ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE LA
MAÎTRISE EN GÉNIE ÉLECTRIQUE
M. Ing.

PAR
Haithem KHALED

ADAPTATION DE L'APPROCHE DE TEST CDIDDQ AUX CIRCUITS PROGRAMMABLES FPGA

MONTRÉAL, LE 24 NOVEMBRE 2011

PRÉSENTATION DU JURY

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

M. Claude Thibeault, directeur de mémoire
Département de génie électrique à l'École de technologie supérieure

M. Jean-François Boland, président du jury
Département de génie électrique à l'École de technologie supérieure

Mme Catherine Laporte, membre du jury
Département de génie électrique à l'École de technologie supérieure

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 24 NOVEMBRE 2011

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

REMERCIEMENTS

Je tiens à remercier sincèrement Monsieur Claude Thibeault, qui, en tant que directeur de mémoire, s'est toujours montré à l'écoute et très disponible tout au long de la réalisation de ce mémoire, ainsi que pour l'inspiration, l'aide et le temps qu'il a bien voulu me consacrer et sans qui ce mémoire n'aurait jamais vu le jour. Mes remerciements s'adressent aussi à Monsieur Yassine Hariri pour sa générosité et la grande patience dont il a su faire preuve malgré ses charges académiques et professionnelles.

Je tiens à remercier vivement tous les membres du jury de ma soutenance, pour l'attention qu'ils ont apportée à mon travail et pour leurs précieuses remarques grâce auxquelles j'ai pu améliorer ce mémoire. Merci aussi à M. Jean-François Boland de m'avoir fait l'honneur de présider ce jury. Merci aussi à Mme Catherine Laporte d'avoir accepté de faire partie du jury de mémoire et de m'avoir fait profiter de ses conseils éclairés.

Je ne saurai oublier l'appui inconditionnel de mes chers parents, ma chère khadija, mes deux frères (Maher, Kacem) et ma sœur (Yosra) qui ont été des piliers face aux durs moments que j'ai affronté. Je remercie aussi mes amis : Atef, Nizar et Amine. Vous m'avez donné plus de confiance!

Finalement, je remercie tous les membres passés et présents du Laboratoire de Communications et d'Intégration de la Microélectronique (LACIME) auprès de qui j'ai trouvé un bel environnement humain pour la recherche et bien plus qu'un lieu de travail. Je pense spécialement à Justine, Riadh et Normand. Merci d'avoir participé à l'élaboration de ce mémoire.

ADAPTATION DE L'APPROCHE DE TEST CDIDDQ AUX CIRCUITS PROGRAMMABLES FPGA

Haithem KHALED

RÉSUMÉ

Ce mémoire propose l'adaptation de la technique de test de courts-circuits CDIDDQ, développée au départ pour les circuits intégrés de type ASIC, pour le test des circuits intégrés configurables de type FPGA à base de SRAM. Après une présentation de l'architecture des circuits ciblés et une revue de littérature dans le domaine du test de ce genre de circuits, une stratégie générale d'adaptation de la technique de test aux interconnections de ces circuits est proposée. Le processus d'adaptation débute par la validation expérimentale de la technique de test en l'appliquant à un circuit ASIC émulé dans un FPGA. Ceci permet de généraliser l'application de la technique aux tests dépendants de l'application et de bénéficier de taux de couverture qu'elle offre qui peut atteindre plus de 90%.

En second lieu, le processus est appliqué d'une manière indépendante de l'application. Dans ce cadre, l'adaptation de la technique a été effectuée pour le test des interconnections des FPGAs. Pour ce faire, une nouvelle structure de configuration est proposée. Elle consiste à mettre sous forme de peignes entrelacés l'ensemble des interconnections ciblées, à savoir les lignes horizontales doubles nord du FPGA Spartan 3e de la compagnie Xilinx. Cette forme de configuration est novatrice et permet de minimiser le nombre de configurations nécessaires durant le processus de test. La mise en place de la technique est assurée par un outil de configuration automatique de la structure développé dans le cadre de ce mémoire. Une plateforme de test à faible coût, développée à partir de deux plateformes commerciales, est utilisée pour valider expérimentalement les concepts. Les résultats obtenus avec cette strategie montrent une nette diminution du temps de test pour une couverture des pannes de 100% comparés aux solutions basées sur des techniques plus complexes.

Mots-clés : FPGA, court-circuit, CDIDDQ, strategie de configuration, structure de peigne, test dependant, test independant.

ADAPTATION DE L'APPROCHE DE TEST CDIDDQ AUX CIRCUITS PROGRAMMABLES FPGA

Haithem KHALED

ABSTRACT

This thesis proposes the adaptation of the CDIDDQ test technique, primarily developed for ASICs, to test configurable SRAM-Based FPGA circuits. After presenting the architecture of these target circuits and reviewing some of the test configuration literature, a general strategy for adapting the technique to test the interconnections of these circuits is proposed. The adaptation process starts with the experimental validation of the test technique by applying it to an ASIC circuit emulated in an FPGA. This results allows the use of this kind of technique for the application dependant testing and benefits of its coverage which can reach more than 90%.

Secondly, the process is applied independently of the application. In this context, the adaptation of the technique is specifically done for testing some FPGA interconnects. For that purpose, a new configuration strategy is proposed to put all the interconnections in the shape of an interleaved comb structure. The proposed configuration is innovative and contributes to reduce test time. The strategy uses a new script-based auto-configuration tool, developed for the target FPGA interconnect, namely the horizontal north double lines in the Xilinx Spartan 3e FPGA. A very low cost test platform merging two commercial boards is designed to experimentally validate the concept. The results obtained show that 100% fault coverage is achieved with a significant reduction in test time compared to the existing solutions.

Keywords: FPGA, bridging defects, CDIDDQ, configuration strategy, comb structure, dependent testing, independent testing.

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

ASIC	Application-specific integrated circuit
ASCII	American standard code for information interchange
ATPG	Automatic test pattern generation
ATE	Automatic test equipment
BlC	Bloc logique cofigurable
BF	Bridging fault
BIST	Built-in self test
CC	Court-circuit
CLB	Configurable logic bloc
CMOS	Complementary Metal-Oxide-Semiconductor
CPLD	Complex programmable logic device
CPD	Central process unit
CUT	Circuit under test
CDIDDQ	Complementary delta direct drain quiescent current
DDB	Design data base
DFT	Design for testability
DRAM	Dynamic random acess memory
DSP	Digital signal processor
EDIF	Electronic design interchange format
ÉTS	École de technologie supérieure
FIR	Finite impulse response
FPGA	Field-programmable gâte array
FSM	Finite state machine
GUI	Graphic user interface
HDL	Hardware description language
IO	Input/output
ISE	Integrated software environnement
ISim	Ise simulator
JTAG	Joint test action group
LE	Logic element
LACIME	Laboratoire de communication et d'intégration de la microélectronique
LFSR	Linear feedback shift register
LOC	Launch on capture
LOS	Launch from (last) shift
LUT	Look-up table
NCD	Native circuit description
NDR	Network data representation
PC	Personal computer
PLB	Programmable logic bloc
PI	Primary input
PIP	Programmable interconnect point
PO	Primary output
PUT	Path under test

RTL	Register to transistor level
SCI	Scan chain input
SCO	Scan chain output
SI	Scan input
SM	Switch matrix
SoC	System on chip
SRAM	Static random access memory
TDF	Transition delay fault
USB	Universal serial bus
VHDL	Very high speed integrated circuit hardware description language
XDL	Xilinx design language
XPS	Xilinx platform studio

INTRODUCTION

Introduits sur le marché en 1985, les circuits programmables FPGA sont devenus des composants incontournables, entre autres sur les cartes de traitement de signal et d'images. Aujourd'hui, ces puces sont devenues les concurrents principaux des ASIC et des DSP en combinant certains avantages des deux et en bénéficiant, comme ses rivales, de l'évolution de la densité d'intégration qui double tous les 18 mois selon la fameuse loi de Moore. Évoluer à ce rythme entraîne toutefois son lot de défis. Ces défis touchent notamment l'étape de test suivant la fabrication de ces circuits de haute intégration.

Cette étape de test est indispensable. Le but principal du test est de s'assurer du bon fonctionnement des circuits fabriqués tout en respectant un budget et un temps prédéfinis. En d'autres termes, cette étape consiste à détecter les circuits qui ne répondent pas aux spécifications, en raison par exemple de défectuosités affectant des transistors ou des interconnections, causant des pannes pouvant se manifester durant le fonctionnement normal du circuit. La détection de ces défectuosités et pannes requiert dans un premier temps la génération des séquences de vecteurs appropriés pouvant les activer et propager leur effet vers des sorties observables, et dans un deuxième temps, l'application de ces vecteurs et l'analyse de la réponse du circuit.

Dans le cas des circuits FPGA, l'application de vecteurs de test signifie au départ la configuration de la puce dans un état précis, permettant d'observer l'effet d'un ensemble de défectuosités et de pannes qui affectent les ressources configurées. Avec l'évolution de la complexité des FPGAs, qui aujourd'hui comportent des centaines de millions (voire des milliards) de transistors de même qu'un très grand nombre d'interconnections, il est devenu nécessaire de reconfigurer le circuit plusieurs fois pour détecter le plus de défectuosités et de pannes possible. Ceci augmente en conséquence le temps requis pour effectuer le test, ce qui se traduit par l'augmentation de coût de cette étape. La minimisation du nombre de configurations constitue à la fois une voie privilégiée dans l'optimisation de l'efficacité du

test en termes de budget et de temps, mais également un défi en ce qui concerne la capacité du test à détecter le plus de défectuosités et de pannes possible.

Une des pistes de solutions potentielles réside dans l'utilisation d'une technique de test basée sur le courant de consommation statique appelée CDIDDQ (Thibeault et Hariri 2009), dont l'efficacité a été démontrée pour le test de circuits intégrés ASIC de technologie CMOS. Cette technique permet d'éliminer les principales sources de variation de courant statique, à savoir les variations d'un lot de puces à l'autre, d'une puce à l'autre et d'un vecteur de test à l'autre. La seule source de variation qui demeure est celle de la mesure elle-même. Cette élimination est l'élément clé permettant à CDIDDQ d'être applicable sur les technologies récentes, alors que les autres techniques de test basées sur le courant de consommation statique, qui étaient jusqu'à très récemment considérées efficaces dans la détection des courts-circuits affectant les puces, ont perdu de leur pouvoir de détection de défectuosités. Ceci est causé par l'augmentation significative des courants de fuite et par celle des variations du courant d'une tranche à une autre et d'une puce à une autre. L'exploration de cette piste se fait en deux étapes, afin de couvrir les deux principales approches de test des FPGA, à savoir 1) les tests dépendants de l'application et, 2) les tests indépendants de l'application. Ces approches seront décrites en détails au chapitre suivant.

Nous proposons donc en premier lieu d'appliquer la technique CDIDDQ aux ASICs émulés dans les FPGAs. Ceci offre la possibilité d'appliquer une telle technique à l'approche des tests dépendants de l'application. Dans un deuxième temps, nous proposons d'adapter la technique CDIDDQ pour les tests indépendants de l'application. Dans ce premier effort d'adaptation, nous nous intéressons principalement au test des interconnections dans les circuits FPGA, éléments qui constituent 80% des ressources des FPGA. De manière plus spécifique, nous concentrons nos efforts sur la détection de courts-circuits qui affectent certaines de ces ressources. Dans ce cadre, nous introduisons une nouvelle structure qui permet de configurer l'ensemble des interconnections ciblées sous forme de peignes entrelacés, sur lesquelles sont appliqués les vecteurs de test CDIDDQ. Cette forme de configuration est novatrice et permet de rentabiliser le processus de test. Nous proposons

notamment un outil de configuration automatique de la structure pour les lignes horizontales doubles nord du Spartan 3e de la famille Xilinx. Notons qu'une plateforme de test à faible coût a été développée pour valider l'application de CDIDDQ lors des deux étapes.

Le reste de ce mémoire est organisé comme suit :

1- Le premier chapitre présente l'architecture des circuits configurables de type FPGA et l'état de l'art pour le domaine du test manufacturier des circuits intégrés.
2- Le deuxième chapitre propose l'application de l'approche de test CDIDDQ développée pour les ASIC émulés dans un FPGA, équivalent à un test dépendant de l'application. Il décrit aussi la plateforme de validation expérimentale et présente les résultats expérimentaux obtenus avec cette plateforme.
3- Le troisième chapitre propose la nouvelle stratégie de configuration ainsi que l'outil automatisé développés pour appliquer la technique CDIDDQ à un ensemble d'interconnections, équivalent à un test indépendant de l'application.
4- Le quatrième chapitre présente les résultats expérimentaux obtenus suite à l'application de la stratégie de configuration du chapitre 3 sur la même plateforme de test.

CHAPITRE 1

NOTIONS DE BASE ET REVUE DE LITTÉRATURE

1.1 Introduction

Dans ce chapitre, nous allons présenter l'architecture des circuits configurables de type FPGA et un état de l'art dans le domaine du test manufacturier des circuits intégrés. Ainsi nous allons faire ressortir les meilleures stratégies de test décrites dans la littérature pour les FPGA contemporains. Ceci nous permettra en conclusion de justifier la contribution apportée par notre proposition.

1.2 Circuits programmables FPGA

Les circuits logiques programmables FPGA sont des circuits intégrés prédiffusés programmables par l'usager. La plupart des FPGA les plus récents contiennent deux types de ressources : des ressources spécialisées dédiées (telles des multiplicateurs et des processeurs embarqués) ainsi que des ressources dites génériques. Dans le cadre de ce projet, nous nous intéressons au second type de ressource. Ces ressources génériques se présentent habituellement sous la forme d'une architecture matricielle interne fixe qui consiste, dans la plupart des cas, en des ports d'entrées/sorties (IOB) et des blocs logiques configurables (BLC) séparés par des interconnections programmables qui traversent le circuit, horizontalement et verticalement. Ces interconnections permettent de relier les BLC entre eux, ainsi qu'avec les ports d'entrées/sorties. La Figure 1.1 présente la structure générale des FPGA. Trois technologies différentes sont utilisées pour programmer les différentes ressources d'un FPGA : les antifusibles, la mémoire flash et la mémoire SRAM. Dans ce projet, nous ciblons les FPGA utilisant la troisième technologie, en particulier le Spartan 3E de la compagnie Xilinx. Ainsi, les interrupteurs programmables dirigeant la propagation des signaux sur les interconnections sont réalisés par des transistors MOS dont l'état est contrôlé par des cellules de mémoire SRAM. De la même manière, la définition des fonctions

contenues dans les BLC est mémorisée dans ce genre de cellules. Par conséquent, toute la configuration d'un FPGA basé sur cette technologie est contenue dans des cellules SRAM.

