les cœurs du système et de faire en sorte que ce test soit le plus rapide possible. Or, dans le cadre des réseaux sur puce, on dispose déjà d'une structure d'acheminement des données dotée d'une forte possibilité de parallélisme. Nous proposons dans ce chapitre, une technique de test basée, d'une part sur la réutilisation du réseau fonctionnel pour le transport des données de test, et d'autre part sur une technique de compression horizontale de ces données.

Dans une première partie, le principe de la réutilisation du réseau pour le test sera présenté. Les méthodes présentées dans la littérature seront ensuite exposées. Enfin, nous détaillerons notre méthode regroupant réutilisation du réseau et compression des données de test, puis nous fournirons une série de résultats expérimentaux sur les systèmes ITC'02.

4.1. Réutilisation du NoC pour le test

4.1.1. Réseau sur puce: définition

Un **réseau sur puce** ou **NoC** n'est pas vraiment différent d'un système sur puce comme nous avons pu en voir dans le Chapitre 3: On y trouve des cœurs réalisant des fonctions spécifiques, et ces cœurs doivent communiquer entre eux. La vraie différence se situe dans la façon dont les cœurs communiquent. Dans les systèmes traditionnels, les transferts de données sont réalisés par des bus. Dans le cadre des réseaux sur puce, comme leur nom l'indique, une structure de communication de la forme d'un réseau est implantée. Ce réseau est parfaitement semblable aux réseaux de télécommunication moderne, comme par exemple le réseau Internet. Une représentation d'un réseau sûr puce est donnée Figure 4.1. Il s'agit d'un exemple de topologie en forme de grille. Il en existe d'autres formes de topologies (tore, hypercube, anneau, etc…) [Duat97], mais, pour plus de clarté dans le reste de l'exposé, nous nous attacherons à ce modèle. La méthode présentée ici est bien sur également applicable à d'autres topologies. Le réseau est constitué de **routeurs** qui assurent d'une part les transmissions de données entre eux mais aussi l'acheminement final de données à un cœur, et, évidemment, les cœurs eux-mêmes. Chaque ensemble routeur+cœur est situé sur un nœud du réseau. Un des avantages du réseau est qu'il est complètement générique et scalable.

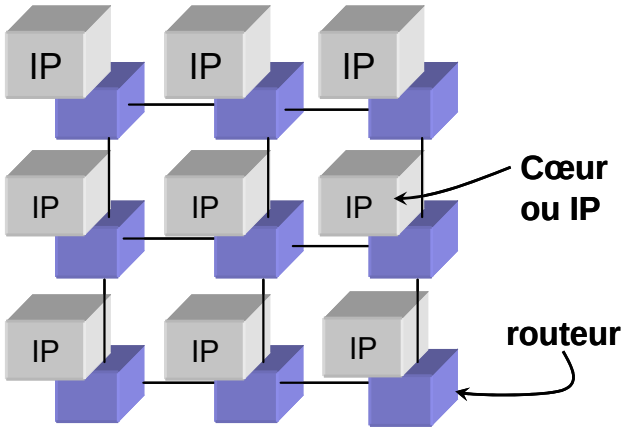


Figure 4.1 : Réseau sur puce

Comme pour les systèmes sur puce traditionnels, un cœur n'est pas connecté directement à un routeur, il faut réaliser une adaptation du format des données entre le réseau et le cœur. En effet, un réseau peut être assimilé localement à un bus de dimension W fixée. Il faut adapter les entrées/sorties de chaque cœur pour le connecter au réseau. On retrouve le même principe que dans les systèmes à bus où un anneau d'isolation (**wrapper**) est inséré entre le cœur et le réseau, ou plus précisément entre le cœur et le routeur comme décrit dans la Figure 4.2. Il faut préciser que ce wrapper est conçu pour ce mode de fonctionnement et qu'il convient donc de le modifier pour qu'il soit utilisable pour le test. De la même manière que pour le test des systèmes traditionnels, le wrapper doit posséder un mode "normal" et un mode "test" compatible avec la norme IEEE 1500 [IEEE05].

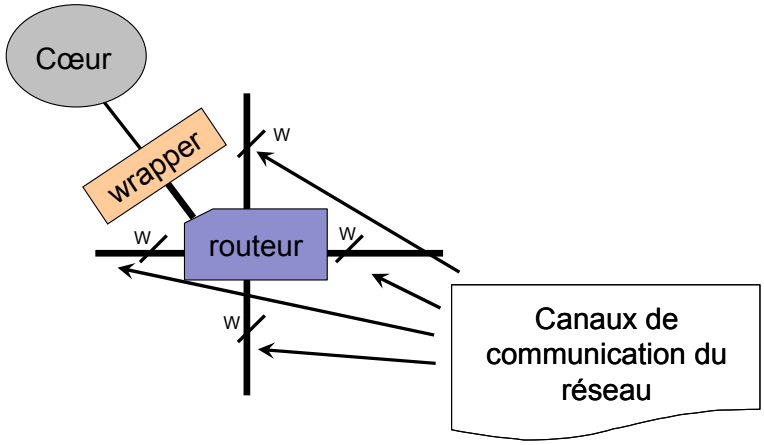


Figure 4.2 : Détail du réseau sur puce

4.1.2. Pourquoi réutiliser le NoC

Dans un système où un réseau est utilisé pour assurer la communication entre les cœurs, il est particulièrement intéressant de chercher à l'utiliser également pour le test de ces cœurs. En effet, le réseau est un moyen de communication efficace qui fournit un accès simple et régulier à tous les cœurs. Mieux, il propose de nombreux chemins possibles pour faire communiquer entre eux deux cœurs. Si un chemin est déjà utilisé par deux autres cœurs, il est facile d'en trouver un autre, peut-être plus long, mais disponible immédiatement. Cette possibilité présente des avantages en mode fonctionnel, mais, et c'est ce qui nous intéresse ici, également en mode test comme cela a été présenté dans [Cota04] [Liu06] [Cota06]. Au chapitre précédent, on a vu que le parallélisme est la clé d'un test plus rapide. Or l'utilisation d'un réseau permet d'avoir plusieurs chemins disponibles pour alimenter un cœur. Cette particularité offre beaucoup plus de souplesse qu'un bus standard.

Pour parvenir au même résultat avec une architecture à bus, le coût en surface serait prohibitif. De même, on peut se poser la question sur la nécessité de rajouter une structure de test à bus compatible IEEE 1500, alors que le réseau, avec très peu de modifications (notamment au niveau des wrappers) peut parfaitement jouer le rôle de TAM (Test Acces Mechanism).

Cependant, il faut noter que le réseau dans son ensemble (routeurs, wrappers, placement des cœurs, etc…) est conçu et optimisé pour le mode fonctionnel, pas pour le test. Autrement dit, l'architecture n'est pas optimisée pour le test, mais ce défaut est largement compensé par la souplesse d'utilisation qui caractérise un réseau.

4.1.3. Réutilisation du NoC pour le test

4.1.3.1. Modifications structurelles

Réutiliser la structure du réseau pour tester les cœurs impose de faire quelques modifications mineures sur le réseau et sur son fonctionnement. Dans un premier temps, il est nécessaire d'adapter les wrappers présents dans le réseau sur puce et les doter d'un mode de fonctionnement adapté au test. Pour cela, il suffit d'utiliser un wrapper répondant à la norme IEEE 1500 exactement comme

dans le cas du test d'un système à bus classique. Dans ce cas, la taille du wrapper est définie par la taille du NoC (par exemple un wrappers de 32 cellules pour un NoC 32 bits). Le reste du réseau (routeurs, topologie) lui, n'est pas modifié. Le test peut être considéré comme une simple tâche du réseau. Les données de test sont présentées aux entrées du système, et les réponses sont récupérées aux sorties (Figure 4.3).

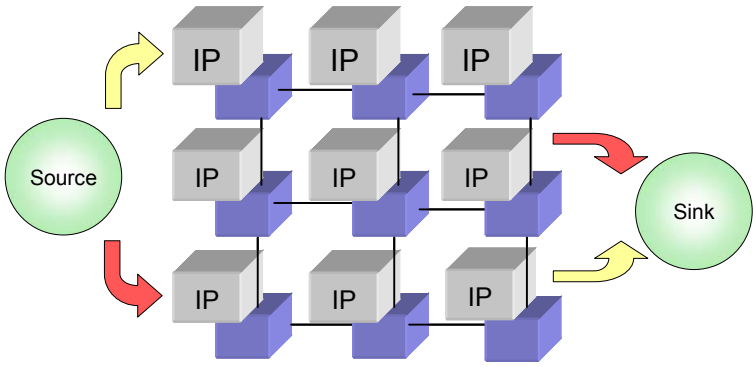


Figure 4.3 : Réutilisation du NoC pour le test

Les broches d'entrée et de sortie du système sont regroupées de manière à former des **ports** d'entrée et de sortie. Pour un NoC de taille W, les broches d'entrée/sortie sont organisées en groupe de W broches. Ces ports sont réutilisés pour le test. Les vecteurs de test traversent alors le NoC de routeur en routeur, depuis un port d'entrée jusqu'à un cœur. De la même manière, les réponses sont renvoyées dans le NoC depuis le cœur jusqu'à un port de sortie. La source des vecteurs et le récepteur des réponses peuvent être soit externes (ATE), soit sur puce en tant que cœur du NoC (BIST, analyseur de signature…)

4.1.3.2. Format des données

En fonctionnement normal, des informations liées à la destination et à la taille du message transféré sont ajoutées aux données transférées à travers le NoC. Le message est constitué d'un **en-tête** contenant les données relatives à la destination du message, les données destinées au cœur, ainsi qu'une **queue** de message indiquant la fin du message. On appelle "**payload**" les données fonctionnelles destinées au cœur. Ce Payload est découpé en "**flits**" qui constituent le message mis en forme sur W bits, W étant la taille du NoC.

Les données de test doivent être adaptées au transfert via le NoC. La Figure 4.4. Le payload est constitué de vecteurs de test, tous ou juste une partie comme cela sera détaillé dans la section 4.2. Un en-tête réservé au test est inséré entre l'en-tête et le payload. Il contient notamment le signal de démarrage du test mettant le wrapper en configuration de test. Il peut également contenir des informations de contrôle telles que les commandes de décalage ou de capture, mais ces informations peuvent aussi bien être contenues dans le payload. L'ensemble de ces données (en-tête, payload et queue) forme un paquet de test.

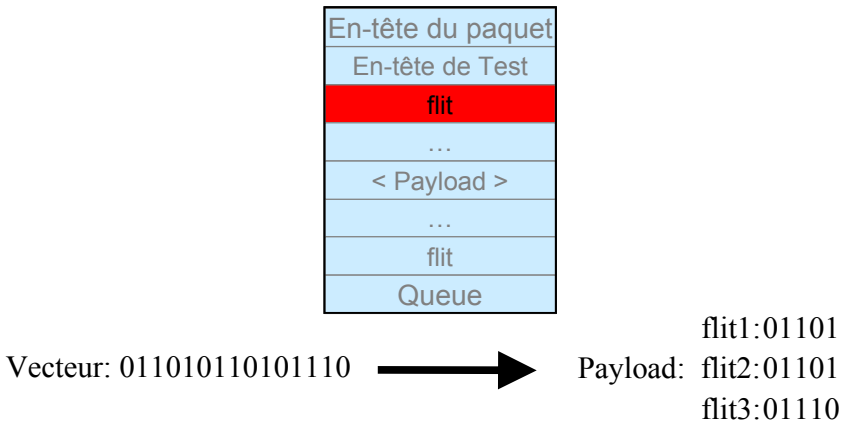


Figure 4.4 : Format des données de test dans le NoC

4.1.4. Limitation du test avec un NoC : les ports d'entrée

Les messages (vecteurs et réponses) circulent des ports d'entrée du système vers les ports de sortie après être passés par le cœur visé. On peut schématiser ce fonctionnement comme différents bus TAM composés chacun d'un port d'entrée, d'un chemin passant par le cœur et d'un port de sortie. La différence avec l'architecture à bus réside dans les ports et le chemin qui peuvent changer d'un cœur à l'autre, voire d'un vecteur à l'autre.

Il apparaît donc que le NoC produit naturellement du parallélisme lors du test puisque plusieurs chemins existent et que, par conséquent, plusieurs cœurs peuvent être testés en même temps sans effort de conception supplémentaire. Cependant, le nombre de voies possibles est limité par la possibilité de créer des ports d'entrée/sortie avec les broches fonctionnelles présentes sur le système. Par exemple le système p93791 extrait de ITC'02 [Benc02] possède 103 entrées, 79 sorties, 66 bidirectionnelles. En utilisant un NoC de 32 bits, il est possible de constituer au maximum 6 ports de test (4 ports d'entrée et 2 de sortie ou 3 ports de chaque). Il existe donc une forte corrélation entre le temps de test du système complet et le nombre de ports disponibles. En effet, le temps de test dépend, entre autres, du parallélisme qui, lui, dépend du nombre de ports.