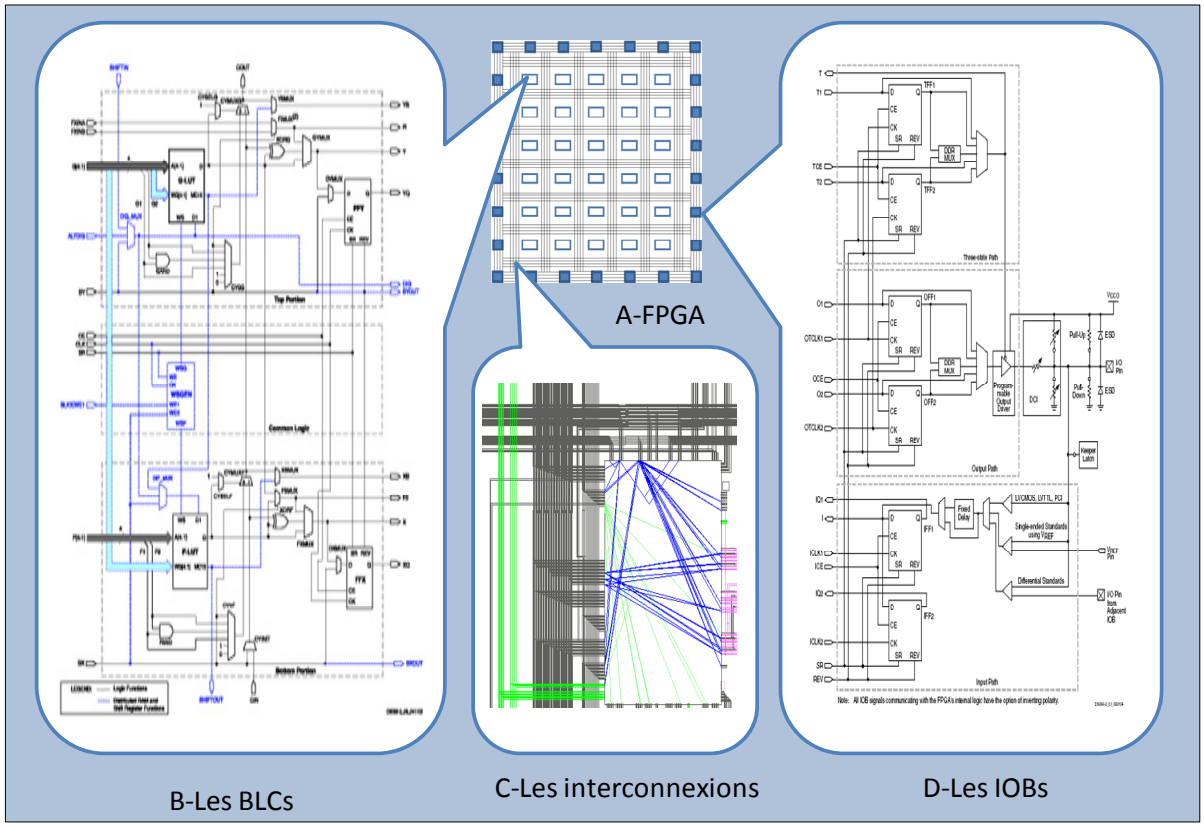

Figure 1.1 Structure d'un FPGA
B et D Tirées de Xilinx (2003, p. 11, 22)
C Tirée du logiciel FPGA Editor (2000)

1.2.1 Blocs logiques configurables

Le concept du FPGA est basé sur l'utilisation de BLC. Dans la terminologie de la compagnie Xilinx, ces blocs sont appelés CLB (*Configurable Logic Block*). Ces blocs constituent la principale ressource pour la mise en œuvre de la logique combinatoire ou séquentielle. Dans le Spartan 3E, chaque BLC contient quatre tranches (*slices*), et chaque tranche contient principalement deux générateurs de fonctions ou tables de conversion (*Look-Up Table, LUT*) et deux bascules.

- LUT : Chaque LUT est une mémoire de 16 bits qui permet entre autres de créer n'importe quelle fonction logique combinatoire à 4 variables d'entrée. Ces mémoires sont rapides et elles ont l'avantage d'être situées au cœur de la fonction à réaliser;
- bascules : La sortie des générateurs de fonctions peut être connectée directement à une sortie de la tranche ou bien être mise en mémoire par deux bascules D. Ces deux bascules ont la même horloge (CLK), le même signal de validation (CE) et la même logique de mise à 0 ou de mise à 1 asynchrone (SR). Les deux bascules peuvent être utilisées indépendamment ou à la suite des générateurs de fonctions.

1.2.2 Ports d'entrées/sorties

Les ports d'entrées/sorties, appelés IOB (*Input-Output Blocks*), sont configurables et répartis sur toute la périphérie du boîtier. Chaque IOB assure l'interface entre une broche d'entrée/sortie du boîtier et la logique interne. À l'intérieur d'un IOB, on distingue trois signaux principaux:

- le signal d'entrée : Il traverse un amplificateur qui, selon sa programmation, peut détecter divers standards (TTL, CMOS, etc.). Ce signal est amené vers les CLB (via les interconnections) soit directement, soit à travers une paire de bascules D. Il peut également être retardé de quelques nanosecondes en passant par un élément de retard programmable;
- le signal de sortie : Ce signal peut être optionnellement inversé à l'intérieur de l'IOB et sortir directement sur la broche, ou bien être mis en mémoire par une paire de bascules D;
- le signal en logique trois états : Il contrôle la mise en haute impédance et la réalisation des lignes bidirectionnelles. Ce signal transporte les données à partir de la logique interne du FPGA par le biais d'un multiplexeur qui offre la possibilité d'insérer une paire de bascules D. Ainsi, ces possibilités offrent une communication facile et contrôlable avec les périphériques extérieurs.

1.2.3 Interconnections programmables

Le rôle de ces interconnections est de relier avec un maximum d'efficacité et de souplesse les BLCs et les IOBs afin que leur taux d'utilisation pour une application donnée soit le plus élevé possible. La structure d'interconnections est constituée des pistes horizontales et verticales et des matrices de commutation (*Switching Matrix*, *SM*). Chaque SM est composée d'interrupteurs programmables permettant d'établir une connexion entre les lignes horizontales et verticales qu'elle reçoit dans chacune de ces quatre faces.

Chaque passage par une SM introduit un délai de propagation qui peut être très pénalisant pour les liaisons à longue distance. Aussi, les interconnections ont été spécialisées en fonction de la distance à parcourir. Comme le présente la Figure 1.2, quatre types d'interconnections sont disponibles dans Spartan 3E :

- les lignes directes : Ces lignes permettent l'établissement de liaisons entre les BLC et les IOB avec un maximum d'efficacité en termes de vitesse et d'occupation du circuit (figure 1.1). De plus, ces lignes relient la SM d'un BLC avec celle de SM de BLC voisin immédiat;
- les lignes doubles (Double Line) : Jouent le même rôle que les lignes directes, sauf qu'elles sont moins connectables et moins flexibles. Ces lignes relient la SM d'un BLC avec la SM du deuxième BLC voisin;
- les lignes sextuples (Hex line) : Ces lignes sont utilisées pour relier la SM d'un BLC à la SM de cinquième BLC voisin;
- les lignes longues (Long Line) : Ce sont de longs segments parcourant toute la longueur ou la largeur du FPGA. Dans le but d'assurer un synchronisme aussi parfait que possible, ces longues lignes permettent d'éviter la multiplicité des points d'interconnections et éventuellement de transmettre avec un minimum de retard les signaux entre les différents éléments. Les longues lignes sont connectées au rythme d'une SM par BLC.

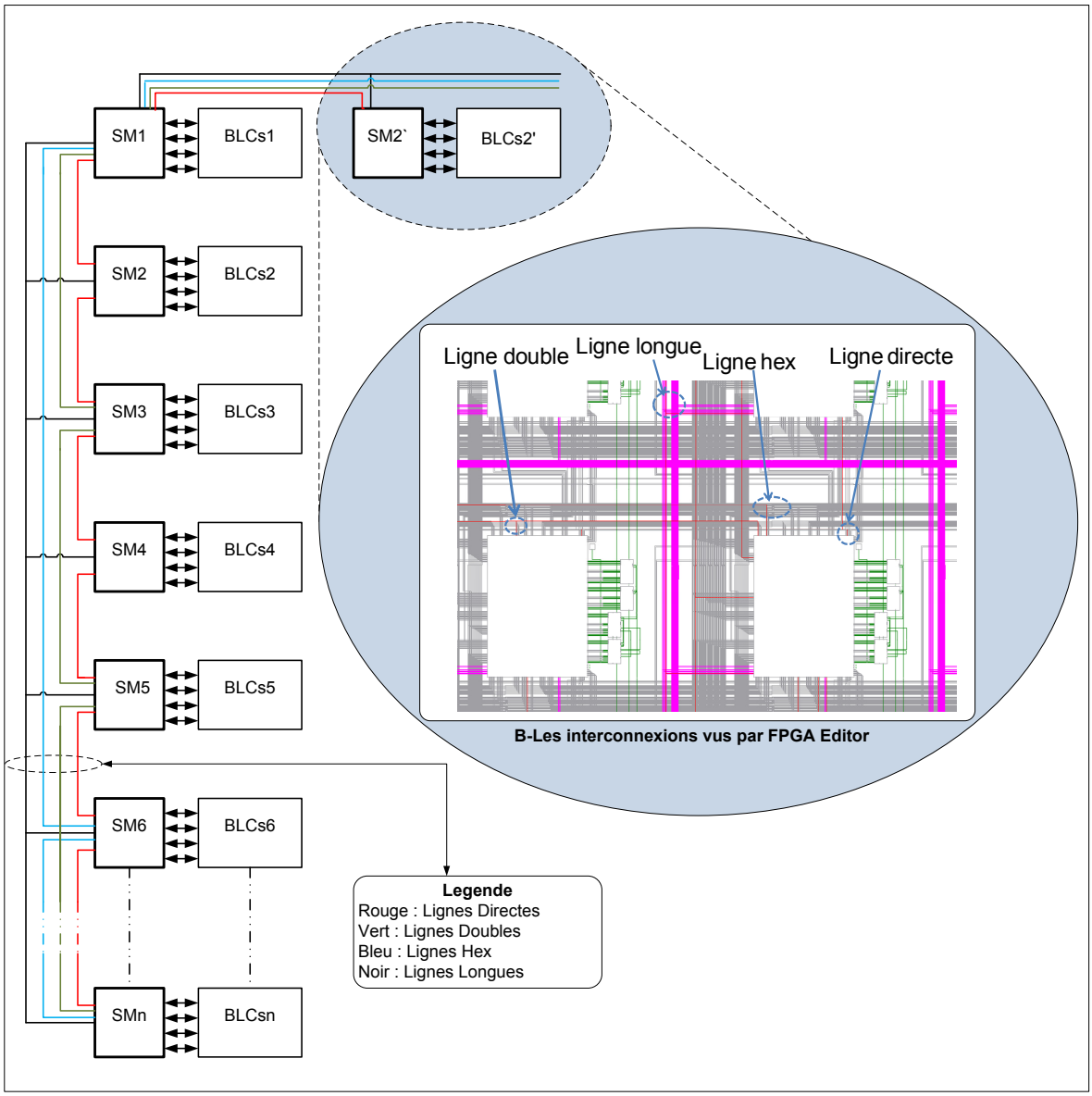

Figure 1.2 La structure des interconnections

1.3 Notions sur le test des circuits intégrés numériques

La condition essentielle pour obtenir un circuit intégré numérique (CI) fiable consiste à déterminer que ce CI est non défectueux (Breuer and Friedman 1976). Il s'agit donc d'avoir la capacité de détecter d'éventuelles pannes de conception ou de fabrication afin d'obtenir les

performances qui garantissent la fiabilité. Pour ce faire, il est indispensable d'avoir un système testable. Selon (Bennetts 1984), un CI est testable si des vecteurs et des séquences de test peuvent être générés, appliqués et évalués de manière à satisfaire les niveaux de performance, le budget et l'échéance prédéfinis. Toutefois, le budget et le temps peuvent souvent dépasser ceux des autres tâches requises durant l'accomplissement de la conception. L'objectif principal de la conception des tests des CI est la réduction de ces paramètres tout en gardant une qualité élevé de test.

1.3.1 Vecteurs et séquences de test

Un vecteur de test est un couple constitué par l'ensemble de valeurs imposées aux signaux d'entrée et l'ensemble de valeurs attendues par les signaux de sortie. En résumé, le développement de vecteurs de test d'un CI est caractérisé par trois facteurs (Bushnell and Agrawal 2000):

- la contrôlabilité : elle mesure le degré de facilité avec laquelle on peut affecter une valeur spécifique à n'importe quel nœud dans le CI;
- l'observabilité : elle mesure le degré de facilité avec laquelle on peut déduire la valeur de n'importe quel nœud du CI uniquement en observant ses sorties;
- la prédictibilité : elle mesure le degré de facilité avec laquelle on peut obtenir la valeur d'une sortie en réponse à une application éventuelle d'une valeur donnée à l'entrée du CI.

Dans le cas d'un circuit combinatoire, seuls les deux premiers facteurs s'appliquent. Le développement de vecteurs de test efficaces est donc plus facile que pour les circuits séquentiels. Il comporte cependant certains pièges. Si, en général, chaque vecteur va permettre de détecter un certain nombre de pannes, certaines pannes peuvent être plus difficiles à mettre en évidence. Notons par exemple qu'il existe des pannes induisant un comportement séquentiel et nécessitant un ordre d'application de vecteurs particulier pour être détectées (Bushnell and Agrawal 2000). Par conséquent, même l'application d'une séquence de test composée de vecteurs recréant les toutes combinaisons possibles (2^N

vecteurs pour un circuit combinatoire possédant N entrées) ne garantit pas la détection de toutes les pannes potentielles.

De par l'ajout de la prédictibilité, le développement de vecteurs de test pour un circuit séquentiel est plus ardu. La détection d'une panne suppose en général de placer tout d'abord le circuit dans un état précis permettant d'activer la panne. Ceci nécessite l'application d'un certain nombre de vecteurs d'entrée, puis de propager la panne éventuelle vers une sortie observable, ce qui peut à nouveau nécessiter l'application d'un certain nombre de vecteurs d'entrée. Le respect de l'ordre d'application des vecteurs d'entrée est donc fondamental, et la notion de vecteur de test est remplacée par la notion de séquence de test ordonnée, pour la détection d'une panne donnée (Bushnell and Agrawal 2000).

Devant l'évolution incessante de la complexité des CI, il a fallu porter une attention particulière au test des circuits, afin de s'assurer qu'il puisse être de qualité suffisante et être effectué de manière efficace. Ceci a mené d'une part au développement des stratégies de conception facilitant la testabilité des circuits (créant essentiellement un mode test rendant combinatoires les circuits séquentiels) et d'autre part à celui d'algorithmes de génération automatique des vecteurs de tests (ATPG).

1.3.2 Principaux modèles de pannes

Pour faciliter la génération de vecteurs de test, un certain nombre de modèles de pannes ont été développés. Ces modèles constituent une représentation abstraite des divers éléments pouvant perturber le fonctionnement d'un CI. Ces perturbations peuvent être induites par les conditions environnementales telles la variation de l'alimentation ou les effets de radiations. Elles peuvent également être causées par des défectuosités introduites par le procédé de fabrication. On distingue trois principaux types de modèle de pannes : collé-à, retard et court-circuit.

1.3.2.1 Modèle de panne collé-à (*stuck-at*)

Le modèle de panne le plus utilisé est le modèle collé-à. Il s'applique classiquement à toutes les parties de circuits réalisées en logique combinatoire avec une densité d'implantation moyenne. Il consiste à supposer l'existence d'un signal quelconque figé sur l'une des deux valeurs logiques (collé-à 0 « s-a-0 » ou collé-à 1 « s-a-1 »). Toute entrée ou sortie de porte peut être ainsi collée à l'une des deux valeurs. Pour un CI comportant N nœuds électriques, il peut donc y avoir N collages à 0 et N collages à 1, soit 2 × N pannes possibles dans le CI. Ce modèle est habituellement appliqué en supposant qu'une seule de ces pannes peut exister dans un circuit donné (« collé-à simple »)(Mourad and Zorian 2000).

Comme illustré à la Figure 1.3, un nœud collé-à 1 (respectivement, 0) peut représenter un court-circuit avec l'alimentation VDD (respectivement, avec la masse). Le CI garde alors un fonctionnement combinatoire, mais avec une fonction modifiée. Le test collé-à consiste alors à vérifier le comportement logique du CI en présence de cette panne.

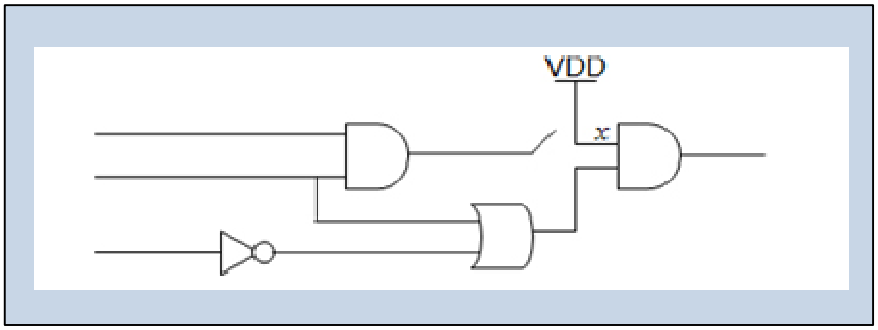

Figure 1.3 Circuit affecté de collé-à 1

1.3.2.2 Modèle de panne de retard

Face à l'intégration nanométrique de la technologie CMOS (moins de 130 nm), l'opportunité d'avoir des circuits résistifs ouverts ou fermés augmente énormément. La présence d'un

circuit résistif ouvert dans un CI ne change pas le comportement logique ; il va agir uniquement sur le temps nécessaire pour obtenir une certaine transition sur un signal interne.

Ces pannes sont indétectables par les tests collé-à, ce qui a conduit à proposer des modèles de pannes de retard (*delay faults*) qui se décomposent en pannes de transition et en pannes de retard de chemins.

- les pannes de transition : elles correspondent à un retard de transition en entrée ou en sortie d'une porte. Ce modèle suppose que le délai supplémentaire apparaît de manière ponctuel à un point donné du circuit et que ce délai est suffisamment important pour causer une erreur de synchronisation (i.e. la transition arrive en retard). Il y a deux types de pannes de transitions : « lente à monter » (slow-to-rise) et « lente à descendre » (slow-to-fall). À l'extrême, une porte « lente à monter » (respectivement à, « lente à descendre ») peut être assimilée à une porte avec un collé-à 0 (respectivement, à 1) à la sortie;
- les pannes de retard de chemins : Dans ce modèle de pannes, tout chemin avec un retard total supérieur à la période d'horloge du système est défectueux. Ce modèle distribue les pannes qui affectent un chemin d'accès complet. Pour chaque chemin d'accès physique P, la connexion d'une entrée à une sortie du circuit a deux chemins retard correspondant. Le chemin de la montée (rising path) (respectivement à, descente (falling path)) est le chemin parcouru par une transition qui est ouverte à la montée (respectivement à, descente) de transition à l'entrée du chemin de P. L'inconvénient de ce modèle est le nombre énorme de chemins devant être pris en compte dans un circuit complexe, rendant la génération et l'application des vecteurs de test très longues.

Les deux types de pannes de retard nécessitent au moins deux vecteurs de test :

- le premier permet de mettre le circuit dans un état tel que la transition souhaitée peut se propager sur le chemin (ou à travers la porte) testé(e) ;
- le second vecteur permet ensuite d'activer la faute, si elle est présente, en créant la transition.

1.3.2.3 Modèle de panne court-circuit

Les courts-circuits (*short* ou *bridge*) sont un autre cas de pannes difficilement détectables par un test collé-à. Comme l'ont noté Gkatziani, Kapur et al. (2007), la probabilité d'avoir un court-circuit entre deux lignes augmente pour les chemins parallèles de faible distance entre eux. Cette panne entre les lignes (voir Figure 1.4) (donc entre deux sorties de portes) peut se traduire soit par un niveau indéterminé, soit par un niveau imposé par l'une des deux portes dans le cas où le dimensionnement des transistors des deux portes est très différent. Par ailleurs, un court-circuit peut transformer un circuit combinatoire en circuit séquentiel, en introduisant un bouclage synchrone ou asynchrone. La détection de ce type de panne est très délicate et nécessite des vecteurs de test spécifiques.

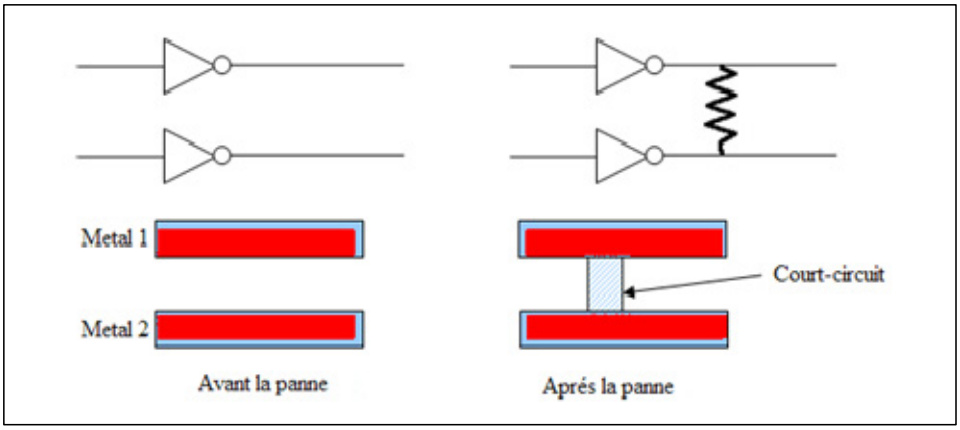

Figure 1.4 Panne de court-circuit entre deux inverseurs

1.3.3 Taux de couverture

Le taux de couverture est la mesure qui permet de quantifier l'efficacité d'une séquence de test particulière pour un circuit sous test. Il est défini comme le rapport entre le nombre de pannes testées et le nombre total de pannes considérées. Le taux de couverture est en général évalué en réalisant des simulations de pannes, qui consistent à simuler la description logique du circuit (netlist) en présence de chacune des pannes du modèle sélectionné (par exemple, le collé-à des interconnections entre les portes). La simulation du circuit sans insertion des pannes sert de référence pour évaluer la détection ou non de chaque panne (dans ce

processus, il est implicitement supposé que les pannes de conception ont été préalablement éliminées et que les pannes sont uniquement dues au processus manufacturier). L'obtention du taux de couverture par simulation de pannes est très coûteuse en temps de calcul, même lorsqu'un accélérateur matériel est employé. Il est donc important de générer un ensemble efficace de vecteurs de test dès le départ, afin d'éviter les itérations sur la simulation des pannes.

1.4 Technique de test CDI_{DDQ}

La modélisation logique des courts-circuits ne permet pas de détecter 100% des pannes causées par la connexion de deux nœuds distincts ou plus dans un CI (Chakravarty and Thadikaran 1997). Or des études ont montré que ce type de pannes peut représenter jusqu'à 70% des pannes qui affectent les ASIC (Chakravarty et Thadikaran 1997). Pour augmenter l'efficacité de la détection de ce type de panne, il a été nécessaire de faire appel aux caractéristiques physiques de la logique CMOS.

1.4.1 Introduction au courant statique I_{DDQ}

Parmi les caractéristiques physiques des circuits CMOS, on trouve en particulier le courant I_{DDQ}. Il s'agit du courant provenant de l'alimentation du circuit lorsqu'il est en état statique. En partant du principe que les circuits CMOS ne consomment pas beaucoup de puissance dans cet état, le courant I_{DDQ} sera alors un courant de fuite très faible provenant des transistors désactivés. De plus, certaines pannes, tels les courts-circuits, peuvent introduire des courants supplémentaires. La technique de test basée sur le courant I_{DDQ} vise à la détection de ces courants supplémentaires, et consiste à mesurer ce courant et à la comparer à un seuil prédéterminé : ainsi, un circuit pour lequel une ou des valeurs de courant I_{DDQ} dépassent ce seuil est considéré défectueux, tel que décrit dans la Figure 1.5.

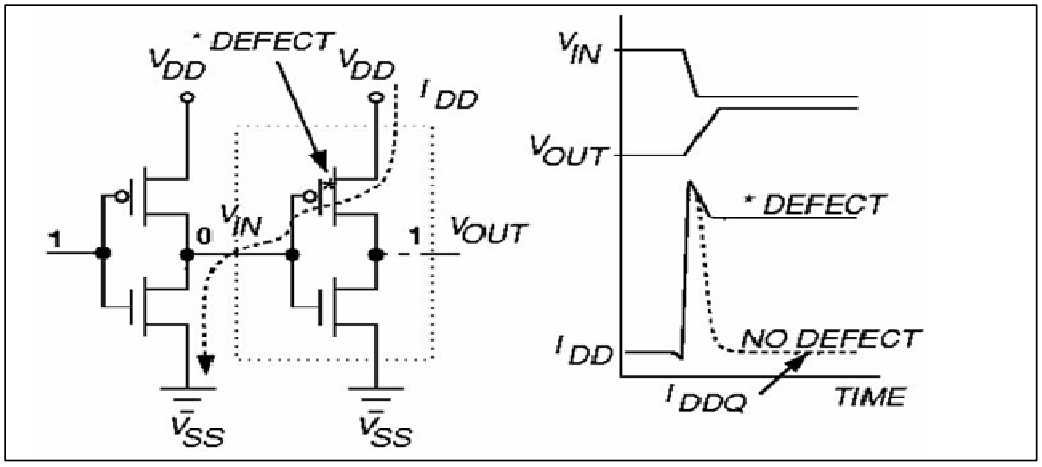

Figure 1.5 Impact d'une panne de court-circuit au courant I_{DDQ}
Tirée de Bushnell, M. L. and V. D. Agrawal (2002, p. 440)

1.4.2 Courant différentiel DI_{DDQ}

L'efficacité de la technique de test I_{DDQ} diminue face au bruit sur les mesures de courant d'une puce à une autre et d'une tranche à une autre. Ce bruit est causé par le courant de fuite engendré par les transistors fermés, et augmente significativement avec la réduction de la taille des transistors. En s'appuyant sur des analyses expérimentales, Thibeault (1998) montre que le courant de fuite suit une distribution normale d'un vecteur de test à l'autre avec une moyenne de zéro et une certaine variance pour les circuits de grande dimension. Dans ce contexte, il a proposé l'utilisation de D (Δ, ou différentiel) I_{DDQ} qui permet de diminuer la variance des distributions du courant, en éliminant celle causée par la différence observée d'une puce à l'autre. En effet, la technique se base sur la mesure de la différence entre deux mesures consécutives (i et i-1) du courant I_{DDQ} comme l'indique l'équation suivante :

$$DI_{DDQ}(i) = I_{DDQ}(i) - I_{DDQ}(i-1), \quad (1.1)$$

En outre, l'auteur a étendu sa technique pour faire le diagnostic du court-circuit. Ainsi, l'exploitation des mesures de DI_{DDQ} permet d'identifier les pannes les plus probables dans un premier temps et de les localiser dans un deuxième temps.

1.4.3 Technique de test CDI_{DDQ}

(Thibeault et Hariri 2009) ont développé une nouvelle technique appelée CDI_{DDQ} (*Complimentary delta* I_{DDQ}) pour étendre davantage la vie utile des tests $\mathbf{C}I_{DDQ}$ dont l'efficacité diminue également avec la réduction de la taille des transistors (CMOS). L'approche utilisée est basée sur des algorithmes combinant l'après-traitement des mesures I_{DDQ} avec une génération modifiée des patrons de test basés sur le modèle panne de transition (TDF), permettant d'identifier de faibles défectuosités lorsque le résultat de combinaisons de mesures est différent de zéro. En effet, contrairement à d'autres approches, CDI_{DDQ} élimine les sources de variation de courant principales à savoir, tranche à tranche, d'IC à IC et de vecteur à vecteur, la seule source restante étant la variance de mesure. Plus de détails sont fournis au chapitre 2.

Pour valider l'approche, les auteurs ont montré que les combinaisons de $C\Delta I_{DDQ}$ appliquées à sept circuits choisis (parmi ISCAS89, ITC99 et FIR1000) sont nulles (l'aspect théorique de la méthode sera introduit au chapitre 2). Notons toutefois que cette technique nécessite une augmentation du nombre des patrons de test (26% en moyenne). Les résultats obtenus ont montré une amélioration significative, comparés avec les résultats donnés par les autres techniques qui visent la détection du même type de panne.

Vu que cette approche est basée sur les informations de la couche physique, la stratégie de développement a été concentrée sur les défectuosités inter-portes qui constituent 90% des défectuosités totales. De ce fait, la limitation principale de cette technique réside dans le fait qu'elle ne détecte pas les défectuosités passives.

1.5 Détection des pannes dans les circuits FPGA

Depuis l'introduction des circuits FPGA sur le marché, des chercheurs se sont intéressés à la détection des pannes qui peuvent affecter ces circuits. De ce fait, on trouve dans la littérature plusieurs articles qui abordent ce sujet. Les approches proposées peuvent être classées en trois catégories. La première catégorie utilise certains aspects du test fonctionnel des ASIC adaptés aux FPGA, catégorie souvent appelée test dépendant de l'application. La deuxième approche exploite la programmabilité de ces circuits pour tester les ressources. Cette approche est connue comme le test indépendant de l'application. La troisième méthode consiste à l'application des tests I_{DDQ} sur la puce qui peut être appliqué également dépendamment et indépendamment des applications.

1.5.1 Test dépendant de l'application

Les puces de type FPGA qui échouent le test manufacturier restent parfois utilisables pour des applications spécifiques. En testant les ressources utilisées par ces applications, certaines puces, marquées déjà comme puces rejetées, peuvent être vendues aux clients. C'est entre autres pour cette raison que le test dépendant de l'application a été défini et par la suite adopté par (Xilinx 2003), question de profiter de l'augmentation de la rentabilité offerte par cette stratégie.

Le concept de base du test dépendant de l'application a été introduit et mis en œuvre par (Krasniewski 1999) afin de détecter les pannes de retard dans un circuit. Dans ce contexte, l'auteur a proposé une technique de test combinatoire exhaustif de l'ensemble de circuits utilisés pour implémenter l'application cible dans le FPGA. L'approche consiste en la décomposition des blocs logiques utilisés en des sous-circuits. Chaque sous-circuit est constitué d'un bloc logique combinatoire ou d'une machine à état finis (FSM) avec une horloge d'entrée et de sortie. Ces sous-circuits sont ensuite regroupés de façon appropriée en fonction des critères temporels. Par la suite, les groupes sont soumis simultanément à des tests autonomes au cours d'une session particulière afin de prévoir la vitesse des circuits et

détecter les problèmes de synchronisation. Bien que les groupes des sous-circuits soient configurés de la même manière ou d'une manière très similaire, il est fort probable que les exigences de la préservation de synchronisation ne soient pas satisfaites. Ceci est dû à l'impact de la structure logique et du placement physique de tous les composants des groupes sur la caractéristique de synchronisation. Ainsi, cette technique n'est pas capable de distinguer les pannes de retard d'un problème particulier de synchronisation causé par les variations des strucures physiques des composants.