4.1.5. Réduction du temps de test

Il est donc intéressant, dans l'objectif de réduire le temps de test total du système, d'augmenter le nombre de ports de test sur le système. Pour cela, une solution simple existe: augmenter le nombre de broches dédiées au test sur le système. Un exemple est donné sur la Figure 4.5 pour le système p93791 (103 entrées, 79 sorties, 66 bidirectionnelles, 32 Cœurs), avec un NoC 32 bits et le nombre de ports d'entrées/sorties variant de 3 à 6 (nombre de ports d'entrée = nombre de ports de sortie). On remarque que cette solution entraîne un coût non négligeable en termes de broches additionnelles sur le design. D'autre part, on constate sur la Figure 4.5 une augmentation du nombre de canaux ATE nécessaires au test de ce système. En effet, l'augmentation du nombre de broches pour le test (et donc le nombre de ports de test) implique l'augmentation du nombre de canaux ATE nécessaires.

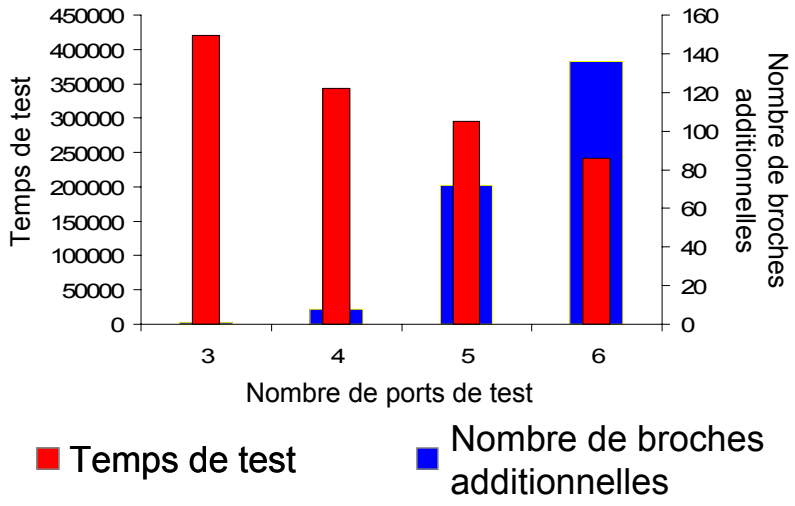


Figure 4.5 : Temps de test vs nombre de ports de broches additionnelles

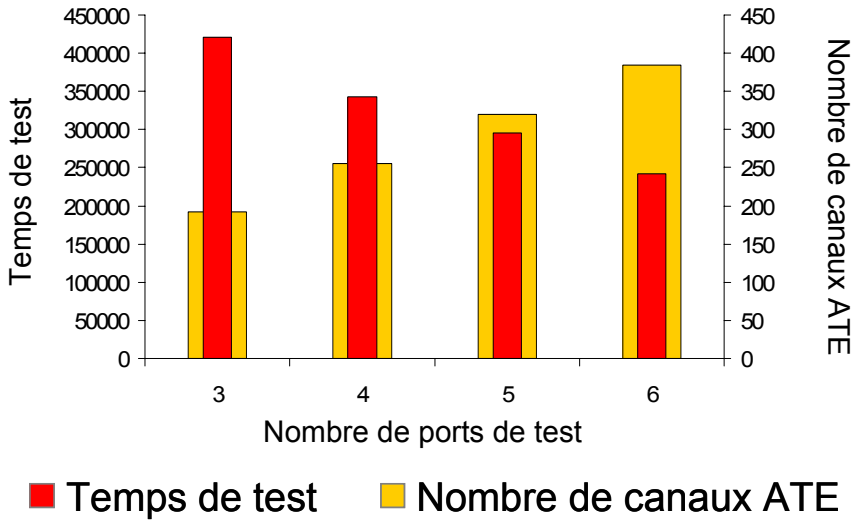


Figure 4.6 : Temps de test vs nombre de canaux ATE

Réduire efficacement le temps de test total du système coûte donc cher en broches additionnelles et en canaux testeur. Obtenir un résultat significatif de cette manière n'est donc pas viable en pratique. Comme dans le Chapitre 3: il est donc intéressant de trouver une technique permettant d'augmenter virtuellement le nombre de broches utilisables par le test (et donc le nombre de ports de test) sans augmenter le nombre réel de broches sur le système, et surtout sans augmenter le nombre de canaux ATE requis pour ce test.

Dans la suite de ce chapitre, un état de l'art sur les méthodes réutilisant le NoC comme moyen de test est présenté. Nous proposons ensuite une méthode basée sur une technique de compression horizontale des données de test. Cette technique permet l'augmentation du nombre de ports de test sans augmentation des canaux testeur avec un faible impact sur le design du système. Les résultats, présentés en fin de chapitre, montrent une réduction significative du temps de test total du système.

4.2. Etat de l'art

L'utilisation de réseaux sur puce comme moyen de communication entre les cœurs d'un système est une technologie relativement nouvelle. Il existe donc peu d'études spécifiques sur le test de tels systèmes. Jusqu'à présent, l'idée communément admise était de les traiter comme des systèmes classiques en ajoutant au design toute l'architecture de test comprenant les bus de test TAM et les modifications des wrappers pour les rendre compatibles avec la norme IEEE 1500. Plus récemment, on a vu apparaître dans la littérature [Cota03], des méthodes introduisant l'idée de réutilisation du NoC pour le test dont le principe général a déjà été décrit dans les sections précédentes de ce chapitre.

4.2.1. Optimisation de l'architecture

De nombreux travaux ont visé à améliorer cette architecture de test et à la rendre plus performante en termes de temps de test. Comme il est difficile de modifier le réseau en lui-même puisqu'il est destiné au mode fonctionnel et non pas exclusivement au test (à la différence des bus TAM qui, eux, sont dédiés au test), ces travaux portent sur des évolutions dans la conception des wrappers en mode test. Il peut s'agir de modifier les cellules du wrappers [Amor06] ou d'en modifier l'agencement pour parvenir à une meilleure adéquation taille du NoC / taille du wrapper [Li08] et donc améliorer la bande passante du réseau afin de diminuer le temps de test total du système. Une autre façon d'optimiser l'utilisation de la largeur de bande du NoC a été présentée dans [Liu05]. Il s'agit ici de jouer sur les fréquences de test des cœurs. Cette technique permet d'envoyer plusieurs messages de test en utilisant les bits inutilisés du NoC. Le wrapper, cadencé par une PLL à une fréquence multiple de celle du testeur (n fois pour n messages simultanés) charge les données dans le cœur. Il a également été proposé dans [Amor04] de réutiliser les cœurs du système dans le rôle de générateur et d'analyseur de données de test. On peut ainsi se passer de ressources externes afin de laisser le NoC adapter au mieux les fréquences de test. Il faut noter que ces techniques peuvent impliquer des coûts en termes de complexité de conception, de surface additionnelle et de consommation électrique lors du test.

Outre ces travaux sur les architectures de test, la question du transfert des données de test a également été étudiée. On trouve dans la littérature deux approches dites "préemptive" ou "non-préemptive".

4.2.2. Test préemptif

Le test préemptif a été introduit dans [Cota04]. Dans cette méthode, un paquet de test contient seulement un vecteur de test (ou une réponse du circuit) d'un cœur donné. De plus, aucun chemin n'est particulièrement réservé pout le test d'un cœur. Les paquets de test sont classés dans une liste, puis, pour chacun, le premier chemin disponible est choisi. Ainsi, les données d'un même cœur peuvent suivre des chemins différents. Par contre, si aucun chemin n'est disponible, le paquet est en attente et donc le test de ce cœur peut être interrompu.

Ce phénomène peut être pénalisant au niveau du temps de test du cœur. En effet, en temps normal, la réponse du circuit à un vecteur est obtenue par décalage dans les chaînes de scan en même temps que le vecteur suivant y est chargé. Si le test est interrompu, les deux opérations ne peuvent plus être simultanées. Notons toutefois que cette méthode a été proposée dans l'objectif de réduire la puissance consommée lors du test. Or, temps de test et consommation sont deux paramètres antagonistes.

Cette méthode permet toutefois une meilleure utilisation de la bande passante offerte par le NoC. En effet, un paquet est envoyé dès qu'un chemin est disponible, sans restriction sur le fait qu'il doive suivre le même chemin que le précédent. Ce degré de liberté supplémentaire permet malgré tout de réduire le temps de test, ou, du moins, de maintenir un même temps avec une contrainte sur la puissance dissipée.

4.2.3. Test non-préemptif

Une autre approche a été présentée dans [Cota06]. L'objectif est de chercher à minimiser le temps de test du système, en tenant compte d'éventuelles contraintes de précédences de test de certains cœurs par rapport à d'autres. La stratégie utilisée est la suivante: une fois qu'un chemin est choisi pour le test d'un cœur, il est utilisé pour transférer tous les vecteurs de test (et toutes les réponses) en un seul paquet de test. Cela signifie que le test d'un cœur ne peut pas être interrompu.

Cette condition peut rendre une partie du réseau totalement inactive lors de l'envoi des séquences de test complètes aux cœurs visés. En contre partie, comme le flot de données est continu, les réponses sont expulsées des chaînes de scan à mesure que les vecteurs suivant sont envoyés. Cela implique un gain de cycles d'horloge par rapport à la méthode précédente. De plus, ce flot constant permet de réduire la synchronisation nécessaire au test d'un cœur. En effet, il n'est plus nécessaire de mettre "en pause " le test d'un cœur, d'où, moins de contrôle sur la puce et moins de données de contrôle à envoyer.

Il faut toutefois remarquer une contrainte dans l'utilisation de cette méthode : le maintien du flot de données de test suppose d'avoir obligatoirement un port d'entrée et un port de sortie monopolisés pendant toute l'opération. Alors que le test préemptif autorise d'avoir un nombre de port d'entrée et de sortie différents, le test non-préemptif impose que ce nombre soit identique.

Les résultats expérimentaux présentés dans [Cota06] montrent que le test non-préemptif conduit, au final, à des temps de test total du système inférieurs à ceux obtenus par le test préemptif. On peut en conclure qu'interrompre le test d'un cœur dans l'objectif d'optimiser l'utilisation du NoC avec une plus grande flexibilité sur le choix des chemins, ne compense pas complètement les délais imposé par cette interruption. Dans la suite, le test non-préemptif a donc été logiquement choisi pour le transfert des données de test. Un exemple applicatif d'ordonnancement du test des cœurs d'un système sera donné dans la section 4.3.4.2.

4.3. Méthode proposée

Nous proposons ici d'appliquer une technique similaire à celle proposée dans le cadre du test des systèmes à bus classiques. Il faut cependant s'adapter à la structure de test du NoC en tenant compte de ses contraintes mais aussi des possibilités d'optimisation qu'elle offre. Dans la section 4.1.4, nous avons vu que le facteur limitant, en termes de temps de test lors de la réutilisation d'un NoC pour le test, est le nombre d'interfaces de test disponibles. Ces interfaces sont créées à partir des broches fonctionnelles disponibles sur le système. Pour un NoC 32 bits, il faut 32 broches pour créer une interface. L'idée que nous proposons est d'utiliser une technique de compression horizontale afin de réduire le nombre broches fonctionnelles nécessaires à la constitution d'une interface de test. Cette approche permet de créer plus d'interfaces de test sans requérir l'ajout de broches supplémentaires et sans augmenter le nombre de canaux testeur nécessaires. Le parallélisme du test des cœurs peut ainsi être augmenté à moindre coût. Comme dans le chapitre précédent, il faut vérifier que, l'augmentation des temps de test individuels des cœurs induite par la décompression est effectivement compensée par le parallélisme et que le temps de test total du système est bien réduit de manière significative.

4.3.1. Architecture de Test

Le principe général est très simple et entraine peu de modifications architecturales par rapport à ce qui a été décrit dans la section 4.1.3. Il s'agit simplement d'insérer entre les broches fonctionnelles de l'interface de test et le wrapper du cœur, un décompresseur en entrée du système pour délivrer les vecteurs de test (Figure 4.7) ou un compresseur en sortie pour récupérer les réponses des cœurs (Figure 4.8). Nous assumons ici, comme dans le Chapitre 3, que les réponses peuvent être compressées avec n'importe quelle méthode de compaction de réponses [Mitr02] [Rajs05]. Il faut noter que ces techniques de compaction de réponses n'influent pas sur le temps de test puisque chaque réponse est traitée sans délai.