(Das et Touba 1999) ont présenté une approche pour détecter les pannes collé-à (*stuck-at*) et les courts-circuits dans les interconnections. Leur procédure systématique appliquée à la configuration originale permet de générer un ensemble de configurations de test. Ceci est fait en modifiant la fonction logique des BLCs pour tester rapidement les interconnections. La fonction logique des BLCs est choisie de telle manière que l'approche " Marche-1" puisse être utilisée pour détecter et localiser toutes les pannes collé-à et les courts-circuits avec un petit ensemble de vecteurs de test. Le principe de l'approche "Marche-1" consiste à contrôler chaque chemin par des 0 et des 1 et les observer après un certain nombre de coups d'horloge. Les auteurs montrent que le processus de génération des configurations de test et des vecteurs de test est entièrement automatisé et très rapide. Le principal inconvénient de cette procédure demeure la limitation de l'observabilité des nœuds qui provoque toujours une complexité importante sur le chemin formé par les bascules à balayage (*scan*) et le nombre important de configurations requis en conséquence.

(Renovell, Faure et al. 2001) ont proposé une architecture d'FPGA plus adaptée pour les tests dépendants des applications. Tout d'abord, les auteurs ont montré l'inefficacité de l'architecture originale de ces circuits pour l'implémentation de circuits séquentiels avec des chaines de bascules à balayage. En conséquence, ils proposent de modifier cette architecture pour créer une chaine de bascules à balayage implicite dans l'FPGA. Ceci est fait par l'ajout d'un multiplexeur dans le BLC pour balayer les registres. La nouvelle architecture, nommée IS-FPGA (*implicit scan* FPGA), permet de tester implicitement n'importe quel circuit séquentiel implémenté dans l'FPGA. Cette technique nécessite une surface de silicium

supplémentaire pour l'implémentation de multiplexeurs additionnels. Donc, il est nécessaire d'évaluer la rentabilité d'une telle approche par rapport à la réduction du temps de test offert.

(Tahoori, McCluskey et al. 2004) ont développé une stratégie de test simple et rapide pour les tests dépendants de l'application de l'FPGA, tout en assurant une couverture de pannes multiples de type collé-à pouvant atteindre 100%. La méthode utilisée réduit le nombre de test à trois. En effet, elle consiste à tester les interconnections et les blocs logiques d'une application implémentée dans le FPGA. La première phase se compose de deux configurations des blocs logiques : la première pour la détection des pannes collé-à 0 en modifiant les BLC en portes ET et la deuxième pour la détection des pannes collé-à 1 en modifiant les BLC en portes OU. En outre, les auteurs ont présenté une méthode de diagnostic basée sur cette technique, qui est capable de localiser la panne sur le chemin combinatoire entre deux bascules. La limitation de cette localisation apparaît lorsqu'il y a de plusieurs chemins combinatoires entre deux bascules. Dans ce cas, tous ces chemins combinatoires sont des suspects qui peuvent causer la panne. La deuxième phase consiste en une seule configuration, pour tester tous les blocs logiques utilisés. Dans cette phase, la configuration originale des blocs logiques utilisés est préservée et la configuration des interconnections et blocs logiques inutilisés est changée pour tester de manière exhaustive tous les blocs logiques utilisés. De ce fait, les auteurs ont présenté une version BIST basée sur un compteur 4-bits et une porte XOR. Pour la validation de la phase, les auteurs ont évalué l'impact d'ajouter le BIST aux circuits de tailles différentes implémentés aux FPGAs XC4000 et Virtex de Xilinx. Les résultats expérimentaux montrent que la taille de circuits BIST, en termes de taux de LUT inutilisées, est raisonnable (entre 1.32% et 15.38%). Cependant, le problème de cette conception est la congestion des signaux sortants du compteur lorsqu'il s'agit d'un design sophistiqué. Ceci cause un vrai défi pour l'observabilité des sorties issues des portes XOR, et nécessite parfois quelques configurations supplémentaires.

1.5.2 Test indépendant de l'application

En général, les circuits FPGA peuvent être programmés selon une infinité de configurations. D'un point de vue du test, certaines configurations peuvent cibler des ressources particulières dans le circuit afin de les rendre plus facilement accessible. Les différentes ressources de l'FPGA ont été visées par les chercheurs au cours des dernières années. Cependant, nous avons concentré notre recherche bibliographique spécifiquement sur le test des interconnections pour les raisons suivantes. Premièrement, près de 80% des transistors à l'intérieur d'un FPGA sont dédiés au réseau de routage programmable. Ainsi, la couverture des pannes qui affectent ce réseau avec un nombre restreint de configurations de test constitue un vrai défi. Aussi, afin de pouvoir tester les autres ressources, il faut s'assurer que les ressources d'interconnections ne contiennent pas de défectuosité.

(Renovell, Portal et al. 1998) ont présenté une approche de test pour la structure d'interconnections des circuits FPGA. Cette structure est constituée des pistes horizontales et verticales et des matrices de commutation (SM). Chaque SM est constituée de transistors permettant d'établir une connexion entre les lignes horizontales et verticales. Le principe de base de l'approche consiste à exploiter toutes les configurations possibles de la SM pour couvrir les pannes de court-circuit et circuit-ouvert à l'intérieur et à l'extérieur de la SM. En effet, la SM reçoit *n* lignes dans chacune de ses quatre faces. Ces lignes sont subdivisées en deux catégories : des lignes connectables pouvant être affectées par des courts-circuits ou par des circuits-ouverts et des lignes non connectables pouvant être affectées par des courts-circuits. En utilisant ce modèle simplifié de SM, les auteurs ont prouvé que trois configurations sont suffisantes pour couvrir 100% de ces pannes. L'application de l'une des trois configurations transforme conceptuellement le circuit en un bus de lignes nécessitant ainsi une séquence de vecteurs de test pour la détection des pannes. Les auteurs ont montré que le nombre total de vecteurs nécessaires pour une couverture complète de séquence (*configuration test sequence*, CTS = 100%) pour les trois configurations (*three test configuration*, 3*CTs) est égal à 3*[log2 (2km + 2)], m étant le nombre de SM et *k* le nombre des signaux d'une face de SM. Ainsi, il est facile de prouver que cette technique nécessite

quelques dizaines de vecteurs de test pour les FPGA contemporains. Vu que l'application d'un vecteur de test approprié pour la détection des courts-circuits présente un facteur important dans le temps global de test, l'adoption de la technique par les fabricants cause un vrai problème de rentabilité.

(Harris et Tessier 2002) ont utilisé l'architecture des FPGAs basée sur les *clusters* pour développer leur approche de test et diagnostic des courts-circuits. Cette architecture a été conçue pour faciliter la localisation des composants dans le circuit. Elle groupe des blocs logiques tels que les LUTs et les bascules dans des *clusters* identiques incluant aussi les SM. La technique proposée par les auteurs permet de couvrir les courts-circuits affectant les lignes à l'intérieur des *clusters* ainsi que celles situées entre les *clusters*. Le principe de l'approche consiste en l'utilisation de plusieurs autotests intégrés (BIST) couvrant un sous-ensemble de *clusters* et se déplaçant pour balayer la totalité des *clusters* constituant le FPGA. Chaque BIST est composé d'un générateur de test et d'un analyseur de signature pour tester chaque *cluster* individuellement. Cependant, le test est compliqué à cause de la haute densité de *cluster*. De ce fait, les auteurs ont analysé l'indépendance et l'équivalence des pannes en exprimant la détectabilité de chaque court-circuit en fonction des entrés⁄sorties du *cluster*. En outre, pour atteindre une résolution maximale de diagnostic, les auteurs se sont assuré que les pannes détectables dans une configuration ne soient pas répétées dans d'autres configurations. En implémentant les algorithmes de configuration, il est montré qu'une couverture de 100% des courts-circuits peut être garantie au coût d'une augmentation du nombre de configurations. En fait, il est montré que le nombre de configuration atteindra 30 pour des *clusters* à 8 entrées. Ce nombre de configurations est désavantageux si on tient compte du nombre de vecteurs de test à appliquer pour chaque configuration.

(Tahoori et Mitra 2005) ont présenté une technique de configuration automatique des circuits FPGA pour les tests manufacturiers indépendants des applications. Cette technique est capable de détecter 100% des pannes de circuits ouverts et des pannes des courts-circuits qui affectent le réseau d'interconnections, y compris les matrices d'interconnections (SMs). Pour ce faire, les auteurs ont divisé le test des SMs en quatre types selon leur direction, puis ils

appliquent l'algorithme de flux maximum de Ford Fulkerson (optimisation des trajectoires parcourus) ainsi que la technique d'entrelacement pour générer les configurations pour chaque direction. En se basant sur la connectivité de SM, deux configurations sont nécessaires pour couvrir une face de SM et les lignes qui y sont connectées. Ceci donne huit configurations au total pour une couverture complète de l'FPGA. Cependant, les auteurs ont considéré seulement les lignes directes dans leur test, tout en ignorant les lignes hex et les lignes doubles qui sont caractérisées par des structures différentes. Notons toutefois que selon les auteurs, le nombre de configurations est proportionnel à la somme des lignes connectées aux quatre directions. De ce fait, deux configurations par direction ne seront plus suffisantes pour les FPGA contemporains.

(Wang, Liou et al. 2005) ont proposé une approche de test de l'FPGA indépendant de l'application qui réduit significativement le temps de test qui est traditionnellement dominé par le nombre de configurations. Cette approche consiste à concevoir deux BIST (*built-in self-test)* qui peuvent fonctionner à la vitesse du FPGA pour tester les pannes de délais de propagation dans les interconnections tout en éliminant les inconvénients des BISTs basés sur la comparaison. Ces circuits BIST comportent deux parties: un générateur de patrons de test (*test pattern generation*, TPG) (registre, XOR et chemin sous test) et un contrôleur qui définit l'instant d'un test (registre et XOR). L'architecture des BISTs utilisée est fondée sur la régularité des FPGA. De ce fait, un petit circuit de test est implémenté de façon répétitive sur chaque BLC de l'FPGA. Chaque circuit de test cible un chemin spécifique (PUT) accompagné d'une boucle de retour. En effet, le circuit compare la différence entre les plus rapides et les plus lents délais de propagation dans un groupe de chemins sous test (PUTs) avec une horloge de test. Cette horloge attaque trois bascules : deux bascules pour les extrémités de PUT et la troisième pour la boucle de retour. Le circuit BIST1 est configuré pour tester les chemins dans le même BLC, alors que le circuit BIST2 est proposé pour tester les chemins entre différents BLCs de l'FPGA. En combinant les deux circuits BIST, les auteurs ont conçu un système de BIST évolutif (BIST1 & 2) pour une couverture élevée des pannes de délais (un taux de couverture qui atteint 98%. Malgré l'efficacité de cette méthode pour la détection des pannes de retards dans les interconnections, il est pratiquement

impossible de cibler d'autres types de pannes à cause de la structure des BISTs avec la boucle de retour. Cette boucle impose un type de vecteurs de test précis.

(Chmelar 2004) a présenté une technique de test qui réduit considérablement le nombre de configurations de test des circuits FPGA dans le but d'optimiser le coût des tests. Cette technique de test est basée sur l'architecture des FPGA contemporains. En effet, ces FPGA sont constitués essentiellement de blocs logiques et des interconnections. Les interconnections sont composées de matrices de commutation (SMs) et de points d'interconnections programmables (*programmable interconnect point*, PIPs). L'auteur a utilisé des FPGA Virtex-II pour tester ces familles. Il montre ainsi que l'activation de plusieurs voies logiques peut réduire le nombre de configurations de test. De ce fait, il a prouvé que seulement huit configurations sont nécessaires pour couvrir 100% des pannes collé-à, PIP collé-on et PIP collé-off pour les FPGA utilisés. Cependant, cette technique s'appuie sur l'hypothèse de l'observabilité et de la contrôlabilité des voies logiques (ou des interconnections), ce qui n'est pas évident en pratique face à la complexité de ces derniers FPGA.

(Tahoori et Mitra 2006) ont introduit une comparaison entre les techniques de test des ASICs et des FPGAs afin de définir le taux de compression des configurations de test pour ces circuits. Il en a résulté un énorme écart entre les taux de compression pour les ASIC (20x-50x) et FPGA (aucun). Les auteurs ont présenté alors la première technique de compression des configurations de test des FPGA dans le but de réduire le volume de données et le temps de chargement de test. L'approche s'appuie sur la régularité des FPGA et sur la méthode de configuration de test des interconnections (verticale, horizontale et diagonale) proposé par (Tahoori 2005). Ainsi, la méthode consiste à générer des configurations de tests composés d'un tableau de modules de test reproductibles (RTM) identiques. De ce fait, au lieu d'enregistrer ou de charger l'ensemble des données de test de configuration, seuls les RTMs sont stockés ou chargés dans l'FPGA. En comparant cette technique avec celle de (Tahoori 2005), les auteurs ont montré qu'un taux de compression entre 7.3x et 117x peut être atteint pour la famille Virtex de Xilinx. Ils ont également ajouté des modifications à la configuration

afin d'accélérer le temps de chargement. Ces modifications sont basées sur le chargement simultané des RTM via des registres parallèles. Il est prouvé que cette technique de compression est indépendante des techniques de configuration utilisées dans la littérature. Il s'agit simplement d'une optimisation des bits de configuration utilisés dans ces techniques. Donc, cette approche pourrait être également utilisée pour optimiser la méthode proposée dans ce mémoire, dans une étape ultérieure.

1.5.3 Tests I_{DDQ}

Étant donné que les FPGA utilisent la même technologie de fabrication que les ASIC, il est évident qu'elles partagent aussi les mêmes pannes et les mêmes techniques de test (Rajsuman 2000). De ce fait, on remarque dans la littérature la présence de la technique de test I_{DDQ} pour la couverture des courts-circuits et des circuits ouverts dans ces puces programmables. En plus, cette méthode ne souffre pas des limitations imposées par l'observabilité des nœuds puisqu'elle est garantie par les mesures du courant. Il est donc possible d'avoir un nombre réduit de configurations. Cependant, le processus de mesure de courant est relativement lent et reste la contrainte dominante.

Dans ce contexte, (Zhao, H. et al. 1998) ont adapté la technique I_{DDQ} pour couvrir 100% des courts-circuits qui affectent les BLC des FPGA. Ils ont suggéré deux stratégies de configuration pour tester simultanément chaque BLC. La première propose de tester les pannes externes entre les modules combinatoires et ceux de la logique séquentielle (les bascules, les multiplexeurs, les LUT). La deuxième teste les pannes internes dans les modules implémentés dans le BLC. En outre, une approche de génération du test appelée *bottom-up* a été appliquée afin de minimiser le nombre de configurations puis de vecteurs de test. Pour ce faire, une librairie de test contenant les configurations et les vecteurs de test pour les BLC a été développée. Les mêmes stratégies ont été adoptées pour le test des IOB dans (ZHAO, H. et al. 1999).