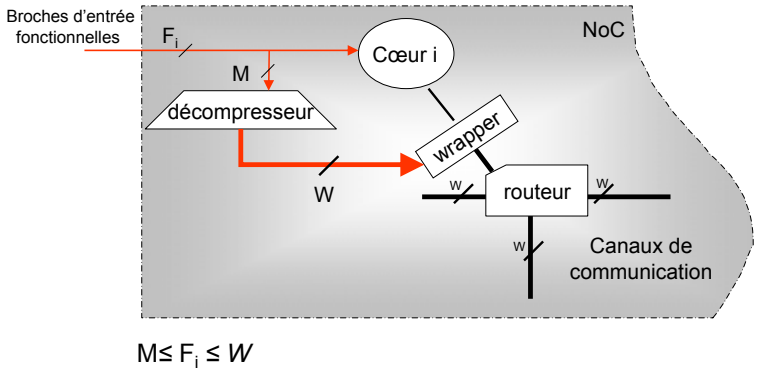


Figure 4.7 : Insertion d'un décompresseur en entrée du système

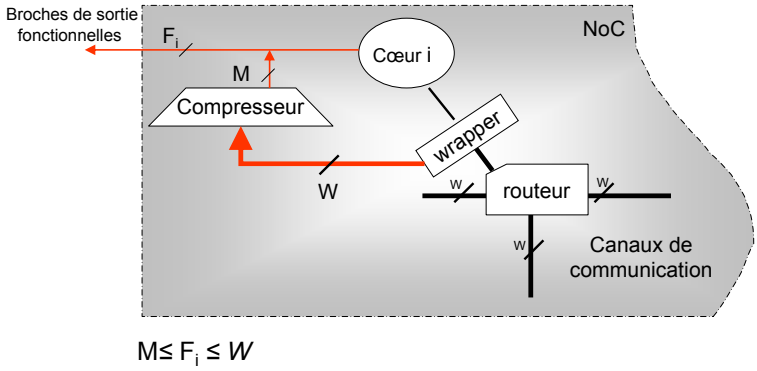


Figure 4.8 : Insertion d'un décompresseur en sortie du système

4.3.2. Impact de la compression dur les données de test

L'insertion d'un décompresseur ne modifie pas les données qui circulent dans le réseau. Cependant, les données qui sont stockées sur le testeur, sont compressées hors ligne. Il a déjà été montré dans les chapitres 2 et 3 que la compression peut engendrer une augmentation du temps de test lors de la présence de données incompressibles. Par ailleurs, la mise en forme des données spécifique au NoC décrite sans la section 4.1.3.2 provoque, de manière plus évidente encore que dans le cadre du test des systèmes à bus classiques, une augmentation du temps de transfert des données compressée. Cette augmentation est due à la présence des en-têtes et queues des paquets de données qui sont complètement spécifiés et donc difficilement compressibles.

Toujours sur l'exemple de la Figure 4.9 de la section 4.1.3.2, le paquet est constitué des en-têtes (l'en-tête normal et celui spécifique au test), des données de test (ici un vecteur de test) et de la queue signifiant la fin du message. Lors de la compression, le paquet entier est compressé, y compris les en-têtes et la queue. Cependant les données d'en-tête et de queue sont complètement spécifiées, il n'y a pas de valeurs indéterminées ou X. Or, toutes les techniques de compression utilisent ces valeurs indéterminées pour fonctionner. Il apparaît donc qu'il est particulièrement difficile de compresser ces données spécifiques au NoC. Cela vient s'ajouter aux éventuelles flits du payload ne pouvant pas être compressées. Notons toutefois que, sur une séquence complète (plusieurs centaines voire milliers de vecteurs), le fait que les données spécifiques au format NoC (en-tête et queue) ne sont pas compressibles est négligeable devant le nombre total de vecteurs, et donc n'entraîne qu'une infime augmentation du temps de test.

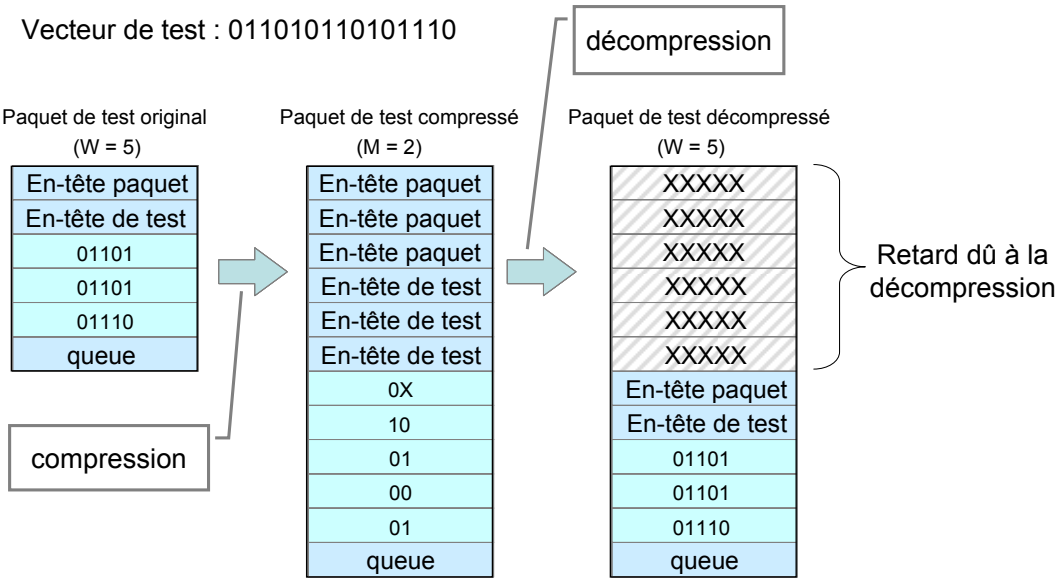


Figure 4.9 : Format des données lors de la compression

Il résulte de cette compression une augmentation du temps de transfert des paquets depuis le testeur vers le cœur. Or, dans le réseau, après décompression, les données ont repris leur format initial. L'augmentation du temps de transfert, et donc, au final, du temps de test du cœur, se traduit par une latence dans le réseau. Celle-ci est illustrée sur la Figure 4.9 par les 6 flits non spécifiées avant le paquet de test décompressé.

4.3.3. Problème et notations

Dans ce chapitre, les notations suivantes sont utilisées: soient n le nombre de cœurs du système, W la taille du NoC, p le nombre de couples de port d'entrée/sortie (ce qui correspond au nombre de décompresseurs) et Fi le nombre de broches fonctionnelles attribuées au port d'entrée i (Fi/W représente donc le rapport de compression r_i sur ce port d'entrée).

Nous ne traitons pas ici de l'aspect compression de sortie mais les mêmes notations peuvent être utilisées. Le temps de test d'un cœur j pour un rapport de compression r_i est noté t^j_{ri}, et le temps total de test du système est noté TAT (pout Temps d'Application du Test).

Comme dans le Chapitre 3, les temps de test de chaque cœur doivent être connus pour pouvoir effectuer un ordonnancement. Nous procédons ici de la même façon que dans la section 3.3.3.1. Les temps sont estimés par identification à la fonction mathématique suivante:

$$t^j_{r_i} = \frac{a}{r_i} + b \qquad \text{Équation 4.1}$$

De la même manière, le temps de test total TAT est le temps sur le plus long parmi les p port d'entrée possibles. Le temps relatif à un port d'entrée i est donné par l'équation 4.2:

$$TAT = \max\left(TAT_i, i = 1..p\right)$$

$$\text{avec} TAT_i = \sum_{\text{cœurs sur le port } i} t^j_{r_i} \qquad \text{Équation 4.2}$$

L'objectif de la méthode proposée ici est de fournir une architecture de test basée sur la réutilisation du NoC d'un système en y ajoutant une technique de compression horizontale. La compression permet de créer plus de ports de test à partir des broches fonctionnelles et ainsi d'augmenter le parallélisme du test des cœurs dans le système afin d'en réduire le temps de test total. Un ordonnancement du test des cœurs est associé à cette architecture. Il définit non seulement l'ordre de test des cœurs, mais aussi les ports d'entrée par lesquels sont acheminés les vecteurs de test (et par quel port de sortie sont récupérées les réponses) et le chemin utilisé à l'intérieur du réseau pour relier entrée-cœur-sortie.

Comme pour le test des systèmes à bus classiques, notre méthode doit donc déterminer un certain nombre de paramètres afin de fournir une architecture complète:

- Le nombre p de couples de ports d'entrée/sortie et leur localisation sur le NoC
- Le nombre de broches Fi (et donc de canaux testeur) attribuées à chaque port
- L'assignation des cœurs à un couple de ports d'entrée/sortie et la détermination d'un chemin d'accès à ce cœur
- Un ordonnancement du test avec un temps de test total TAT minimal

Il s'agit sensiblement du même problème que celui présenté dans la section 3.3.2 concernant le test des systèmes à bus classiques avec toutefois quelques différences. Tout d'abord dans le cadre des NoC, nous supposons les wrappers adaptés au réseau, c'est-à-dire un wrapper d'une taille maximale W bits pour un réseau de W bits. Pour un circuit avec moins de W chaînes de scan, la taille du wrapper est fixée au nombre de chaînes de scan. Ceci a pour effet de réduire la complexité de l'algorithme d'exploration d'architecture et l'évaluation préalable des temps de test de chaque cœur.

D'autre part, à la différence des architectures de test basées sur des TAM, lorsqu'un cœur est assigné à un couple d'entrée sortie, l'algorithme de test non-préemptif utilisé ici [Cota06] doit définir un chemin à travers le réseau. De même, la localisation des ports d'entrée/sortie est définie par ce même algorithme. Ces points ne seront pas détaillés dans la suite, la méthode étant utilisée ici sans modification.

4.3.4. Test non-préemptif et compression

4.3.4.1. Exploration architecturale

Il s'agit dans un premier temps de lister les différentes répartitions possibles des M canaux ATE disponibles en p ports d'entrée de test. Le nombre de ports p peut varier en théorie de 1 à R, où R est le nombre de nœuds dans le réseau. Pour un seul port, il n'y a qu'une seule partition et le parallélisme est minimal. Par contre, pour R ports, tous les cœurs sont directement reliés au testeur et aucun ordonnancement n'est nécessaire. En pratique l'exploration est donc limitée entre 2 et R/2 ports (ligne 1 sur la Figure 4.10). Pour chaque configuration du nombre de ports, les partitions de M canaux testeur en P ports sont obtenues, comme dans la section 3.3.3.2, par l'équation:

$$X(M,p) = \sum_{k=1}^{p} X(M-p,k) \text{ with} X(M,M) = X(M,1) = 1$$
$$\text{and} X(M,p) = 0 \quad \text{si p} > \text{n}$$

Équation 4.3

Ensuite, pour chaque partition de M canaux en p ports (ligne 2 sur la Figure 4.10), les paquets de test sont définis (a\). Pour chaque cœur, les temps de test en fonction des p rapports de compression relatif à cette partition sont évalués et mémorisés. Une fois ces données obtenues, l'ordonnancement de test non-préemptif est exécuté (b\). Le temps de test total obtenu est mémorisé s'il est meilleur que ceux précédemment obtenu par l'algorithme (c\).

1. Pour un nombre de port p variant de 2 à R/2
2. Pour chaque partition valide de M en p parties
   - a\ Définir les paquets de test de chaque cœur sur chaque port
   - b\ Exécuter l'ordonnancement non-préemptif
   - c\ Si le temps de test total obtenu est meilleur, mémoriser l'architecture et le temps de test

Figure 4.10 : Algorithme d'exploration architecturale

4.3.4.2. Ordonnancement du test non-préemptif

Il s'agit du point 2.b\ de l'algorithme précédent donné Figure 4.10. Pour tous les p ports donnés, le nombre de broche Fi est connu. Le rapport de compression Fi/W de chaque port est donc déterminé, permettant de définir le temps de test de chaque cœur selon le port par lequel ses données de test arrivent. Toutes les données nécessaires à l'exécution de l'ordonnancement de test non-préemptif [Cota06] sont désormais disponibles. Cet Algorithme est décrit sur la Figure 4.11.

Tout d'abord les différents chemins d'accès à un cœur sont listés. Il faut au préalable déterminer la localisation physique des ports de test dans le réseau (sur quel nœud).

Or, il n'existe pas de méthode simple permettant la détermination de la configuration optimale. Les auteurs de [Cota06] utilisent une boucle externe parcourant l'ensemble des solutions. Cette boucle est limitée par un nombre d'itération défini par l'utilisateur en fonction du temps de calcul accordé à l'obtention d'une solution. Une fois une localisation des ports obtenue, les différents chemins d'accès pour chaque cœur sont listés.

L'étape suivante consiste à trier les paquets de test de chaque cœur par temps de test décroissant. Or, comme chaque cœur possède un temps de test différent selon le port d'entrée par lequel sont acheminés ses vecteurs de test, les paquets sont triés par temps de test probable décroissant en prenant en compte le nombre de flits du paquet sans compression.

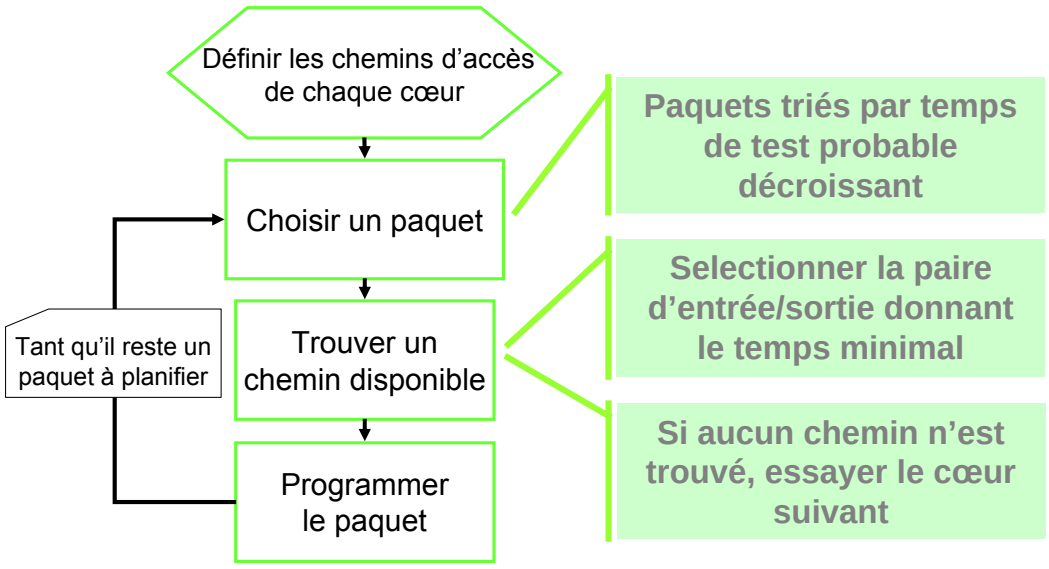


Figure 4.11 : Algorithme d'ordonnancement de test non-préemptif

L'ordonnancement est exécuté en commençant par le premier paquet de la liste (celui avec le temps de test probable le plus grand). La paire de ports d'entrée/sortie augmentant le moins le temps de test total est sélectionnée parmi les paires disponibles. Un second critère est utilisé pour arbitrer les choix possibles: il s'agit de maintenir un équilibre entre les ports (par exemple, ne pas planifier un cœur sur un port possédant déjà un ou des cœurs planifiés alors qu'un autre port n'en possède aucun). Même si sur ce pas de l'algorithme, le choix n'influe pas sur le temps de test total, un mauvais équilibrage risque grandement d'être pénalisant sur les prochains pas.

Rappelons ici que la principale caractéristique du test non-préemptif est de bloquer un chemin tant que les données (vecteurs et réponses) de test d'un cœur n'ont pas été intégralement transmises. Il est donc parfois possible qu'aucun chemin ne puisse être choisi. Par exemple, si le routeur d'un cœur est déjà occupé au transfert des données d'un autre cœur, ce cœur ne peut pas être testé tant que le routeur n'est pas libéré. Dans ce cas, le cœur est provisoirement abandonné, il sera traité dès qu'un chemin sera libre. L'opération est répétée tant qu'il reste des cœurs non planifiés.

Une fois tous les cœurs traités et planifiés, on obtient pour cette architecture (choix des ports d'entrée/sortie et nombre de broches associées à chaque port) un ordonnancement précis du test de tous les cœurs du système et le temps de test total de cette configuration. C'est ce temps qui va être comparé avec ceux des autres architectures afin de retenir au final, l'architecture et l'ordonnancement fournissant le meilleur temps de test total du système.

4.3.4.3. Exemple applicatif

Prenons un exemple pour illustrer le fonctionnement de cet algorithme d'ordonnancement. Considérons un système composé de 10 cœurs et dont les communications sont assurées par un NoC en grille 3x4. Plaçons-nous dans une configuration à 3 ports d'entrée de test alimentés respectivement par 10, 10 et 12 bits. Les trois paires de ports d'entrée/sortie sont localisées sur les

nœuds (3/9), (6/7) et (8/4). Les paquets de test sont classés par temps de test probable décroissant selon la liste: 6, 5, 4, 8, 10, 7, 3, 9, 2, 1. Cette configuration est reportée sur la Figure 4.12.

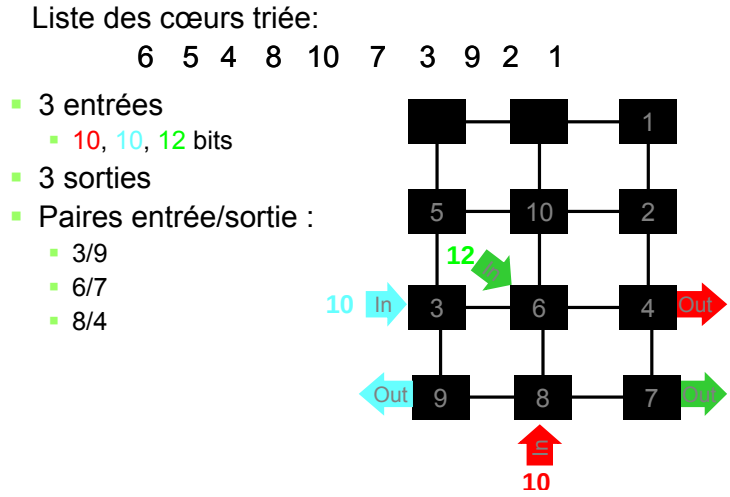


Figure 4.12 : Ordonnancement à l'état initial

Le premier cœur (ou paquet de test de ce cœur) est traité. Rappelons que le temps de test d'un cœur dépend du port et du chemin utilisé pour acheminer le paquet de test. Sur l'exemple de la Figure 4.13, le temps est de 9869 cycles sur un port de 12 bits ou de 10856 sur un port de 10 bits (ici le temps est identique sur les deux ports puisque le cœur 6 est à la même distance du premier port sur le cœur 8 et du deuxième port sur le cœur 3). Le paquet de test du cœur 6 est planifié sur le port impliquant le temps de test global le plus petit. Dans notre exemple, il s'agit du troisième port situé sur le cœur 6.

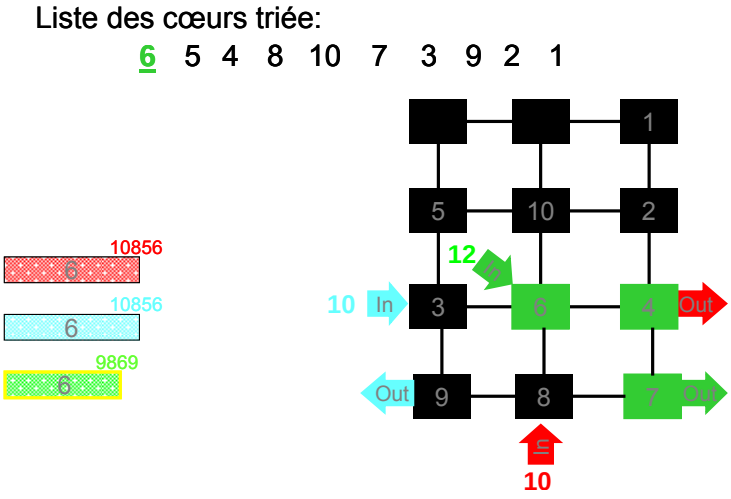


Figure 4.13 : Traitement du cœur 6

Le cœur 6 est donc planifié sur le troisième port et le temps de test total est mis à jour sur le temps maximum de chaque port, en l'occurrence le troisième, à 9869 cycles.

Le deuxième cœur à traiter est le numéro 5. Comme pour le cœur 6, le temps varie selon le port et le chemin utilisé pour acheminer le paquet de test. On remarquera cette fois que les temps

sur les premier et deuxième ports sont différents. En effet, le cœur 5 est à une distance de 3 nœuds du premier port et à une distance de 1 nœud du deuxième. Le temps pour acheminer le paquet à partir du premier port (6850 cycles) est donc logiquement supérieur à celui obtenu en l'acheminant à partir du deuxième (6826 cycles). L'algorithme choisit de planifier le cœur sur le port augmentant le moins le temps de test total (ce qui exclu le troisième port) et maintenant un équilibre sur les temps des autres ports. Le cœur 5 est donc logiquement planifié sur le deuxième port (Figure 4.14).

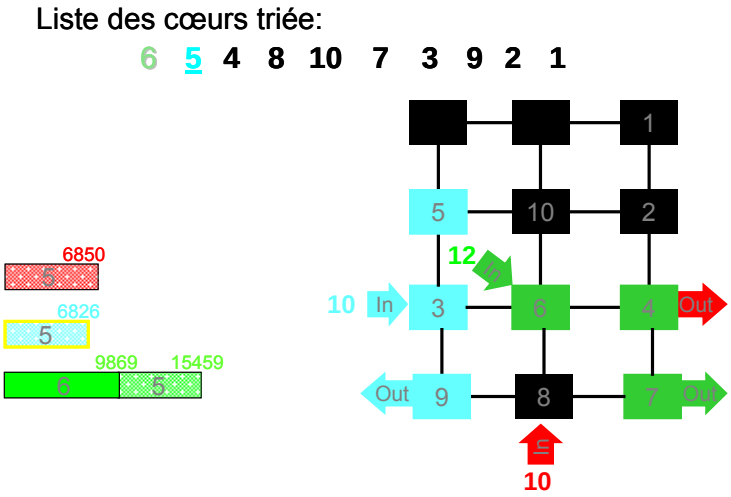


Figure 4.14 : Traitement du cœur 5

La même opération est répétée pour le cœur 4. La condition sur le temps de test total exclut à nouveau le troisième port. Afin de maintenir l'équilibre sur les temps de test des autres ports, le cœur 4 est finalement planifié sur le premier port (Figure 4.15).

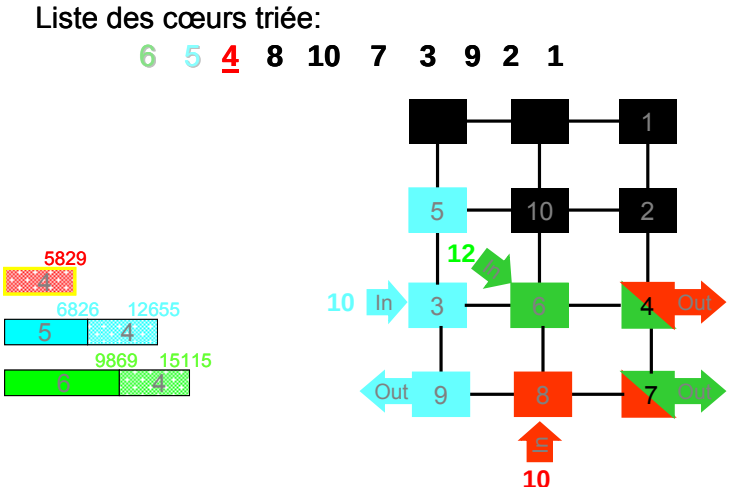


Figure 4.15 : Traitement du cœur 4

Ainsi, pas à pas, les cœurs 8, 10, 7 et 3 sont successivement planifiés (Figure 4.16). Intéressons nous maintenant au cas du cœur 9. Il apparaît dans la liste après le cœur 3. Cependant, Il n'existe aucun chemin disponible pour son test à cette étape de l'algorithme. Le cœur 9 est donc mis en attente, il sera traité dès que possible.

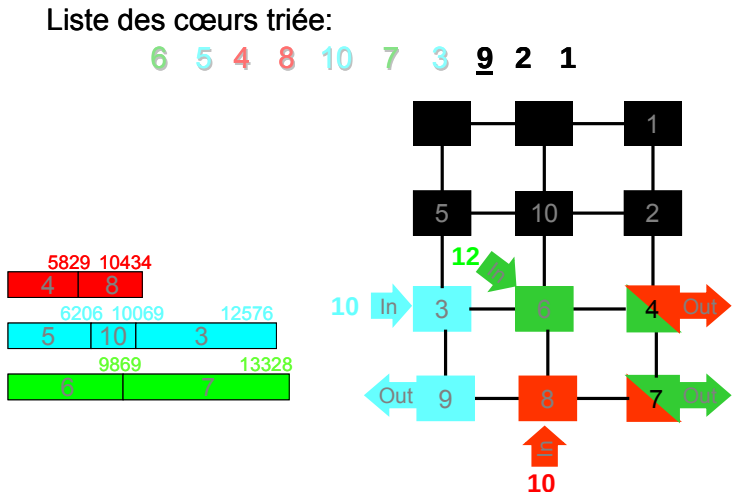


Figure 4.16 : Traitement du cœur 9 impossible

Comme il n'y a toujours aucun chemin disponible pour tester le cœur 9, les cœurs 2 puis 1 sont traités. Finalement, le cœur 9 peut être planifié sur le deuxième port. On peut observer le résultat final de l'ordonnancement sur la Figure 4.17. Le temps de test final obtenu pour cette configuration (3 ports d'entrée sur les nœuds 8, 3 et 6 de respectivement 10, 10 et 12 bits) est de 13412 cycles.