Pour tester les interconnections, les mêmes auteurs ont proposé dans (Zhao, H. et al. 1998) un modèle d'interconnections constitué des matrices de contrôle et des lignes de connection.

Par la suite, en se basant sur la topologie de deux blocs du modèle, les configurations et les vecteurs sont générés parallèlement. Ainsi, vue l'homogénéité du modèle, les auteurs ont pu réduire le nombre de configuration qui restent néanmoins encore un peu élevé (20 configurations et 11 vecteurs de test pour Xilinx XC4000).

Face à la difficulté de distinguer un faible court-circuit à partir d'un grand courant de fuite, (Chmelar et Toutounchi 2004) ont utilisé le courant différentiel DI_{DDQ}. L'approche de génération des configurations de test consiste à diviser les interconnections en deux groupes. Un groupe des interconnections sera affecté au 0 logique. Un deuxième groupe qui correspond à l'interconnection adjacente au premier groupe sera affecté au 1 logique. Ainsi, cette configuration permet d'activer un court-circuit entre les deux groupes et de localiser la panne. Dans cette approche, le nombre de configurations atteint cinq pour couvrir toutes les interconnections de Virtex II. Ainsi, à l'application de deux vecteurs de test pour chaque configuration, les auteurs ont été capables de détecter un court-circuit de 1kΩ. En se référant aux résultats obtenus pour l'application de $C\Delta I_{DDQ}$ décrite précédemment, on peut confirmer que cette limite de détection ne présente pas la valeur optimale.

1.5.4 Sommaire

Il est nécessaire de faire ressortir les meilleures stratégies de test décrites précédemment. Ceci nous permettra de justifier la contribution apportée par notre proposition. De ce fait, nous présentons au Tableau 1.1 une récapitulation des meilleurs résultats obtenus pour les stratégies de test des interconnections des puces FPGA développées de nos jours. Le choix de ces stratégies s'appuie en fait sur les résultats mentionnés dans les trois dernières colonnes du tableau. Il s'agit du taux de couverture (Taux de Couv., proportion des pannes détéctés), du nombre de configurations (Nbre. Conf.) et du nombre de vecteurs de test pour chaque configuration (Nbre.Vec./Conf).

On remarque dans ce tableau qu'il existe des méthodes de test efficaces pour la détection des pannes de retard et de collé-à pour les tests dépendants et indépendants de l'application.

Dans les deux modèles de pannes, les méthodes se concentrent sur la minimisation du nombre de configurations sans tenir compte du nombre de vecteurs de test. Ceci est dû à l'aspect logique de l'opération de détection de panne. Cette opération est caractérisée par une vitesse d'application d'un vecteur de test similaire à la vitesse de fonctionnement optimal du FPGA. Par contre, pour la détection des pannes de court-circuit, les auteurs ont toujours eu tendance à minimiser le nombre de vecteurs de test. Ce nombre reste toujours élevé pour les tests dépendants de l'application, même que la proposition de (Renovell, Faure et al. 2001) s'appuie sur l'émulation d'un ASIC sur une architecture simplifiée d'un FPGA pour un taux de couverture de 80%. Dans ce cadre, nous proposons d'émuler un ASIC sur un FPGA commercial et d'appliquer en suite la technique de test $C\Delta I_{DDQ}$. Cette proposition nous permettra en suite d'avoir un taux de couverture qui atteindra plus de (Thibeault et Hariri 2009) 90%. Pour les tests indépendants de l'application, la stratégie de (Chmelar et Toutounchi 2004) est la meilleure solution jusqu'à présent. L'inconvénient de cette technique est qu'elle est conçue pour l'application de la technique ΔI_{DDQ}. Ceci signifie qu'elle ne permet pas d'ajouter les vecteurs complémentaires de test et de bénéficier des avantages offerts par $C\Delta I_{DDQ}$. Donc, nous proposons en conséquence une nouvelle configuration plus souple qui offre la possibilité de l'application de la technique de test cible tout en ayant un nombre de vecteurs de test très réduit.

Tableau 1.1 Récapitulation sur les techniques de test pour les FPGA contemporains

Modèle de panne	**Technique de test**	**Stratégie de test**	**Taux de Couv.**	**Nbre. Conf.**	**Nbre.Vec./ Conf**
Retard	dépendante	(Krasniewski 1999)	99%	1	>100
	indépendante	(Wang, Liou et al. 2005)	100%	20	3
Collé-à	dépendante	(Tahoori, et al. 2004)	100%	3	>60
	indépendante	(Chmelar 2004)	100%	8	6
Court-circuit	dépendante	(Renovell, Faure et al. 2001)	80%	1	>200
	indépendante	(Chmelar et Toutounchi 2004)	100%	5	2

1.6 Conclusion

Dans ce chapitre, nous avons présenté l'architecture des circuits configurables de type FPGA et un état de l'art dans le domaine du test manufacturier des circuits intégrés. Ainsi nous avons passé en revue les principales contributions concernant le test de ces circuits. Certains chercheurs ont tiré profit du fait que les FPGA partagent les mêmes propriétés physiques et les mêmes types de pannes que les ASIC pour adapter aux circuits FPGA des techniques de test déjà développées pour les circuits ASIC.

Dans le prochain chapitre, nous proposons l'application de l'approche de test $C\Delta I_{DDQ}$ développée pour l'ASIC (émulé dans un FPGA), équivalant à un test dépendant de l'application. Ceci constitue la première étape vers l'application de ce test aux FPGA de manière indépendante de l'application.

CHAPITRE 2

APPLICATION DE L'APPROCHE CDI_{DDQ} PAR L'ÉMULATION DE CIRCUITS SOUS TEST ASIC PAR DES CIRCUITS FPGA

2.1 Introduction

Dans ce chapitre, nous allons présenter l'adaptation de la technique de test CDI_{DDQ} aux tests dépendants de l'application des circuits FPGA. Pour ce faire, il est indispensable d'introduire en premier lieu la théorie ainsi que les algorithmes et les outils commerciaux permettant l'élaboration de la technique en mode ASIC. En deuxième lieu, nous définissons la stratégie de conversion de la technique du domaine ASIC vers le domaine FPGA. Nous visons dans cette étape à simuler logiquement le comportement d'un circuit dans la phase de test CDI_{DDQ} à l'aide de l'outil de synthèse et de simulation des circuits FPGA de la famille Xilinx, ainsi qu'à émuler ce circuit sous test dans le circuit FPGA. Pour cette dernière tâche, nous proposons et utilisons une nouvelle version d'un testeur développé au LACIME pour l'implantation et la validation expérimentale de l'approche.

2.2 Mise en place et principe de la technique CDI_{DDQ} en mode ASIC

Au chapitre 1, nous avons brièvement introduit la technique de test CDI_{DDQ} pour les ASIC. Cependant, afin de pouvoir l'utiliser et l'adapter aux circuits FPGA, il est nécessaire de comprendre l'impact de l'utilisation d'une telle technique dans le processus traditionnel de conception et de comprendre la théorie sur laquelle elle s'appuie. Notons au départ que la technique a été développée en minimisant l'impact sur ce flot de conception. La figure 2.1 présente ce processus et les modifications requises par le processus de génération des vecteurs de la technique $C\Delta I_{DDQ}$:

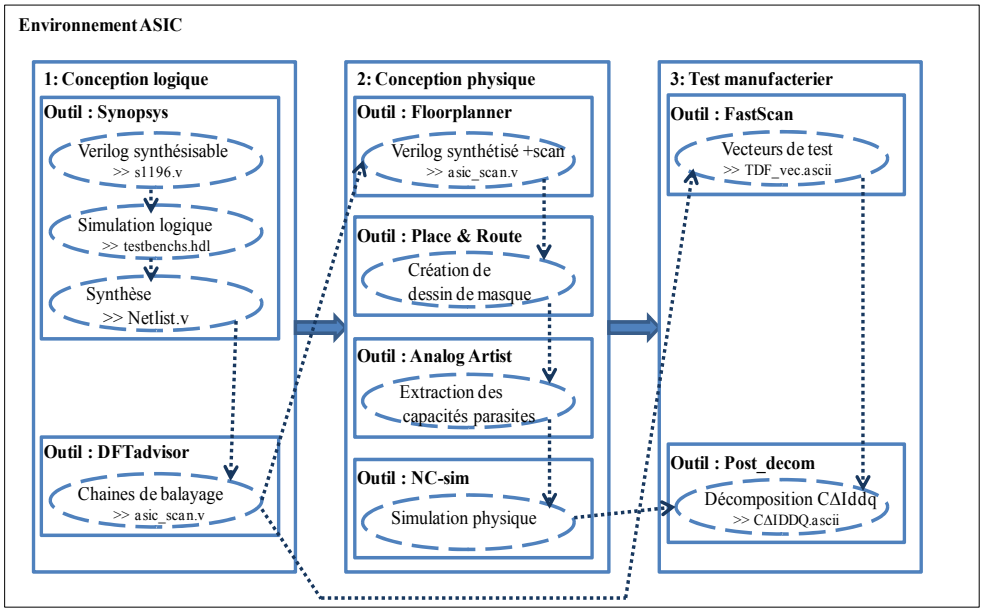

Figure 2.1 Processus de conception et génération des vecteurs de test $C\Delta I_{DDQ}$

Comme la Figure 2.1 l'indique, la première étape est celle de la conception logique et consiste à écrire du code « Verilog » synthétisable (pour le circuit « s1196.v », dans cet exemple), à simuler et à synthétiser ce code. Dans ce cas précis, l'utilisation de l'outil « Synopsys » permet de générer le fichier netlist propre à ce circuit « s1196.v ». Ensuite, on passe à l'insertion des circuiteries (registres) de balayage qui est assurée par l'outil « DFTadvisor » de Mentor. L'étape de conception physique permet par la suite de créer le dessin de masque à l'aide des outils de conception « Floorplanner » et « Place and Route » de Cadence. Une fois la conception physique terminée, l'outil « Analog Artist » de Cadence extrait les capacités parasites qui seront utiles pour la génération des vecteurs de test. Notons que cette extraction fait partie du flot conventionnel, afin d'estimer plus précisément la fréquence d'opération maximale et la puissance dissipée. Pour générer les vecteurs de scan adéquats pour le circuit de balayage, « FastScan » de Mentor permet de produire des ensembles différents de test «TDF_vec.ascii ». Finalement, l'algorithme de décomposition des vecteurs TDF «Post-decom » génère le fichier «CΔIDDQ .ascii » qui contient la totalité des vecteurs de la technique, et qui tient compte des capacités parasites extraites pour identifier les sites réalistes potentiels des courts-circuits.

2.2.1 Phases de synthèse logique et physique

La synthèse logique est une étape de conception qui consiste à transformer la description comportementale d'un circuit en un domaine structurel sous forme d'un réseau de portes logiques. Cette étape est maintenant réalisée automatiquement à l'aide d'outils de synthèse commerciaux très efficaces tels que Design Compiler de Synopsys, BuildGates, RC de Cadence, et Leonardo de Mentor Graphics. L'outil de synthèse effectue une compilation logique associée à des contraintes de temps définies dans la spécification d'un circuit logique numérique. Suivant les contraintes imposées, on obtiendra un CI optimisé spécifiquement en surface ou en vitesse. Pour ce faire, l'outil s'appuie sur des algorithmes d'optimisation booléenne pour réduire la complexité du modèle au niveau transfert de registre (RTL. « *Register Transfer Level* »). Ces algorithmes cherchent à limiter la redondance des calculs, tout en préservant la fonctionnalité du circuit. Une fois les expressions booléennes optimisées, l'outil de synthèse utilise une bibliothèque de cellules logiques prédéfinies pour effectuer la conversion technologique. Cette bibliothèque est constituée des portes logiques fournies par les fabricants pour refléter la technologie cible. Ainsi, la phase de conversion consiste à décomposer les expressions booléennes en fonction de portes logiques simples (ET, OU...) ou complexes à plusieurs entrées.

Finalement, le schéma obtenu par la synthèse logique sera accompagné par une liste des interconnections (netlist) qui résume tous les points de connection entre chaque composant du circuit. Par la suite, le réseau électrique décrit dans le netlist peut être utilisé par les applications de synthèse physique, telles que les logiciels de placement et routage, ou par les logiciels de simulation. Les étapes de placement et de routage consistent à fixer la position des différents éléments du circuit dans un plan et de connecter physiquement ces éléments. Ces deux étapes déterminent les performances finales du circuit. L'extraction des capacités parasites, la simulation physique et l'estimation statique de la puissance dissipée (non illustrée à la Fig. 2.1) permettent de valider l'étape du placement et le routage. Le simulateur incorpore sous forme de délais l'impact des ses capacités parasites extraites pour réaliser une

simulation proche du circuit final. Ces capacités sont également utilisées pour l'estimation statique de la puissance et la génération des vecteurs de test CDI_{DDQ}.

2.2.2 Chaîne de balayage

L'évolution de la densité d'intégration a forcé l'utilisation de techniques facilitant le test des puces. Une des ces techniques, appelée mux-scan, est présentée à la Figure 2.2. Dans laquelle on retrouve en a) un circuit séquentiel simple, et en b), le même circuit avec une chaine de balayage de type mux-scan. Il s'agit simplement d'attacher des multiplexeurs aux bascules du circuit. Ces multiplexeurs permettront donc de piloter l'état de chaque bloc logique à travers l'entrée *scan enable*. L'insertion de la chaine de balayage est automatisée par les outils de test spécifiques tel que DFTAdvisor.

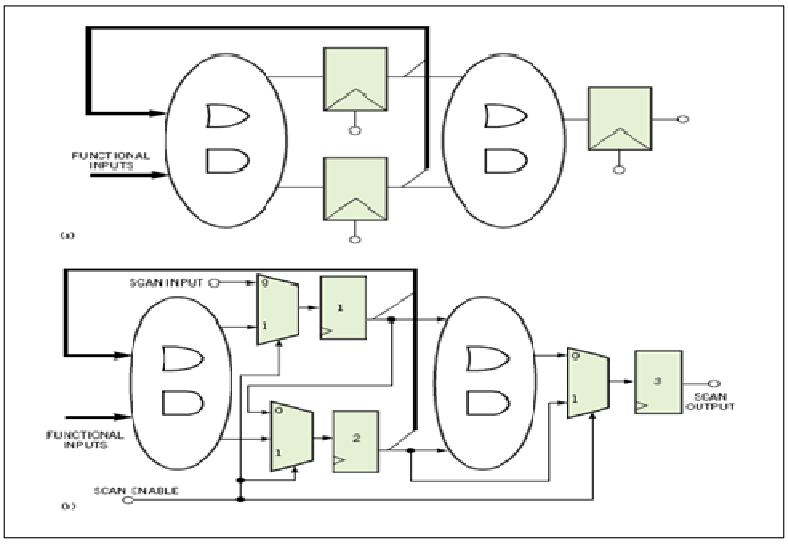

Figure 2.2 ASIC implémentant un circuit de balayage
Tirée de Ken Jaramillo (2000, p. 68)

2.2.3 Vecteurs de test

L'insertion de la chaîne de balayage impose deux types d'entrées/sorties (E/S) : les E/S dédiées au fonctionnement normal et les E/S dédiées au test et à la chaine de balayage. De ce fait, l'outil ATPG décompose les vecteurs en deux parties : la première représente l'ensemble des valeurs en entrée avec les valeurs attendues en sortie, alors que la deuxième

partie représente les valeurs à injecter à l'entrée de la chaîne de balayage, suivies des valeurs attendues à la sortie de la chaîne. Comme le présente la Figure 2.3, chaque partie de vecteur est accompagnée d'une chaine de bits d'information (*pattern info*). Ces bits sont utilisés pour déterminer le numéro du patron associé au vecteur, le numéro du cycle et le type de vecteur (d'entrée/sortie ou de balayage). Ces informations seront classées dans deux types de fichiers :

- un fichier séquence qui définit la séquence de pilotage des multiplexeurs et des registres dans la chaine de balayage. Ce pilotage synchronise le forçage des différentes entrées du circuit aux valeurs prédéfinies à des instants précis;
- un fichier valeur qui contient les valeurs des entrées prédéfinies à chaque forçage accompagnées par des valeurs de sorties correspondantes.

Les deux fichiers sont par la suite utilisés dans la phase de simulation. Durant cette phase, le simulateur exploite le comportement physique du netlist suite à l'exécution de ces deux fichiers. Il compare ainsi les valeurs prévues pour les sorties avec les résultats obtenus et génère son rapport final.

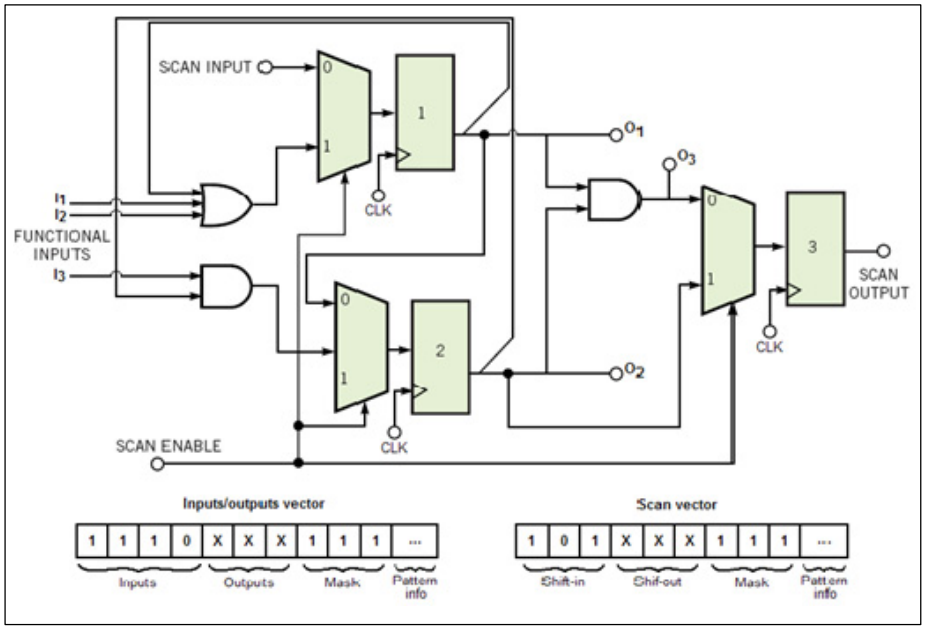

Figure 2.3 Exemple de vecteur de test
Tirée de Ken Jaramillo (2000, p. 69)

2.2.4 Technique CDI_{DDQ} et algorithme de décomposition

Comme cela a été mentionné précédemment, la technique de test CDI_{DDQ} utilise des patrons de test de panne de transition (TDF) pour activer les pannes des courts-circuits, et combine les valeurs de courant I_{DDQ} mesurées après l'application de chaque vecteur. Pour ce faire, on procède à la décomposition d'une paire de patrons TDF conventionnels (Vw et Vx, Figure 2.4) afin d'obtenir des patrons dits complémentaires qui permettent l'activation de la panne (Vy et Vz, Figure 2.4). Théoriquement, en respectant les conditions environnementales (voltage et température stables), il est prouvé que la combinaison $C\Delta I_{DDQ}$ du courant de fuite de quatre patrons obtenue est égale à zéro dans en l'absence d'une panne de court-circuit,

$$C\Delta IDDQ(w,x,y,z) = I(V_Y) + I(V_Z) - I(V_W) - I(V_X) = 0 \tag{2.1}$$

Dans le cas contraire, il y a deux possibilités dépendamment si la panne transitoire ayant comme nœud M et N a été activée ($M \neq N$) ou non ($M = N$) par Vw. Mais pour chacune des deux possibilités, on retrouve le courant consommé par la panne reliant les nœuds M et N. Ceci est traduit par l'équation suivante :

$$\left|C\Delta IDDQ(w,x,y,z)\right| = I_{br,01} + I_{br,10} \tag{2.2}$$

Où I_{br} présente le courant consommé par le court-circuit. Les vecteurs de test générés par l'ATPG ciblent, selon le choix de type de test, des pannes précises pouvant se produire dans la puce. Pour la technique CDI_{DDQ}, ces pannes correspondent aux nœuds entre lesquels apparaissent des capacités parasites, indiquant la proximité de ces nœuds et identifiant du même coup les sites potentiels pour les courts-circuits. Le rôle de l'algorithme de décomposition consiste alors à adapter les vecteurs TDF à la technique de test CDI_{DDQ} tout en gardant le taux de couverture offert par le TDF. La Figure 2.4 montre un exemple de décomposition des vecteurs. Suite à une analyse approfondie du circuit, l'outil ATPG génère deux patrons de test TDF (V_W et V_X), couvrant les nœuds *M* et *N*, identifiés comme sites

potentiels de court-circuit de par la présence de la capacité parasite (Cp). L'algorithme de décomposition analyse l'indépendance de chaque entrée du circuit aux deux nœuds M et N. Par la suite, il associe chaque entrée à un des deux groupes de vecteurs qui reflète son degré d'indépendance. Finalement la génération de deux autres vecteurs (V_Y et V_Z) résulte de la décomposition et associe chacun de ces 2 nouveaux vecteurs à ses groupes:

$$V_W = \left[\{V_{W,\alpha}\},\{V_{W,\beta}\},\{V_{W,\chi}\},\{V_{W,\delta}\}\right]$$
$$V_X = \left[\{V_{X,\alpha}\},\{V_{X,\beta}\},\{V_{X,\chi}\},\{V_{X,\delta}\}\right]$$
$$V_Y = \left[\{V_{X,\alpha}\},\{V_{W,\beta}\},\{V_{X,\chi}\},\{V_{X,\delta}\}\right]$$
$$V_Z = \left[\{V_{W,\alpha}\},\{V_{X,\beta}\},\{V_{W,\chi}\},\{V_{W,\delta}\}\right]$$

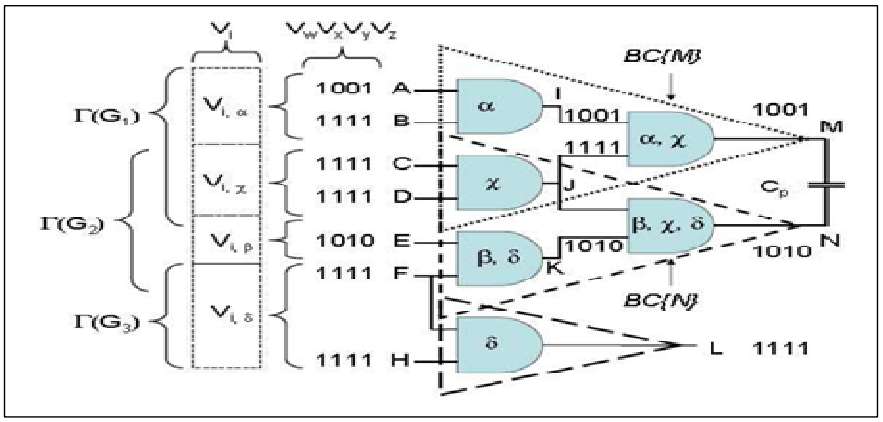

Figure 2.4 Algorithme de décomposition $C\Delta I_{DDQ}$
Tirée de Claude Thibeault et Yassine Hariri (2011, p. 26)

Dans l'exemple de la figure 2.4, la première paire de vecteurs (V_W et V_X) crée 2 transitions descendantes, A↓I↓M↓ et E↓K↓N↓. La décomposition de cette paire de vecteurs fait en sorte que l'application individuelle de chacun des 2 vecteurs supplémentaires (V_Y et V_Z) après le premier vecteur (V_W) ne permet qu'une seule des deux transitions, A↓I↓M↓ pour V_Y et E↓K↓N↓ pour V_Z.

2.3 Conversion ASIC /FPGA

Le passage du domaine ASIC vers le domaine du FPGA n'est pas trivial, même lorsqu'il s'agit de la même technologie de fabrication. L'analyse des paramètres de sortie du domaine ASIC est une étape fondamentale pour établir la conversion. Ces paramètres ont été définis

dans la première partie de ce chapitre. Reste à les exploiter et à les décoder pour établir la conversion adéquate. En fait, deux types de fichiers se présentent à la sortie:

- le fichier netlist : il s'agit de la description logique de circuit à implémenter et à tester. Ce circuit est caractérisé par sa conception en vue de test. Il inclut donc la chaine de balayage dans son code;
- les fichiers de simulation : ces sont les deux fichiers générés par l'ATPG pour simuler le comportement du netlist pour un type de test donné. Un des fichiers définit le déroulement de la séquence de test, alors que l'autre sert à enregistrer les valeurs des vecteurs de la séquence.

Ces fichiers ne sont pas compatibles avec les outils de synthèse et simulation des FPGA. Il est donc indispensable de les décoder afin de les implémenter convenablement.

2.3.1 Conversion de netlist du domaine des ASIC vers celui des FPGA

Nous nous limitons dans ce chapitre à valider expérimentalement les résultats obtenus lors du développement de la technique de test $C\Delta I_{DDQ}$, sur la base de circuits séquentiels du *benchmark* ISCAS 89 et de filtres de type FIR. Après la phase d'insertion des registres à balayage (scan), nous obtenons un netlist en format porte logique codé en Verilog. Ce code est utilisé par les outils de placement et routage pour produire le dessin des masques de la puce. Malheureusement, chaque manufacturier utilise souvent une syntaxe qui lui est propre pour l'encodage des netlists, ce qui peut les rendre incompatibles avec ceux des autres manufacturiers, y compris ceux des FPGAs. Cette incompatibilité est due à l'utilisation des bases de données de conception (DDB) différentes pour décrire les portes logiques. L'analyse de la solution de rechange, à savoir l'utilisation du format d'échange de conception EDIF, qui normalise la conversion de netlist d'un fabricant à un autre, a révélé que cette solution posait tout autant de difficultés que l'approche de conversion du netlist Verilog. C'est donc vers cette dernière approche de conversion du netlist que nous nous sommes finalement tournés. Le code du netlist est généré par les outils de synthèse logique. Il est basé sur la description matérielle du circuit au niveau logique.

À ce niveau, les caractéristiques d'un système sont décrites par des liens logiques avec leurs propriétés temporelles. Tous les signaux sont des signaux discrets («0», «1», «X», «Z»). Les opérations utilisables sont des primitives logiques prédéfinies (ET, OU, NON, etc.). L'échange des netlists entre les outils de synthèse de différents fabricants est laborieux pour les raisons suivantes :

- l'absence d'un standard Verilog de netlist qui normalise la syntaxe;
- la différence entre les modèles internes dans les netlists des fabricants.

Ce problème oblige les concepteurs soit à manipuler manuellement les netlist, soit à développer des scripts de décodage automatique des syntaxes. Dans les deux cas, la même stratégie d'adaptation est adoptée. Cette stratégie consiste à extraire toutes les informations nécessaires du netlist source et à les classer a priori dans une base de données de conception (DDB). Par la suite, à chaque information est associée la traduction correspondant au netlist cible. Enfin, la structure de base du netlist cible est adaptée avec les traductions obtenues. La classification des informations dans la DDB selon des classes et des sous-classes précises constitue un facteur important pour l'accélération de processus d'adaptation. En fait, en se basant sur la syntaxe générale du langage Verilog, chaque classe devrait décrire une famille d'informations ayant généralement les mêmes fonctions. Les familles principales des informations sont présentées au tableau 2.1.

Tableau 2.1 Classification des familles des données en Verilog

Classe	**Sous classes**	**Mots clés de code**
Type de données	Ports	*input a, b, c;* *output d;* *inout e;*

	Signaux	*wire[2:0] f; //vecteur f[2], f[1], f[0]*
		tri[7:0] bus; //bus de 8-bit à état de tri
		integer i; //entier codé 32-bit
		reg r; //registre d'un bit

	Paramètres	*type[range] g;*

Classe	**Sous classes**	**Mots clés de code**
Circuits combinatoires	Portes logiques	*and A1(w1, u, s);not N1(ns, s);* *and A2(w2, v, ns); or O1(f, w1, w2);*
	Affectations	*assign w1 = a ^ b;*
Opérateurs	Bits	*'x', 'z', '0', '1'*
	Logiques	*false, true*
	Réducteurs	*y = &x*
	Arithmétiques	**,-,/,+,--,++,==,!==,…*
Structures de contrôle	Boucle toujours	*Always @(sensitivity_list)* *[begin] […] [end]*
	Condition si	*if(t == y) z = 12; else z = 22;*
	Condition selon	*case (t)* *0: y = w[0]; ..7: y = w[n]; default: y = x;* *endcase*
	Boucle pour	*for(i=0;i<10;i=i+1) begin* *s[i] = p[i] ^ q[i]; end*
	Boucle tant que	*while(condition) begin… update end*
Modules séquentiels	D-classique	*module classicD (D, clk, Q, Qn)(..)endmodule*
	D-asynchrone	*module DFFAsyncC (D, clk, rst, Q, prst)(..)endmodule*
	JK-classique	*module jkff(J, K, clk, Q);(..)endmodule*

Après avoir défini les mots clés et leurs classes correspondantes, le processus d'extraction des informations et de remplissage de la DDB est exécuté selon la structure de base du langage Verilog. Cette structure constitue l'architecture de base de description d'un circuit ou

des sous-circuits d'un netlist donné. Chaque circuit ou sous-circuit correspond à un module qui prend la structure suivante :

module *nom_module* ***(ports****_listes des noms);[****declarations*** *des données];*
*[****affectations*** *états];*
*[****initialisations*** *block];*
[block ***always****];*
[instanciations des portes];
[instanciations des modules];
fin module

Toutefois, les informations liées à un module donné seront associées à une DDB unique à ce module qui définit son degré d'indépendance dans le code. Par exemple, les informations du module principal sont associées à la DDB-pr, alors que les informations des sous-modules (1,2,..,*n*) du module principal sont associées aux (DDB-pr-1, DDB-pr-2, DDB-pr-.., DDB-pr-*n*), où *n* constitue le nombre des sous-modules dans le module principal.

Finalement, les codes appropriés de chaque DDB sont générés en respectant la syntaxe de la technologie cible (Xilinx). Le netlist final est produit par l'assemblage des codes générés selon la structure donnée précédemment (voir ANNEXES I et II pour les netlists ASIC et FPGA).

2.3.2 Décodage des fichiers de simulation

L'outil Fastscan est utilisé pour la validation de l'approche de test CDI_{DDQ}. Cet outil ne permet pas en fait de visualiser facilement l'allure des signaux lors de simulations. Cependant, le rapport final de simulation est accessible par les concepteurs. Ce rapport est obtenu suite à l'exécution des fichiers de séquences et de valeurs mentionnés à la section 2.2.3. Afin de concevoir le testeur expérimental approprié à la technique, il est nécessaire de comprendre le fonctionnement du simulateur logique.