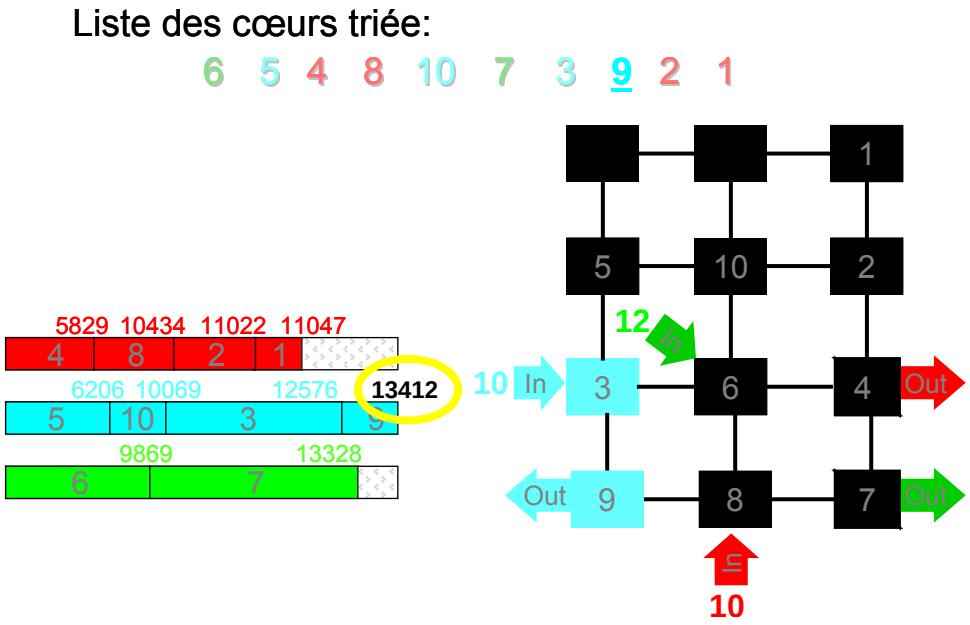


Figure 4.17 : Résultat final de l'ordonnancement

4.4. Résultats

4.4.1. Procédure expérimentale

Nous avons appliqué cette méthode sur plusieurs systèmes extraits de ITC'02 [Benc02] ont certains détails sont donnés en Annexe B. Nous reportons ici les résultats obtenus sur les systèmes d695, g1023 et p93791. Pour les mêmes raisons que celles citées dans le Chapitre 3, les séquences

de test ont été obtenues par génération aléatoire, en tenant compte du nombre de vecteurs reportés dans [Benc02] avec 80% de valeurs non spécifiées (ce pourcentage étant bien en dessous de la réalité des circuits commerciaux).

Les communications sont assurées par le réseau sur puce SOCIN développé à l'UFRGS[2] [Zefe03]. La topologie de ce type de NoC est une grille avec des canaux de communications bidirectionnels de 32 bits. Rappelons que les wrappers des cœurs sont adaptés au réseau, c'est-à-dire d'une taille maximale de 32 bits. Pour un circuit avec moins de 32 chaînes de scan, la taille du wrapper est fixée au nombre de chaînes de scan.

Le choix de la technique de compression repose sur les critères déjà évoqués dans les chapitres précédents: indépendance vis-à-vis de la netlist, indépendance vis-à-vis des données de test, non recours à des outils de test tels que des générateurs de vecteurs ou des simulateurs de fautes, possibilité de partage d'un décompresseur pour plusieurs cœurs différents. N'importe quelle technique répondant à ces critères peut être utilisée ici. Pour les mêmes raisons de simplicité évoquées dans le Chapitre 3, nous avons utilisé la technique de compression présentée dans le Chapitre 2.

4.4.2. Résultats et analyse

Le premier exemple étudié ici concerne le système d695 tiré de ITC'02. Il s'agit d'un système de 10 cœurs, avec 32 broches d'entrée et 32 broches de sortie. Nous présentons tout d'abord une configuration arbitraire de répartition des 32 broches d'entrée en 3 ports de test, respectivement 12, 10 et 10 bits. Cette configuration ne prétend pas à l'optimalité du résultat mais elle permet de bien comprendre les bénéfices de cette méthode. Les résultats sont donnés sur le Tableau 4.1.

Configuration	Temps de test (cycles)	# canaux ATE et # broches
1 port d'entrée (32 bits) sans compression	36588	32
3 ports d'entrée (3x32 bits) sans compression	15293 (-58.2%)	96 (+200%)
3 ports d'entrée (12, 10, et 10 bits) avec compression	24395 (-33.3%)	32 (+0%)

Tableau 4.1 : Configuration 3 ports (12, 10 et 10 bits) sur le système d695

La première ligne de ce tableau indique un temps de test total de ce système de 36588 cycles pour une configuration avec un seul port de test utilisant la totalité des 32 broches d'entrée sans compression. La deuxième ligne montre qu'en augmentant le nombre de ports de test sans compression (3 ports de 32 bits chacun donc 96 bits d'entrée et 96 canaux ATE) on parvient à réduire de manière significative le temps de test total du système. Cela confirme, une fois de plus,

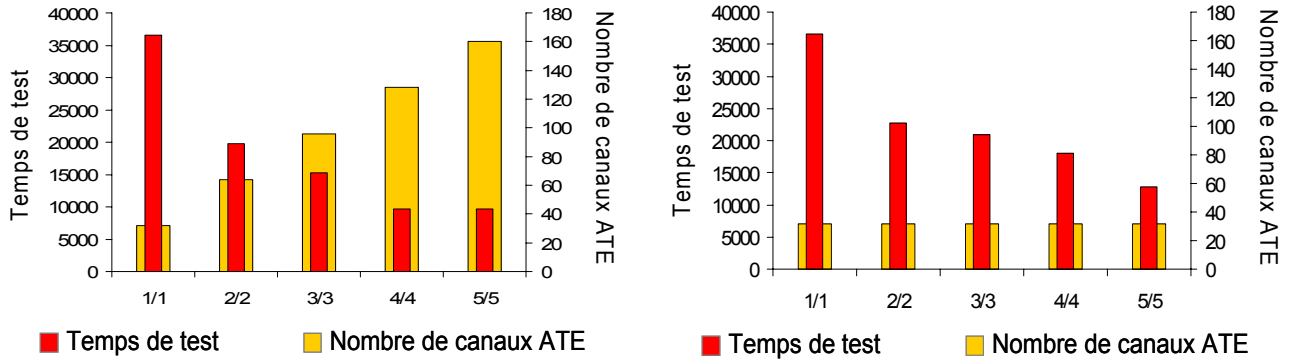
[2] Universidade Federal do Rio Grande do Sul, Université d'état de Porto Alegre au Brésil (http://www.ufrgs.br)

que la réduction du temps de test passe par le parallélisme. Cependant, cette réduction se fait au prix d'une sévère augmentation (200%) du nombre de canaux testeur et des broches d'entrée sur le système. Cela engendre des coûts supplémentaires en termes de matériel de test externe et sur le design du système lui-même (tripler le nombre de broches d'entrée d'un système n'est pas anodin).

Par contre, sur la troisième ligne, on remarque que l'utilisation d'une technique de compression, entraîne une réduction de 33% du temps de test total pour un coût quasiment nul (la surface nécessaire aux 3 décompresseurs est négligeable devant la taille du système). Cela illustre parfaitement le fait que, malgré une légère augmentation des temps de test individuels des cœurs, l'augmentation du parallélisme induite par l'utilisation de la compression horizontale permet effectivement de diminuer le temps de test total d'un système.

De plus, ce résultat correspond à une configuration particulière. L'exploration architecturale effectuée par notre algorithme permet d'atteindre des réductions de temps de test total bien plus importantes, pour un coût en canaux testeur constant et sans recours à des broches additionnelles, comme le montrent les résultats ci-dessous.

Passons maintenant au résultat de l'exécution complète de l'algorithme d'ordonnancement de test non-préemptif avec compression. Sur la Figure 4.18 (a), sans compression, on constate bien que le temps de test diminue avec l'augmentation du nombre de ports de test mais au prix d'une augmentation en nombre de canaux testeur (et de broches d'entrées) requis pour le test de ce système.



(a) Sans compression (b) Avec compression

Figure 4.18 : Temps de test vs nombre de ports pour le système d695

Le tableau ci-dessous donne les résultats complets pour le système d695. L'exploration architecturale est effectuée depuis 1 seul port de test jusqu'à 5 ports. On peut remarquer que pour un budget donné de 32 canaux testeur (première ligne sans compression, 1 port de test), notre technique apporte (avec la configuration à 5 ports de test, dernière ligne avec compression) une réduction du temps de test total de 65%.

Il est également intéressant de remarquer que pour un temps de test à peu près équivalent (deuxième ligne (2 ports) sans compression, comparée à la troisième (3 ports) ou quatrième ligne (4 ports) avec compression) notre solution permet une économie de 50% des canaux testeur requis

pour l'acheminement des vecteurs de test. Notons de plus que notre solution permet de toujours rester dans les limites des broches disponibles sur le système sans n'avoir jamais besoin de recourir à l'ajout de broches supplémentaires.

Système	# de ports	Sans Compression		Avec Compression		
		Temps de test (cycles)	# de canaux ATE	Temps de test (cycles)	# de canaux ATE	Meilleure partition
d695 (32 entrées)	1/1	**36588**	**32**	n/a	n/a	n/a
	2/2	**19788**	**64**	22737	32	4/28
	3/3	15293	96	20945	32	2/15/15
	4/4	9652	128	**18067**	**32**	**1/1/4/26**
	5/5	9652	160	**12853**	**32**	**1/1/1/2/27**

Tableau 4.2 : Résultats complets pour le système d695

On retrouve le même comportement avec les résultats obtenus sur le système g1023 (14 cœurs et 56 broches d'entrée). Notre méthode apporte une réduction à la fois du temps de test de 38% et des besoins en canaux testeur de 20% (première ligne (2 ports) sans compression comparée à la troisième ligne (4 ports) avec compression). On remarque aussi qu'à temps de test équivalent (troisième ligne (4 ports) avec et sans compression) notre technique apporte une réduction de 60% des besoins en canaux testeur.

Il apparaît également, qu'au-delà de 4 ports de test, le temps de test ne diminue plus. Cette saturation peut s'expliquer par le fait qu'il y a un cœur dominant dans le système qui utilise un port de test durant toute la période du test et qui impose son temps de test au reste du système. Aucune amélioration ne pourra être apportée avec un NoC 32 bits. De plus on note que lors de l'utilisation de la compression, il y a toujours un port utilisant la totalité des 32 bits du canal du NoC. Ceci se traduit par l'absence de décompresseur sur ce port de test. C'est évidemment sur ce port qu'a été placé le cœur dominant pour limiter son impact sur le temps de test total.

On observe aussi que pour certaines configurations, la totalité des broches disponibles n'est pas utilisée. Ceci s'explique par le fait que l'algorithme d'exploration sauvegarde la première solution valide qui réduit le temps de test total (l'utilisation des broches restantes ne permet pas de réduire ce temps). Ces broches inutilisées pourraient servir ultérieurement pour combiner notre méthode avec, par exemple, un autre format de paquet de données.

Système	# de ports	Sans Compression		Avec Compression		
		Temps de test (cycles)	# de canaux ATE	Temps de test (cycles)	# de canaux ATE	Meilleure partition
g1023 (56 entrées)	2/2	**23777**	**64**	25453	52	20/32
	3/3	16051	96	18883	56	12/12/32
	4/4	**14453**	**128**	**14865**	**50**	**6/6/6/32**
	5/5	14453	160	14865	56	6/6/6/6/32

Tableau 4.3 : Résultats complets pour le système g1023

Les derniers résultats concernent le système p93791 qui inclut 32 cœurs pour 118 broches d'entrée. Notons que dans ce système, les cœurs sont répartis sur deux niveaux. Cela signifie que certains cœurs ne sont pas directement connectés au NoC. Les données passent par le wrapper du cœur de niveau 1 et sont ensuite transmises au cœur de niveau 2 qui, lui, ne possède pas de wrapper propre. Cette hiérarchie, dont une illustration est donnée Figure 4.19, est respectée dans notre procédure expérimentale lors du transfert des paquets de test. Sur cet exemple, les données envoyées lors du test du cœur 4 passent par le wrapper du cœur 3. Elles suivent donc le même chemin et passent par le même décompresseur.

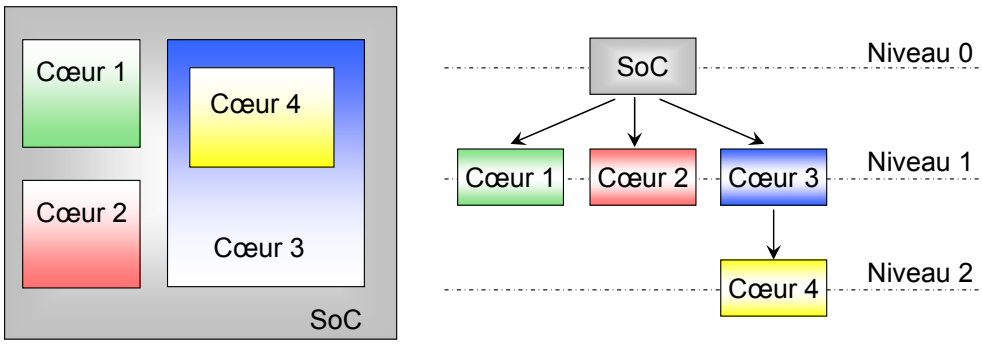


Figure 4.19 : Hiérarchie des cœurs dans un système su puces

L'ensemble des résultats du système p93791 est reporté sur le Tableau 4.4. On observe, comme sur le système g1023, une saturation à partir de 5 ports de test (qui est toujours due à un cœur dominant) et la non utilisation de la totalité du budget des 118 canaux testeur. Bien qu'il puisse sembler, à première vue, que les résultats obtenus par notre méthode soient moins bons en termes de temps de test, avec un budget maximum de 118 canaux testeur, notre méthode (avec 5 ports, quatrième ligne) apporte une réduction de 32% du temps de test total du système par rapport à une technique sans compression (3 ports, deuxième ligne). Même en ajoutant 10 broches de test sur le système utilisant une technique sans compression (4 ports, troisième ligne), notre méthode est toujours meilleure de 14% tout en requérant 17% de canaux testeur en moins.