2.3.2.1 Décryptage des fichiers de séquence

Le format des fichiers de séquences fournis par Fastscan est de type ASCII (voir ANNEXE III pour un extrait de fichier). Ce fichier ASCII est divisé en cinq parties: l'entête, la configuration, les données de balayage, les données de test et les cellules de registre à balayage. L'analyse du contenu de chaque partie est indispensable pour reconstruire algorithmiquement une séquence de test complète et compatible avec la technologie visée. Ainsi, les informations contenues dans chaque partie sont décrites dans ce qui suit.

- Entête : Les données de l'entête sont des informations générales associées au fichier. Il s'agit d'une section pour l'identification des différents paramètres qui caractérisent le processus de génération de patrons de test à savoir : le modèle de l'outil ATGP, le nom de circuit, la date de création, les statistiques (couverture des pannes, nombre des pannes et des patrons de test…), le paramétrage (l'environnement de l'ATGP) et des messages sur le contenu de circuit (nombre des pins, les horloges, etc.);
- Configuration (setup_data) : La section de configuration contient la description générale de la procédure de test. Cette description inclut en premier lieu la déclaration des bus des entrées primaires (PI) et des sorties primaires (PO) contenues dans le circuit. En second lieu, le fichier introduit les domaines d'horloge utilisés. Par la suite, nous passons à la configuration de la chaine de balayage. Cette configuration définit la longueur et les valeurs des entrées (SCI) et les sorties (SCO) associées à la chaine. En dernier lieu, on définit la liste des événements à réaliser durant le test. La liste des événements appelle des commandes spécifiques à appliquer aux différentes données utilisées telles que les données de balayage, les données de test et les cellules de registre scan. Les commandes utilisées peuvent être des forçages, des chargements, des déchargements et des mesures qui seront toujours accompagnés des temps d'exécution;
- Données de balayage : Seulement les commandes de chargement et de déchargement sont admissibles pour les données de balayage. Lors de l'exécution de la commande de chargement, le processus de test place les valeurs de SCI bit par bit dans la chaine de balayage. À chaque front d'horloge, chaque bit téléchargé se déplace dans le circuit pour laisser sa place au bit suivant et balayer ainsi tous les registres dans la chaine

sélectionnée. Cette opération est nommée " mode de déplacement des bits (Shift)". Juste après cette opération, on applique le vecteur chargé et on exécute la commande de déchargement pour vider la chaine de balayage et la préparer pour une nouvelle opération;

- Données de test : Les données de test contiennent les patrons générés par FastScan. Chaque patron de test est constitué essentiellement d'une liste des événements ordonnés chronologiquement. En effet, dans chaque événement, le patron de test définit la valeur à affecter à la commande et son temps d'exécution. Évidement, les valeurs définies dans cette section respectent les dimensions des valeurs affectées dans les sections précédentes;
- Cellules du registre à balayage : Les cellules du registre à balayage sont utilisés pour le balayage dans le processus de test. La configuration des registres est nécessaire avant la mise en marche de processus de test. Cette configuration consiste à définir le mode de fonctionnement de chaque registre (inverseur ou non-inverseur).

2.3.2.2 Décryptage du fichier de valeurs

Le fichier des valeurs contient toutes les valeurs à utiliser dans le processus de test. À travers ces données, le simulateur peut vérifier l'exactitude des valeurs injectées lors de simulations et comparer par la suite les valeurs obtenues avec celles attendues. L'analyse du contenu de ce fichier sert à mieux comprendre la séquence de test. Ainsi, le développement d'un émulateur matériel se base sur ce fichier. L'extrait suivant d'un fichier de valeurs nous éclaire sur les données numériques traitées par le simulateur (voir le fichier complet à l'ANNEXE IV).

```
// Pattern 3 Cycle 25 Timestamp 1120 ns  (1)
      1010100100001111011XXXXXXXXXXXXXXXX00000000000000011010  (2)
      1110100111010000101010011101000010001111111111111111111001  (3)
```

```
      (1) Information
      Pattern x :    Numéro du patron courant
      Cycle x :     Numéro du cycle d'exécution
      Timestamp x : Temps écoulé depuis le début de la simulation
```

```
      (2) Vecteur de test
      [101010010000111011][XXXXXXXXXXXXXXX][000000000000000][1][1][010]
      [PI_bits]           [PO_bits]         [MASK_bits]      [1] [2] [3]
      [1] : [Increment_bit]
      [2] : [Timeplate_bits]
      [3] : [PatType_bits]
```

```
      (3) Vecteur scan
      [111010011101000010][101001110100001000][1111111111111111][001]
      [SI_bits]           [SO_bits]            [MASK_bits]         [1]
      [1] : [PatType_bits]
```

Figure 2.5 Extrait du fichier des valeurs et interprétation (1,2 et3)

Dans l'extrait ci-dessus, on remarque que chaque patron contient trois types de lignes : ligne d'information, ligne de vecteur de test et ligne de vecteur scan. Lorsque le simulateur est appelé, il procède à une lecture ligne par ligne pour en ressortir le numéro de patron, le numéro du cycle, l'intégrité des vecteurs et ainsi de suite. Une fois que ces données sont prises, il en tire les valeurs à appliquer sur le circuit bit par bit. Les annexes V et VI décrivent plus en détails les paramètres utilisés dans les vecteurs.

2.4 Conception d'un testeur CDI_{DDQ}

Suite à une analyse approfondie des fichiers de séquences et de valeurs, nous avons pu extraire l'enchainement des événements de test et affecter des valeurs aux paramètres utilisés. Ainsi, nous avons distingué les domaines d'horloge utilisés pour déclencher de tels événements. Ceci nous permet en conséquence d'exploiter les spécifications comportementales de testeur semi-automatique utilisé. Ainsi, l'analyse comportementale

nous permet d'obtenir le code VHDL approprié du testeur, donc, de concevoir en conséquence le testeur CDI_{DDQ} adéquat.

2.4.1 Étude comportementale

La première phase de l'étude comportementale consiste à définir la liste des événements qui constituent la procédure de test. En effet, en s'appuyant sur la figure 2.3 et l'analyse des fichiers de test, nous avons conclu que cette liste respecte l'ordre suivant :

E1- Forçage des entrées primaires PI et chargement de chaine de balayage SCI.

E2- Forçage des entrées primaires PI et mesure des sorties primaires PO.

E3- Pulsation d'horloge et déchargement de chaine de balayage SCI.

E4- Fin du cycle.

En se basant sur la liste ci-dessus, nous définissons deux domaines d'horloge à savoir : une horloge principale pour synchroniser le balayage des événements (E1, E2, E3 et E4) et une horloge secondaire pour synchroniser les sous-événements dans chaque événement. L'horloge principale (CLK_SEQ) (ou le séquenceur) est une horloge interne du le testeur qui génère une boucle des fronts pour le pilotage des événements. La fréquence de cette horloge dépend de l'accomplissement des tâches des sous-événements dans le cas du chargement et déchargement des chaînes de balayage. En outre, la fréquence dépend aussi de la technologie utilisée pour la mesure des sorties primaires. Suite au déclenchement d'un événement principal, le testeur fait appel à l'horloge secondaire (CLK) pour synchroniser les sous-événements dans le CUT. Le rôle de la première liste de sous-événements consiste à forcer les entrées primaires du CUT à des valeurs prédéfinies et à charger en parallèle la chaine de registres à balayage. Donc, le nombre de fronts à utiliser dans cette tache est égal au nombre de registres à balayage utilisés. Reste à définir la vitesse de chargement qui dépend en toute évidence du circuit FPGA utilisé. Le chargement de la dernière bascule déclenche le deuxième événement principal durant lequel nous devrons faire les mesures appropriées. Pour ce faire, nous désactivons la chaine de registres à balayage pour retourner au fonctionnement normal de circuit (SCE='0'). Afin de comprendre la chronologie à adopter pour faire les mesures, il est important de se rappeler du concept de base de la technique

CDI_{DDQ}, qui respecte la même chronologie de mesure que pour le test des pannes de transition. Il en résulte que quatre mesures sont requises pour valider l'approche, à commencer par deux pour la première paire de vecteurs : une mesure avant la transition et une après la transition. Par contre, la troisième et la quatrième mesure se font dans deux autres cycles de test. Chacune de ces deux dernières mesures est effectuée après la transition. Dans tous les cas, une zone morte doit être insérée afin d'atteindre le régime permanent et de faciliter l'opération de mesure. À la suite de la deuxième zone de mesure, nous passons à la dernière phase qui consiste à faire la pulsation de déchargement de la chaine de registres à balayage. La Figure 2.6 présente le chronogramme approprié du testeur à concevoir.

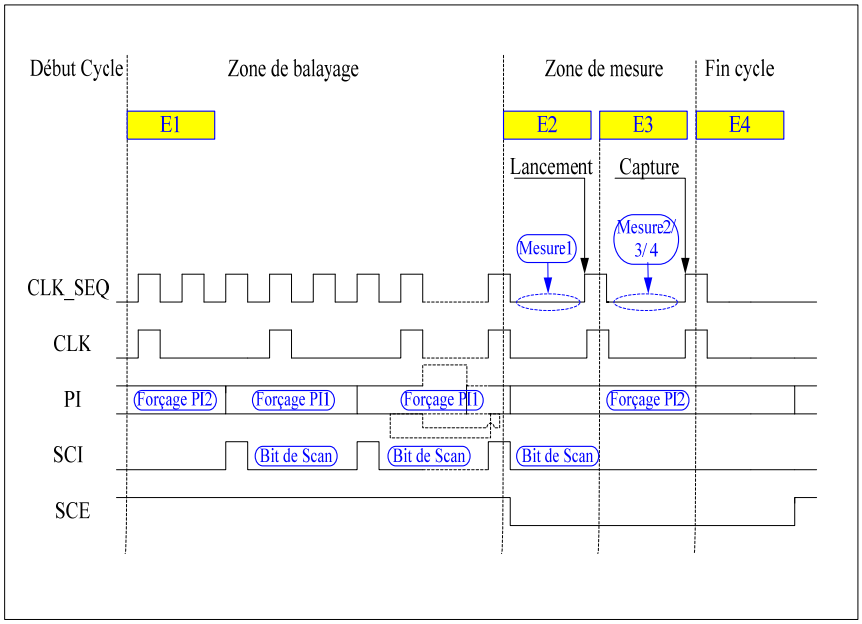

Figure 2.6 Chronogramme de fonctionnement de testeur $C\Delta I_{DDQ}$

On y retrouve la spécification fonctionnelle de signal séquenceur (CLK_SEQ) et des signaux qui attaquent le circuit sous test (CLK, PI, SCI, SCE). E1 (respectivement E2, E3 et E4) présente le premier (respectivement deuxième, troisième et quatrième) événement. PI1 et PI2 présentent les valeurs à affecter aux entrées primaires PI pendant de chaque forçage. Les zones de mesures sont caractérisées par un temps très lent par rapport à fréquence de la zone de balayage. Dans ces zones, le testeur prélève les mesures des sorties primaires issues du CUT. Dans notre cas, il s'agit de mesures de courant IDDQ. L'approche consiste alors à mesurer la consommation du courant statique du CUT dans un intervalle de temps

relativement long. Reste à traduire ce chronogramme au niveau RTL pour développer le code approprié.

2.4.2 Modélisation du testeur

Après la définition des fonctionnalités attendues du testeur, la tâche suivante consiste à modéliser en RTL le système complet. Cette modélisation permet de choisir les éléments qui valident en termes de performance les besoins de l'application. Généralement, les éléments qui constituent les modèles RTL sont à base de logique séquentielle et combinatoire. Chaque élément permet de générer ou de manipuler les signaux qui définissent le comportement de circuit. On distingue deux types des signaux :

- les signaux de commande : il existe trois signaux de commande principaux pour le testeur : un signal interne d'horloge (CLK_SEQ) et deux signaux externes (l'horloge de test (CLK) et le maître de test (SCE)). Le premier signal contrôle la synchronisation des différentes séquences dans le circuit et permet aussi de générer les deux signaux externes. La génération de ces signaux se fait exclusivement dans l'unité de contrôle du testeur. Cette unité est constituée d'une machine à états finis synchrone (FSM). Par définition, la FSM balaie un ensemble d'états finis en fonction des entrées de transition et des états antérieurs. Pour chaque état, la FSM permet de générer le front CLK et le SCE approprié. La Figure 2.7 présente la FSM adéquate pour la génération du signal CLK. Dans la figure, cnt_nb_CLK désigne le compteur du nombre de fronts de CLK qui incrémente après chaque groupe de quatre fronts du séquenceur dans l'événement E1. Ainsi, chaque incrémentation de compteur signifie une pulsation de front CLK. La variable nb_reg_sc indique le nombre de bascules de la chaîne à balayage du CUT;

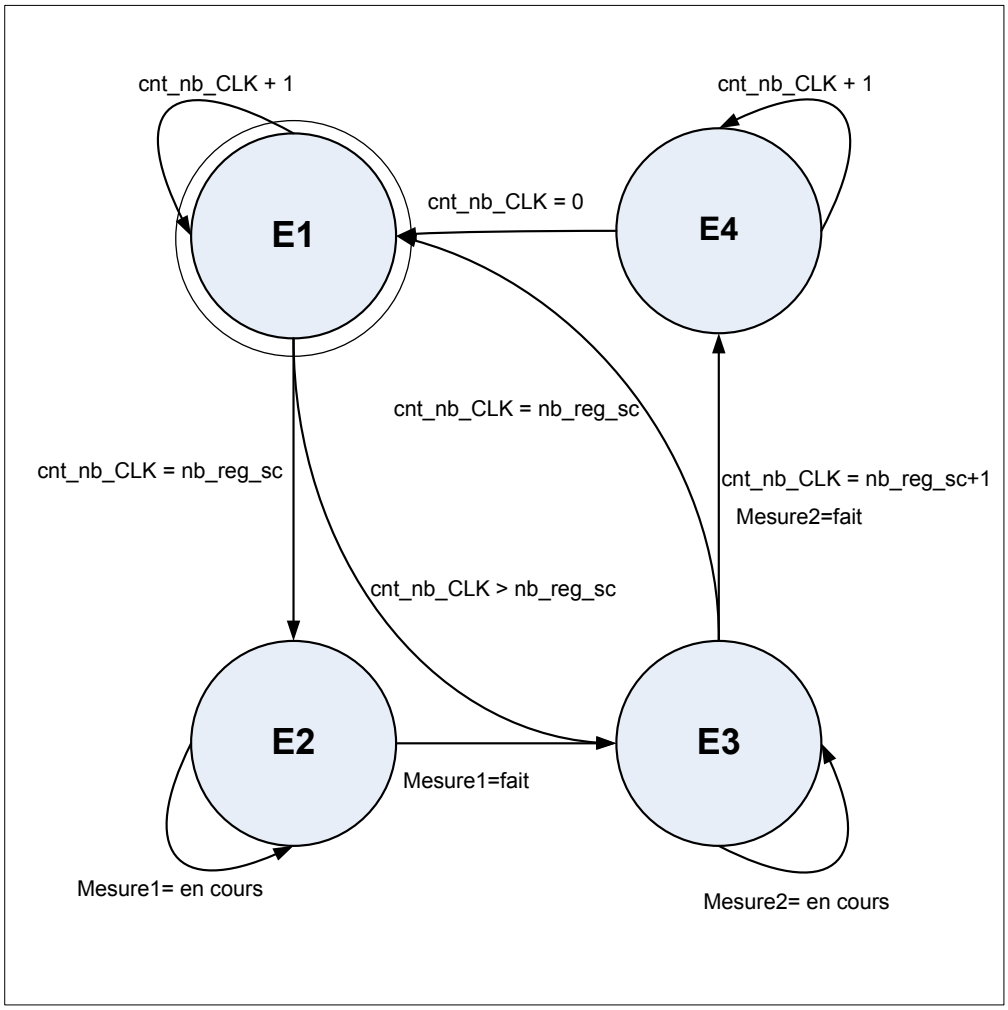

Figure 2.7 FSM de génération d'horloge de testeur $C\Delta I_{DDQ}$

- les signaux des données. Ce sont les signaux de forçage (PI et SCI) qui seront appliqués au CUT suite à une pulsation d'un front de CLK. Ces données seront extraites des fichiers de valeurs mentionnées précédemment et seront stockées dans des mémoires internes du circuit testeur. Vu que la taille de ces données est petite (moins de 16 K), la meilleure façon de classer ces valeurs est d'intégrer des mémoires vives RAM dans le FPGA. Ceci permet de traiter les données dans les mêmes domaines d'horloges utilisés et d'accélérer ainsi le processus de test. La Figure 2.8 présente le RTL de testeur complet incluant les mémoires SRAM intégrées.