Système	# de ports	Sans Compression		Avec Compression		
		Temps de test (cycles)	# de canaux ATE	Temps de test (cycles)	# de canaux ATE	Meilleure partition
p93791 (118 entrées)	2/2	593225	64	593225	64	32/32
	3/3	**395818**	**96**	458164	66	2/32/32
	4/4	311520	128	332843	101	5/32/32/32
	5/5	244550	160	**268459**	**106**	5/5/32/32/32
	6/6	244550	192	268459	106	5/5/5/32/32/32
	7/7	244550	224	268459	106	5/5/5/5/32/32/32

Tableau 4.4 : Résultats complets pour le système p93791

4.5. Conclusion

Nous avons proposé dans ce chapitre une méthode innovante de test des cœurs de systèmes sur puce dont les communications sont assurées par un réseau sur puce (NoC). Notre méthode est basée à la fois sur la réutilisation du NoC et sur la compression horizontale des données de test.

Le temps de test total de ce genre de système dépend essentiellement du nombre de ports de test qu'il est possible de créer en fonction des broches d'entrée de test présentes sur le système. En effet, plus il y a de ports de test, plus le parallélisme augmente, et par conséquent, plus le temps de test total diminue. La solution la plus simple pour augmenter ce nombre de ports, et ainsi, réduire le temps de test total du système, est de rajouter des broches de test et donc d'augmenter le nombre de canaux testeurs devant alimenter ces broches.

Nous avons montré qu'il était possible d'augmenter le nombre de ports de test d'un NoC sans avoir à augmenter le nombre de canaux testeur ni le nombre de broches d'entrée de test. La compression horizontale des données de test permet de diminuer de manière significative le nombre de broches nécessaires à la création d'un port de test, certes au prix d'une légère augmentation du temps de test des cœurs dont les données de test sont acheminées par ce port. Mais nous avons démontré que l'augmentation du parallélisme induite par l'augmentation du nombre de ports compense largement cette augmentation des temps de test individuels.

Nous avons illustré l'efficacité de notre méthode par des résultats expérimentaux obtenus sur des systèmes de référence ITC'02 dont les cœurs sont connectés au NoC SOCIN développé à l'université de Porto Alegre au Brésil. La combinaison de la technique de compression présentée dans le Chapitre 2 et d'un ordonnancement non-préemptif du test des cœurs de ces systèmes a confirmé la validité de notre méthode. Nous avons ainsi pu montrer que le temps de test total d'un système pouvait être réduit de manière significative (de 30 à 60% selon les systèmes) sans augmentation des ressources testeur (nombre de canaux ATE) ni du nombre de broches d'entrée du système, et cela pour un coût minime en surface de silicium pour les décompresseurs. On peut également aborder le problème dans l'autre sens et fournir une nette réduction du nombre de canaux testeur pour un temps de test total du système équivalent.

4.6. Références bibliographiques

[Beni02] L. Benini, G. De Micheli: Networks on chips: a new SoC paradigm. IEEE Computer, Vol 35, N° 1, 2002, pp. 70–80.

[Chap84] D. M. Chapiro: Globally-asynchronous locally-synchronous systems. Thèse de doctorat, Stanford University, 1984.

[Duat97] J. Duato, S. Yalamanchili, L. Ni: Interconnection networks: an engineering approach. IEEE Computer Society Press, 1997.

[IEEE05] http://grouper.ieee.org/groups/1500/

[Cota04] E. Cota, L. Carro, M. Lubaszewski: Reusing an on-chip network for the test of core-based systems. ACM Transactions on Design Automation of Electronic Systems, Vol. 9, N° 4, 2004, pp. 471-499.

[Liu06] C. Liu, Z. Link, Z.; D.K. Pradhan: Reuse-based test access and integrated test scheduling for network-on-chip. Proceedings of the Conference on Design, Automation and Test in Europe, DATE 2006, pp. 303-308.

[Cota06] E. Cota, C. Liu: Constraint-driven test scheduling for NoC-based systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 25, N° 11, 2006, pp. 2465-2478.

[Benc02] E.J. Marinissen, V. Iyengar, K. Chakrabarty: A set of benchmarks for modular testing of SOCs. Proceedings of the International Test Conference, ITC 2002, pp. 519-528. http://itc02socbenchm.pratt.duke.edu/

[Cota03] E. Cota, M. Kreutz, C. A. Zeferino, L. Carro, M. Lubaszewski, A. Susin: The impact of NoC reuse on the testing of core-based systems. Proceedings of the 21st VLSI Test Symposium, VTS 2001, pp. 128-134.

[Amor06] A.M. Amory, K. Goossens, E.J. Marinissen, M. Lubaszewski, F. Moraes: Wrapper design for the reuse of networks-on-chip as test access mechanism. Proceedings of the European Test Symposium, ETS 2006, pp. 213-218.

[Li08] J. Li , Q. Xu , Y. Hu , X. Li: Channel width utilization improvement in testing NoC-based systems for test time reduction. Proceedings of the 4th International Symposium on Electronic Design, Test and Applications, DELTA 2008; pp. 26-31.

[Liu05] C. Liu, V. Iyengar, J. Shi, E. Cota: Power-aware test scheduling in network-on-chip using variable-rate on-chip clocking. Proceedings of the 23rd Symposium on VLSI Test Symposium, VTS 2005, pp. 349-354.

[Amor04] A.M. Amory, E. Cota, M. Lubaszewski, F.G. Moraes: Reducing test time with processor reuse in network-on-chip based systems. Proceedings of the 17th Symposium on Integrated Circuits and System Design, SBCCI 2004, pp. 111-116.

[Mitr02] S. Mitra, K.S. Kim: X-compact, an efficient response compaction technique for test cost reduction. Proceedings of the International Test Conference, ITC 2002, pp: 311-320.

[Rajs05] J. Rajski, J. Tyszer, C. Wang, S.M. Reddy: Finite memory test response compactors for embedded test applications. IEEE Transaction on Computer-Aided Design, Vol. 24, N° 4, 2005, pp. 622- 634.

[Zefe03] C.A. Zeferino, A.A. Susin: SoCIN: a parametric and scalable network-on-chip. Proceedings of the 16th Symposium on Integrated Circuits and Systems Design, SBCCI 2003, pp. 169-174.

Conclusion

Il semble évident que les systèmes microélectroniques ont désormais envahi notre quotidien. Il est difficile de passer ne serait-ce que quelques heures sans y être confronté d'une manière ou d'une autre. Pour pénétrer le marché du grand public, les fabricants ont dû fournir des produits non seulement de qualité, en très grande quantité, mais à des tarifs abordables. Un des axes de recherche pour y parvenir a été de proposer des méthodes de test efficaces (augmentation de la qualité du produit final) tout en maîtrisant leur coût (diminution des besoins en termes d'équipement, en temps de test). Cet axe de recherche est identifié comme primordial dans le domaine de la microélectronique et se décline en de nombreuses voies selon le type de composant adressé (analogique, mixte, mémoire, MEMS, numérique). Pour ce qui concerne les composants numériques, une attention particulière doit être portée aux infrastructures de test permettant l'acheminement des données au sein des systèmes. En effet, si l'on peut aujourd'hui s'appuyer sur des outils efficaces pour ce qui est de la génération de données de test, il reste néanmoins problématique de transporter un grand nombre de données vers tous les cœurs numériques d'un système au travers d'interfaces relativement de plus en plus étroites.

La piste qui nous a particulièrement intéressés dans cette thèse s'appuie sur la possibilité de stocker les données de test de manière compressée et de les décompresser au moment de leur utilisation, c'est-à-dire avant d'être appliquées aux cœurs sous test. Les techniques de compression permettent de réduire fortement les besoins matériels (canaux testeurs), mais aussi et surtout de réduire le temps d'exécution du test. De très nombreuses techniques ont été proposées dans la littérature, certaines faisant d'ores et déjà même partie de suite de TCAD. Toutefois elles se cantonnent encore aujourd'hui au traitement des blocs numériques isolés sans prendre en compte la dimension système avec la nécessité de définir une infrastructure pour l'ensemble et la possibilité de partage de ressource.

Or la tendance actuelle est à l'intégration de systèmes complets (SoC) contenant sur une même puce des blocs numériques, des interfaces, des convertisseurs, de la mémoire, des processeurs, etc. Comme nous l'avons vu dans ce manuscrit, les techniques de compression utilisées pour tester les circuits intégrés numériques isolés ne sont pas adaptées au traitement de systèmes complets contenant éventuellement plusieurs de ces circuits. En effet, sommairement, tester un circuit consiste à le stimuler avec des données et à vérifier que les valeurs attendues sont bien présentes en sortie. Or, dans un système, les entrées / sorties des cœurs numériques ne sont plus

directement accessible. Le test de ces cœurs nécessite donc la mise en œuvre d'anneau d'isolation (IEEE 1500) et la définition d'une infrastructure de transport des données de test des entrées/sorties du système vers les entrées/sorties de ses cœurs. L'optimisation globale du système, où plutôt de son test pour ce qui nous intéresse ici, demande nécessairement de traiter conjointement l'infrastructure de transport des données (le bus de test TAM) et les infrastructures de décompression des données de test. D'autre part, les techniques de compression doivent tenir compte du paradigme des systèmes sur puce à base de cœurs où les informations nécessaires au développement de séquences de test ne son pas toujours accessibles à l'intégrateur du système. Plus récemment encore, l'apparition des architectures de systèmes sur puces à base de réseaux d'interconnexion (NoC) a posé le même problème de définition de l'infrastructure de test au niveau système.

C'est dans ce cadre général que se sont développés ses travaux de thèse avec d'une part la proposition d'une technique de compression simple (décompresseur faible coût), générique et surtout compatible avec le test des systèmes des méthodes de génération de données de test et de l'implantation structurelle du cœur sous test) et d'autre part une méthode d'optimisation pour la définition d'infrastructure de test au niveau système.

C'est en exploitant la présence d'un très grand nombre de valeurs non spécifiées dans les données de test des circuits, qu'il est possible de générer des séquences « compressibles ». La technique proposée ici permet de transférer au décodeur sur puce la différence arithmétique (par morceau) entre deux vecteurs de test consécutifs plutôt que les vecteurs de test eux-mêmes. Cette différence étant généralement nettement plus petite en nombre de bit que les vecteurs originaux, elle peut se coder sur un nombre réduit de bits. Nous avons présenté dans ce manuscrit des résultats expérimentaux obtenus à l'aide de notre technique sur des circuits ISCAS'89. Ces expériences montrent des gains pouvant atteindre 50% du volume de départ et 60% du temps de test initial par rapport à l'application d'une séquence de test optimisée (sans X) et donc non « compressible ».

Au niveau système, le but initial est essentiellement d'ordonnancer le test des différents cœurs qui composent le système et d'obtenir ainsi compromis intéressant entre temps de test minimal et infrastructure de test à faible coût. La compression horizontale permet de propager des données de taille réduite, de tester plus de cœurs en parallèle pour un budget donné (taille du bus de test), et donc de réduire le temps de test global du système. La méthode présentée dans ce manuscrit s'appuie conjointement sur l'ordonnancement du test des cœurs et sur une technique de compression horizontale. Bien que les résultats présentés dans ce manuscrit aient été obtenus avec la technique de compression développée au cours de cette thèse, toute autre technique de compression répondant aux contraintes du test des SoC aurait pu être utilisée à la place. Ces résultats montrent que notre méthode permet de diminuer de moitié le temps de test d'un système dont l'infrastructure de test s'appuie sur un bus.

Nous avons ensuite étendu l'application de la compression au domaine du test des systèmes intégrant un réseau sur puce (NoC). La réutilisation du NoC pour le transport des données de test est

d'autant plus intéressante que le réseau a un coût d'implantation important et qu'il est donc plus judicieux de réutiliser cette infrastructure pour le transport des données en mode test. Une fois encore la compression permet d'augmenter le nombre de cœurs pouvant être testé en parallèle. De la même façon que pour le test des SoC, nous avons développé une méthode combinant ordonnancement du test des cœurs du NoC et compression de données de test. Puisque les données de tes compressées peuvent être acheminée au travers d'interface réduite utilisant un faible nombre d'entrées / sorties, notre méthode permet de créer plus de chemin dans le NoC et donc tester encore plus de cœur en même temps. Les expériences montrent dans ce cas que notre méthode de compression appliquée au test des systèmes basés sur des NoC permet d'obtenir des réductions de temps de test allant de 30 à 60% selon les systèmes.

Au final, nous avons proposé une technique de compression des données de test simple, efficace, générique et compatible avec le test des systèmes sur puce (qu'ils soient basés sur une infrastructure à bus de test dédié ou sur un NOC). Nous avons ensuite montré que l'utilisation d'une telle technique de compression peut être extrêmement efficace en termes de réduction du temps de test total d'un système si elle est combinée intelligemment avec un ordonnancement du test de ces cœurs.

A court terme il serait intéressant d'étudier précisément l'impact de l'ajout de la compression dans le cadre des systèmes basés sur des NoC, par exemple sur les modifications à apporter à la conception des wrappers ou sur la surface additionnelle nécessaire à l'implantation des décompresseurs. Il pourrait également être intéressant, lors de la recherche d'architecture de test des SoC, de proposer d'autres heuristiques d'optimisation permettant une vraie conception conjointe anneau d'isolation/mécanisme d'accès/compresseur. A plus long terme, une piste de recherche serait d'étudier l'application d'une technique de compression au nouveau paradigme de conception des systèmes microélectroniques que sont les puces 3D. D'autre part, les aspects de faible consommation électrique étant de plus en plus présent dans les étapes de conception, l'étude d'une technique de compression compatible avec le test des systèmes tout en étant orientée vers la réduction de la puissance dissipée lors du test pourrait faire l'objet de travaux futurs.

Liste des Publications

2006 J. Dalmasso, M-L. Flottes, B. Rouzeyre: Fitting ATE Channels with Scan Chains: a Comparison between a Test Data Compression Technique and Serial Loading of Scan Chains. Proceedings of the Third International Workshop on Electronic Design, Test and Applications, DELTA 2006, pp. 295-300.

J. Dalmasso: Compression de Données de Test : Réduction du Nombre de Broches et Gain en Temps de Test. 9ièmes Journées Nationales du Réseau Doctoral de Microélectronique, JNRDM 2006.

2007 J. Dalmasso, M-L. Flottes, B. Rouzeyre: Test data compression and TAM design. Proceedings of the International Conference on Very Large Scale Integration of System-on-Chip, VLSI-SoC 2007, pp. 178-184.

J. Dalmasso: Systems-on-Chip: Use of test data compression technique to reduce test time. Proceedings of the 3rd Conference on Ph.D. Research in Microelectronics and Electronics, PRIME 2007.

J. Dalmasso, M-L. Flottes, B. Rouzeyre: TAM design and test data compression for SoC test cost reduction. Informal digest of Papers of the 12th IEEE European Test Symposium, ETS 2007.

E. Cota, J. Dalmasso, M-L. Flottes, B. Rouzeyre: Improving NoC-based testing through compression schemes. Workshop on Diagnostic Services in Network-on-Chips - Test, Debug, and On-Line Monitoring at the Conference on Design, Automation and Test in Europe, DATE 2007.

2008 J. Dalmasso, E. Cota, M-L. Flottes, B. Rouzeyre: Improving the Test of NoC-Based SoCs with Help of Compression Schemes. Proceedings of the IEEE Computer Society Annual Symposium on VLSI, ISVLSI 2008, pp. 139-144.

2009 J. Dalmasso, M-L. Flottes, B. Rouzeyre: Compression-based SoC Test Infrastructures. VLSI-SoC: Advanced Topics on Systems on a Chip, Vol 291, IFIP Series, Editors R.Reis, V. Mooney, P.Hasler (Boston: Springer), pp. 53-67.

Table des Figures

Figure 1.1 : Wafer dont les circuits défectueux sont marqués 16
Figure 1.2 : Structure de test 20
Figure 1.3 : Testeur Agilent 93000 20
Figure 1.4 : Connexion du testeur au circuit sous test 20
Figure 1.5 : Loi de Moore 21
Figure 1.6 : Circuit séquentiel 23
Figure 1.7 : Bascule D modifiée pour le test 24
Figure 1.8 : exemple de circuit avec chaîne de scan 24
Figure 1.9 : Circuit sous test avec chaîne de scan unique 25
Figure 1.10 : Exemple de données de test: 1 seule chaîne de scan 25
Figure 1.11 : Circuit sous test avec chaînes de scan multiples 26
Figure 1.12 : Exemple de données de test: 2 chaînes de scan 27
Figure 1.13 : Architecture de test BIST 28
Figure 1.14 : Architecture de Test avec compression 29
Figure 2.1 : Architecture BIST générale 36
Figure 2.2 : Architecture BIST STUMPS 37
Figure 2.3 : Architecture BIST hybride 39
Figure 2.4 : Test aléatoire pondéré interne 40
Figure 2.5 : Test Aléatoire pondéré par le testeur 41
Figure 2.6 : Test Aléatoire modifié par "Bit-Flipping" 42
Figure 2.7 : Test intégré par procédé arithmétique 43
Figure 2.8 : Test intégré par réutilisation d'une structure sécurisée AES 44
Figure 2.9 : Arbre correspondant au codage de Huffman 46
Figure 2.10 : Arbre correspondant au codage de Huffman 46
Figure 2.11 : Exemple de décodage 3bits pour un code RLE 48
Figure 2.12 : Sérialisation des données de test. 51

Figure 2.13 : Switch reconfigurables pour 2 entrées et 4 chaînes de scan 53
Figure 2.14 : Exemple de chaînes de scan compatible (a) et incompatible (b) 53
Figure 2.15 : Switch élémentaire du réseau oméga 53
Figure 2.16 : Architecture Circular Scan 54
Figure 2.17 : Bascule Pour le Circular Scan modifié 55
Figure 2.18 : Architecture Circular Scan Modifiée 55
Figure 2.19 : Architecture Parallèle-Série Full Scan (PFFS) 56
Figure 2.20 : Architecture RSSA de la technologie "Adapative Scan" 58
Figure 2.21 : Reconfiguration dynamique de l'architecture RSSA 59
Figure 2.22 : Décompresseur VirtualScan de SynTest 60
Figure 2.23 : Séquence LFSR et réseau de décompression associé 61
Figure 2.24 : Décompresseur linéaire reconfigurable 62
Figure 2.25 : Structure combinée décompresseur séquentiel et encodage statistique 63
Figure 2.26 : Méthode de réinitialisation dynamique d'un décompresseur linéaire séquentiel 64
Figure 2.27 : Technologie EDT de Mentor Graphics® 65
Figure 2.28 : Générateur en anneau implémenté à partir du polynôme $x^{32}+x^{18}+x^{14}+x^{9}+1$ 65
Figure 2.29 : Schéma général de compression par dictionnaire 66
Figure 2.30 : Format des données du codage par dictionnaire avec correction 67
Figure 2.31 : Architecture de test pour un codage par dictionnaire avec une correction 67
Figure 2.32 : Structure du décompresseur pour 4 chaînes de scan et 2 entrées testeur 71
Figure 2.33 : Compression par codage des différences 72
Figure 2.34 : Nouvelle structure du décompresseur sur puce 74
Figure 2.35 : Architecture détaillée de notre décompresseur 75
Figure 2.36 : Modes de fonctionnements du décompresseur et signaux de contrôle 76
Figure 2.37 : Etat initial du décompresseur et des chaînes de scan 76
Figure 2.38 : Initialisation de la première slice S_1 (cycle 1) 77
Figure 2.39 : Initialisation de la première slice S_1 (cycle 2) 77
Figure 2.40 : Initialisation de la première slice S_1 (cycle 3) 78
Figure 2.41 : Envoi de la seconde slice S_2 (Cycle 4) 78
Figure 2.42 : Envoi de la troisième slice S_3 (Cycle 5) 79
Figure 2.43 : Chargement de la slice S_4 (Cycle 6) 79
Figure 2.44 : Chargement de la slice S_4 (Cycle 7) 80
Figure 2.45 : Chargement de la slice S_4 (Cycle 8) 80
Figure 2.46 : Activation du circuit et capture des réponses (Cycle 9) 81

Figure 2.47 : Récupération des réponses dans les chaînes de scan 81
Figure 2.48 : Expulsion des réponses et chargement des nouvelles slices 82
Figure 2.49 : Exemple de séquence de test compressée 82
Figure 2.50 : Assignation des X lors de l'initialisation d'une slice 83
Figure 2.51 : Assignation des X lors du choix de la slice compressée 84
Figure 2.52 : Impact de l'assignation des X sur la séquence compressée 84
Figure 2.53 : Ordonnancement des colonnes 85
Figure 2.54 : Algorithme de compression 86
Figure 2.55 : Sérialisation des données de test 88
Figure 2.56 : Résultats de la compression pour la configuration 32 chaînes de scan 90
Figure 2.57 : Résultats de la sérialisation pour la configuration 32 chaînes de scan 91
Figure 2.58 : Gains en temps de test et volume pour la configuration 32 chaînes de scan 92
Figure 3.1 : Exemple de système sur puces contenant 6 cœurs 101
Figure 3.2 : Architecture de test des systèmes IEEE 1500 102
Figure 3.3 : Test en série des cœurs d'un système 103
Figure 3.4 : Test en parallèle des cœurs d'un système 103
Figure 3.5 : Différentes approches de l'ordonnancement du test 104
Figure 3.6 : Conception du wrapper 106
Figure 3.7 : Positions des décompresseurs 108
Figure 3.8 : Concaténation de 2 séquences de test T1 et T2 pour le test en parallèle 109
Figure 3.9 : Concaténation de 2 séquences de test T1 et T3 pour le test en parallèle 110
Figure 3.10 : Exemple d'architecture de test avec compression 111
Figure 3.11 : Architecture vise et variables à déterminer 112
Figure 3.12 : Algorithme général de recherche d'architecture de test 113
Figure 3.13 : Temps de test vs. taille du wrapper du $10^{ème}$ cœur du système D695 (ITC'02) 114
Figure 3.14 : Comparaison entre calcul et estimation du temps de test du circuit S38417 115
Figure 3.15 : Exemple de découpage ATE / TAM 116
Figure 3.16 : algorithme d'assignation des cœurs et d'ordonnancement du test 117
Figure 3.17 : Solution initiale 118
Figure 3.18 : Recherche du bus avec le temps de test le plus grand 118
Figure 3.19 : Déplacement virtuel du premier cœur sur les autres bus 119
Figure 3.20 : Déplacement virtuel du deuxième cœur sur les autres bus 119
Figure 3.21 : Mémorisation du choix donnant le meilleurs temps de test total 120
Figure 3.22 : Nouvelle recherche du bus avec le temps de test le plus grand 120

Figure 3.23 : Déplacement virtuel du premier cœur sur les autres bus 121
Figure 3.24 : Système n°1 à base de cœurs ISCAS 89 121
Figure 3.25 : Architecture retenue pour le système n°1 124
Figure 3.26 : Ordonnancement du test pour le système n°1 124
Figure 4.1 : Réseau sur puce 135
Figure 4.2 : Détail du réseau sur puce 136
Figure 4.3 : Réutilisation du NoC pour le test 137
Figure 4.4 : Format des données de test dans le NoC 138
Figure 4.5 : Temps de test vs nombre de ports de broches additionnelles 139
Figure 4.6 : Temps de test vs nombre de canaux ATE 139
Figure 4.7 : Insertion d'un décompresseur en entrée du système 142
Figure 4.8 : Insertion d'un décompresseur en sortie du système 143
Figure 4.9 : Format des données lors de la compression 144
Figure 4.10 : Algorithme d'exploration architecturale 146
Figure 4.11 : Algorithme d'ordonnancement de test non-préemptif 147
Figure 4.12 : Ordonnancement à l'état initial 148
Figure 4.13 : Traitement du cœur 6 148
Figure 4.14 : Traitement du cœur 5 149
Figure 4.15 : Traitement du cœur 4 149
Figure 4.16 : Traitement du cœur 9 impossible 150
Figure 4.17 : Résultat final de l'ordonnancement 150
Figure 4.18 : Temps de test vs nombre de ports pour le système d695 152
Figure 4.19 : Hiérarchie des cœurs dans un système su puces 154

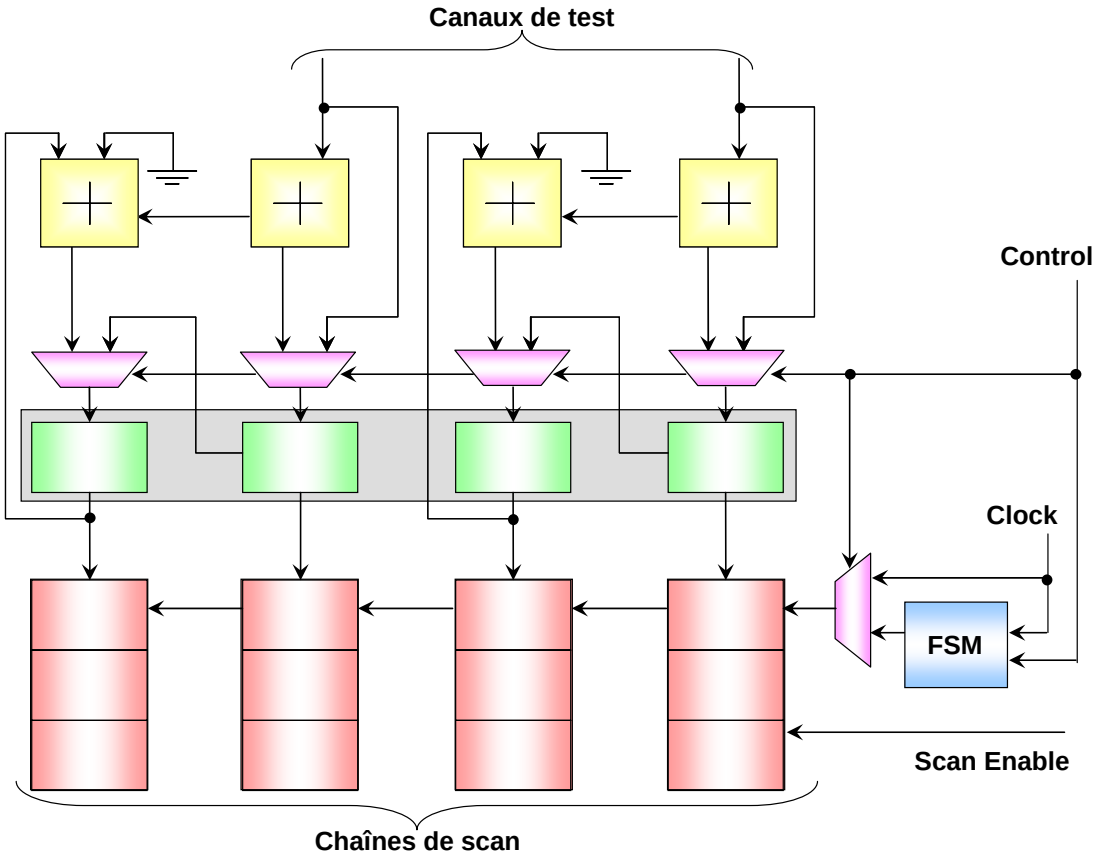
Table des Tableaux

Tableau 1.1 : Compaction de vecteurs 18
Tableau 2.1 : Comparaison entre LFRS et AES utilisé en générateur de vecteur 44
Tableau 2.2 : Comparaison entre code de Huffman et code de Huffman modifié 47
Tableau 2.3 : Exemple de code FDR 49
Tableau 2.4 : Exemple de code de Golomb pour m=4 49
Tableau 2.5 : Résultats obtenus pour le circuit s5378 90
Tableau 2.6 : Résultats d'autres circuits ISCAS'89 93
Tableau 3.1 : Caractéristiques des cœurs du système n°1 122
Tableau 3.2 : Caractéristiques des cœurs du système n°2 122
Tableau 3.3 : Résultats pour le système n°1 123
Tableau 3.4 : Comparaison avec les architectures standards 125
Tableau 3.5 : Comparaison avec les architectures standards pour le système n°1 126
Tableau 3.6 : Comparaison avec les architectures standards pour le système n°2 126
Tableau 4.1 : Configuration 3 ports (12, 10 et 10 bits) sur le système d695 151
Tableau 4.2 : Résultats complets pour le système d695 153
Tableau 4.3 : résultats complet pour le système g1023 153
Tableau 4.4 : Résultats complets pour le système p93791 154

Annexe A

Estimation du nombre de portes logiques nécessaires à la synthèse d'un décompresseur pour N chaînes de scan et M canaux de test.

- Additionneurs (1 par chaîne de scan)
  - sous le canal de test: 2 xor, 2 and, 1 or
  - à l'intérieur d'un étage : 1 xor, 1 and
    - ⇨ Nb de portes = $5 \cdot M + 2(N - M)$
- Multiplexeurs 1 bit 2 voies (1 par chaîne de scan) : 1 not, 2 and, 1 or
  - pour N chaînes de scan
    - ⇨ Nb de portes = $5 \cdot N$
- Registre: 1 bascule par chaîne de scan = 5 and, 1 not
  - pour N chaînes de scan
    - ⇨ Nb de portes = $6 \cdot N$
- Contrôleur chaînes de scan
  - Multiplexeur 1 bit 2 voies : 1 not, 2 and, 1 or
    - ⇨ 5
  - Compteur jusqu'à K=$\lceil N / M \rceil$ sur C = $\lceil \log_2(K) \rceil$ bits : C bascules (6*C portes), C and
    - ⇨ $7 \cdot C$
  - Comparateur C bits : C xor, (C-1) and
    - ⇨ $2 \cdot C - 1$
- Total (pour M canaux de test et N chaînes de scan)
    - ⇨ Total = $18 \cdot N + 3 \cdot M + 9 \cdot C + 4$



Architecture détaillée du décompresseur sur puce

		Nombre de canaux de test M						
		1	2	4	8	16	32	64
Nombre de chaînes de scan N	2	56						
	4	101	95	101				
	8	182	176	173				
	16	335	329	326	329			
	32	632	626	623	626	641		
	64	1217	1211	1208	1211	1226	1265	
	128	2378	2372	2369	2372	2387	2426	2513

Evaluation du nombre de portes logiques du décompresseur sur puce

Annexe B

Descriptif rapide des benchmarks circuits ITC'89 et systèmes sur puces ITC'02.

- **Circuits ISCAS'89**

Circuits	# entrées	# sorties	# portes	# bascules
s5378	35	49	2779	179
s9234	36	39	5597	211
s13207	62	152	7951	638
s15850	77	150	9772	534
s35932	35	320	16065	1728
s38417	28	106	22179	1636
s38584	38	304	19253	1426

- **Systèmes sur puce ITC'02**

SoC	Nombre de						Contributeur
	Modules	Tests	Niveaux	I/Os	FFs	Vecteurs	
u226	10	9	2	376	1040	5148569	Universidade Federal do Rio Grande do Sul
d281	9	15	2	2931	882	8818	Duke University
d695	11	10	2	1845	6384	881	Duke University
h953	9	8	2	929	4657	1100	National Tsing Hua University
g1023	15	14	2	3707	1546	2349	University of Stuttgart
f2126	5	4	2	1597	13996	962	Faraday Technologies
q12710	5	4	2	13167	12991	4612	Hewlett-Packard
p22810	29	30	3	4283	24723	24890	Philps Semiconductors
p34392	20	21	3	2057	20948	66349	Philps Semiconductors
p93791	33	32	3	6943	89973	22987	Philps Research
t512505	32	31	2	8663	68051	10479	Texas Instruments
a586710	8	7	3	3755	37656	10850894	Analog Devices

➢ Exemple de détails pour le système g1023 tiré de ITC'02

Circuits	# Bascules	# Vecteurs	# Chaînes de scan
0	0		
1	592	134	14
2	167	74	2
3	53	57	1
4	216	268	4
5	127	51	4
6	94	36	2
7	94	34	2
8	104	31	2
9	64	68	1
10	13	29	1
11	9	15	1
12	13	16	1
13	0	512	0
14	0	1024	0

Index

A

Adaptive Scan · 57
AES · 43
Architecture RSSA · 57
Architecture Scan · 22
ATE · 20
ATPG · 18
Automatic Test Pattern Generation · 18

B

BIST · 27, 36, 68
BIST non LFSR · 42
Bit-flipping · 41
Boundary Scan · 23
Broadcasting · 52
Built-In Self Test · 27

C

Canaux testeur · 21, 101
Chaîne de scan unique · 24
Chaînes de scan · 23, 101
Chaînes de scan multiples · 25
Circuit sous test · 20
Circuits séquentiels · 22
Circular scan · 54
Codages Statistiques · 45
Cœurs · 69, 101
Collage · 17
Compression de données · 27
Compression horizontale · 22, 28, 50, 68, 94, 101, 107, 108, 109, 141, 142, 144
Compression par dictionnaire · 65
Compression verticale · 22, 28, 45, 68, 107
Cube de test · 18
CUT · 20

D

Décompresseur linéaire · 60
Décompresseur sur puce · 73
Défauts · 16
Design reuse · 69
DFT · 19, 35
DFTMAX® · 57

E

EDT · 64
Embedded Deterministic Test · 64
Erreur · 16

F

Faute logique · 17
FDR · 48
Flits · 136
Frequency-Directed Run-length · 48

G

GALS · 133
Golomb · 49

H

Huffman (codage) · 45

I

IEEE 1500 · 101, 135
Illinois Scan · 56, 57
Intellectual Property · 35, 68
IP · 35, 68, 101
ISCAS'89 · 89, 93, 121
ITC'02 · 122, 137, 149, 150

L

LFSR · 36
Logique glue · 69, 101
Loi de Moore · 21
LUT · 40

M

Mentor Graphics® · 64
MISR · 37
Modèles de fautes · 17

N

Netlist · 35
Netlists · 89, 107
Network-on-Chip · 133
NoC · 133, 134, 139

O

Ordonnancement du test · 102, 124, 132, 139, 145

P

Payload · 136
Ports de test · 136, 137
Profondeur mémoire · 21
PRPG · 37

R

Reconfiguration dynamique · 54
Reconfiguration statique · 52
Réduction des coûts de test · 22
Réseau sur puce · 134
Réseau sur Puce · 133
RLE · 47
Routeur · 134
Run-Length Encoding · 47

S

Scan path · 23
Séquence de test · 18
Séquences aléatoires Modifiées · 39
Séquences déterministes modifiées · 38
Sérialisation des données · 51, 87
Sessions de test
    Longueur fixe · 104
Sessions de test
    Longueur variables · 104
Sink · 19
SoC · 69, 101
Source · 19
STUMPS · 37
Switchs reconfigurables · 52
Synopsis® · 89, 121
Synopsys® · 57
SynTest® · 59
System-on-Chip · 69, 101

T

TAM · 101, 106, 135
Taux de couverture · 19
Test Acces Mechanism · 101, 135
Test des Circuits intégrés · 15
Test non-préemptif · 140, 144, 145
Test préemptif · 139
Testeur
    ATE · 20
TestKompress · 64
TETRAMAX® · 89, 121
Time to market · 69
Top-up (vecteurs de) · 38

V

Vecteurs de test · 18
VirtualScan · 59

W

Wrapper · 101, 105, 134

COMPRESSION DE DONNÉES DE TEST POUR ARCHITECTURE DE SYSTÈMES INTÉGRÉS BASÉE SUR BUS OU RÉSEAUX ET RÉDUCTION DU TEMPS DE TEST

RÉSUMÉ : Les circuits intégrés devenant de plus en plus complexes, leur test demande des efforts considérables se répercutant sur le coût de développement et de production de ces composants. De nombreux travaux ont porté sur la réduction du coût de ce test en utilisant en particulier les techniques de compression de données de test. Toutefois ces techniques n'adressent que des cœurs numériques dont les concepteurs détiennent la connaissance de toutes les informations structurelles et donc en pratique n'adressent que le test de sous-blocs d'un système complet.

Dans cette thèse, nous proposons tout d'abord une nouvelle technique de compression des données de test pour les circuits intégrés compatible avec le paradigme de la conception de systèmes (SoC) à partir de fonctions pré-synthétisées (IPs ou cœurs).

Puis, deux méthodes de test des systèmes utilisant la compression sont proposées. La première est relative au test des systèmes SoC utilisant l'architecture de test IEEE 1500 (avec un mécanisme d'accès au test de type bus), la deuxième concerne le test des systèmes pour lesquels la communication interne s'appuie sur des structures de type réseau sur puce (NoC). Ces deux méthodes utilisent conjointement un ordonnancement du test des cœurs du système avec une technique de compression horizontale afin d'augmenter le parallélisme du test des cœurs constituant le système et ce, à coût matériel constant. Les résultats expérimentaux sur des systèmes sur puces de référence montrent des réductions du temps de test du système complet de l'ordre de 50%.

TEST DATA COMPRESSION FOR INTEGRATED SYSTEMS ARCHITECTURE BASED ON BUS OR NETWORKS AND TEST TIME REDUCTION

ABSTRACT: While microelectronics systems become more and more complex, test costs have increased in the same way. Last years have seen many works focused on test cost reduction by using test data compression. However these techniques only focus on individual digital circuits whose structural implementation (netlist) is fully known by the designer. Therefore, they are not suitable for the testing of cores of a complete system.

The goal of this PhD work was to provide a new solution for test data compression of integrated circuits taking into account the paradigm of systems-on-chip (SoC) built from pre-synthesized functions (IPs or cores).

Then two systems testing method using compression are proposed for two different system architectures. The first one concerns SoC with IEEE 1500 test architecture (with bus-based test access mechanism), the second one concerns NoC-based systems. Both techniques use test scheduling methods combined with test data compression for better exploration of the design space. The idea is to increase test parallelism with no hardware extra cost. Experimental results performed on system-on-chip benchmarks show that the use of test data compression leads to test time reduction of about 50% at system level.