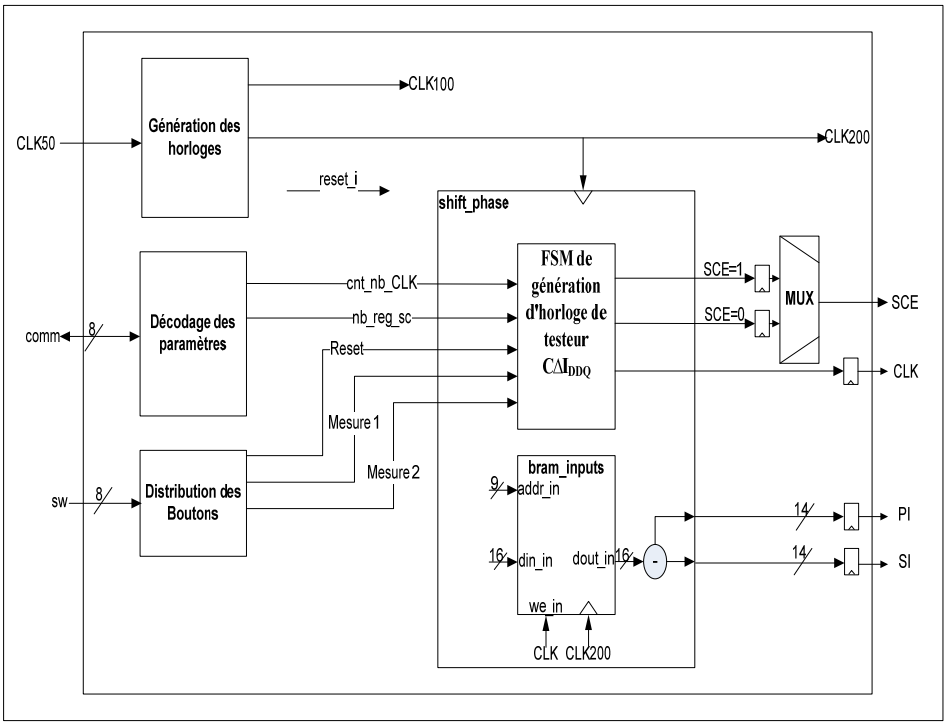

Figure 2.8 RTL du testeur $C\Delta I_{DDQ}$

2.5 Implémentation et validation expérimentale

2.5.1 Conception matérielle de la plateforme de test.

Dans le but de valider expérimentalement la détection des pannes de courts-circuits dans un ASIC émulé dans une puce FPGA, nous avons développé une plateforme de test FPGA à faible coût. Cette plateforme est constituée d'appareils et de cartes électroniques disponibles au laboratoire LACIME de l'ÉTS. Comme l'indique la figure 2.9, cette plateforme est formée essentiellement de :

- deux cartes électroniques à base de FPGA pour simuler un testeur et un CUT. L'isolation de testeur au CUT permet d'éliminer l'impact de variations électriques d'une carte à l'autre. Ainsi, la consommation électrique de la carte CUT reflète la consommation de son FPGA. Chaque carte doit posséder un port de communication pour l'interfaçage entre les deux cartes. La carte testeur doit contenir de la mémoire pour stocker les

vecteurs de test, et des boutons de commande. La carte sous test doit contenir seulement l'ASIC émulé par le FPGA. En raison des disponibilités, notre choix s'est porté sur deux cartes Nexys 2 de Digilent qui répondaient le mieux à nos besoins;

- une carte interface pour faire la liaison entre les deux cartes Nexys 2 : cette carte doit contenir des broches d'extension pour le branchement des résistances de courts-circuits et la vérification des vecteurs de test. La carte a été fabriquée au sein du laboratoire électrique de l'école;
- analyseur de puissance : il fournit l'alimentation électrique aux cartes et permet la visualisation en temps réel de la consommation de chaque carte. L'analyseur choisi est le N6705B DC Power Analyzer d'Agilent;
- analyseur logique : pour vérifier logiquement les valeurs des bits de test. Le type d'analyseur est Logicport d'Intronix. Il est connecté à la carte interface;
- micro ordinateur : pour configurer les cartes FPGA avec le logiciel Adept de Digilent et analyser les signaux émis par l'analyseur logique.

Dans cette plateforme, le processus de mesure de courant est toujours commandé par la carte testeur. C'est à partir de cette carte qu'on envoie les vecteurs de test à la carte sous test pour visualiser leurs effets. L'identification des valeurs de mesure se fait soit visuellement par une lecture simple sur l'analyseur logique, soit par l'enregistrement en temps réel d'une séquence de test automatique. Il suffit donc d'alimenter les deux cartes tout en respectant les domaines des tensions indiqués sur la figure2.9. Ensuite, la configuration de deux FPGA permet de lancer la séquence de test.

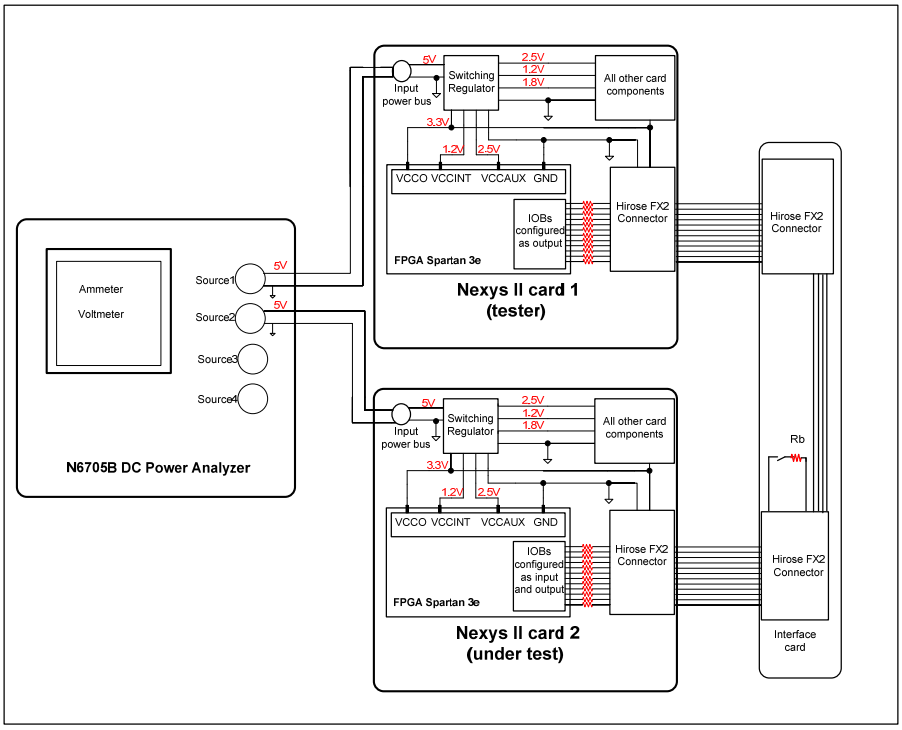

Figure 2.9 Plateforme de test des FPGA

2.5.2 Configuration de la plateforme aux applications spécifiques

Deux configurations sont nécessaires pour la mise en marche de la séquence de test. La première configuration concerne la carte du testeur. En s'appuyant sur la description du code RTL du testeur CDI_{DDQ} dans la figure 2.8, nous avons généré le code VHDL approprié pour l'application. Ce code contient évidemment les mémoires SRAM stockant les codes binaires extraits de fichier de valeurs déjà mentionnés. En fait, les codes binaires sont envoyés par le testeur à la carte sous test au fur et à mesure de la séquence de test. La deuxième configuration cible la carte sous test. Elle consiste à émuler un ASIC dans la puce FPGA pour le mettre sous test. Nous avons choisi le circuit S1196 de la famille des circuits séquentiels du benchmark ISCAS 89. Ce circuit est caractérisé par : 14 entrées, 14 sorties, 18 bascules D, 141 inverseurs, 388 portes (118 ET + 119 Non-ET + 101 OU + 50 Non-Ou).

Ce circuit a au départ été conçu comme un ASIC, incluant la transformation des bascules en chaîne de balayage dans ce circuit et la génération des vecteurs de test appropriés. Le

nouveau netlist a été par la suite converti pour une implémentation sur FPGA. Outre les entrées et sorties additionnelles dues à la chaîne de balayage (entrées : le scan-en et le scan-in; sorties : le scan out), 2 autres sorties ont été ajoutées, à savoir celles alimentées par les noeuds de court-circuit virtuel (M et N). L'insertion du court-circuit se fait en propageant les nœuds d'une capacité parasite vers une sortie observable.

2.5.3 Simulation logique

La première phase de vérification de la plateforme de test revient à simuler le comportement de nouveaux netlists de circuit S1196 face à un banc de test (*test bench*) $C\Delta I_{DDQ}$. Le simulateur Modelsim va simuler dans le temps chaque signal d'interconnections du netlist en fonction des séquences de stimuli décrites à la figure 2.6. En effet, la séquence de test est composée de plusieurs patrons, chaque patron contenant quatre vecteurs qui ciblent la détection d'un court-circuit dans une paire de nœuds bien précis. À titre d'exemple, nous présentons dans les figures ci-dessous (2.10, 2.11 et 2.12) les résultats de simulation de quatre vecteurs (Vw, Vx, Vy et Vz) de test $C\Delta I_{DDQ}$ suite à l'application du patron 17 arbitrairement choisis.

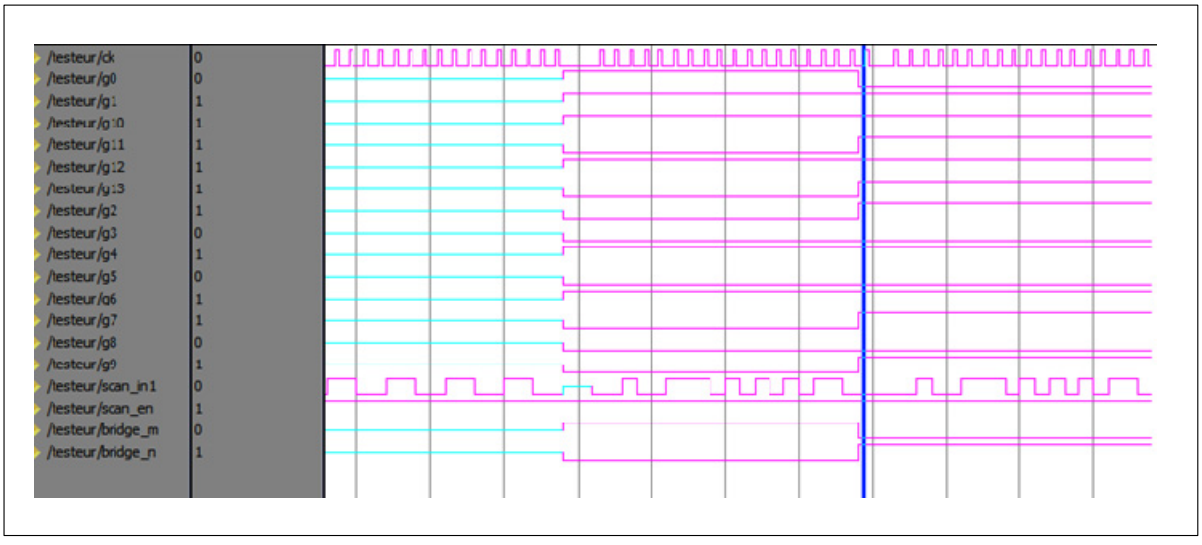

Figure 2.10 Simulation logique de Vw_Vx pour le patron 17

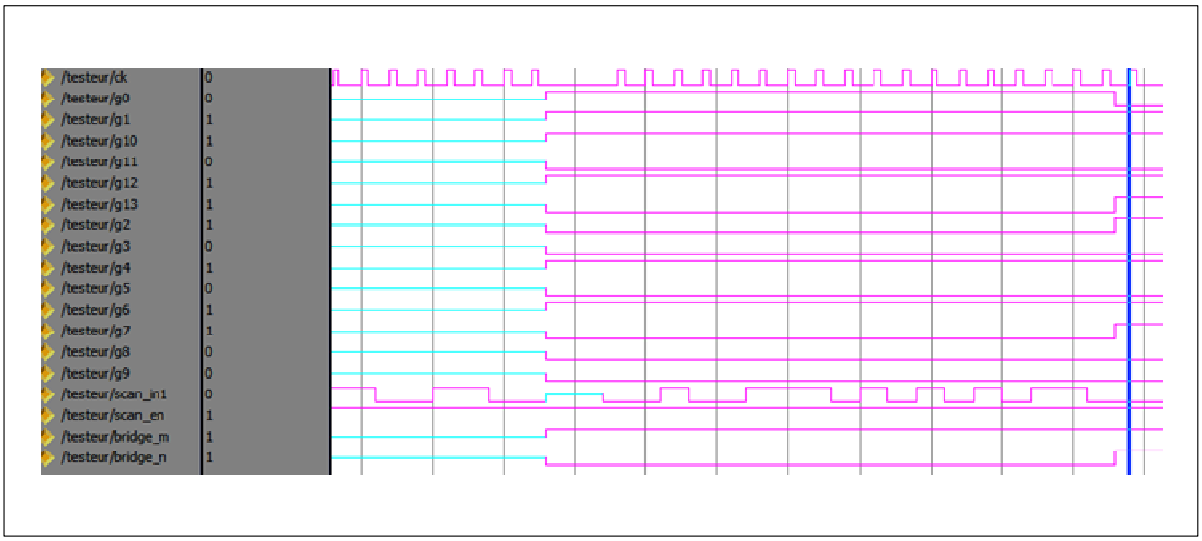

Figure 2.11 Simulation logique de Vy pour le patron 17

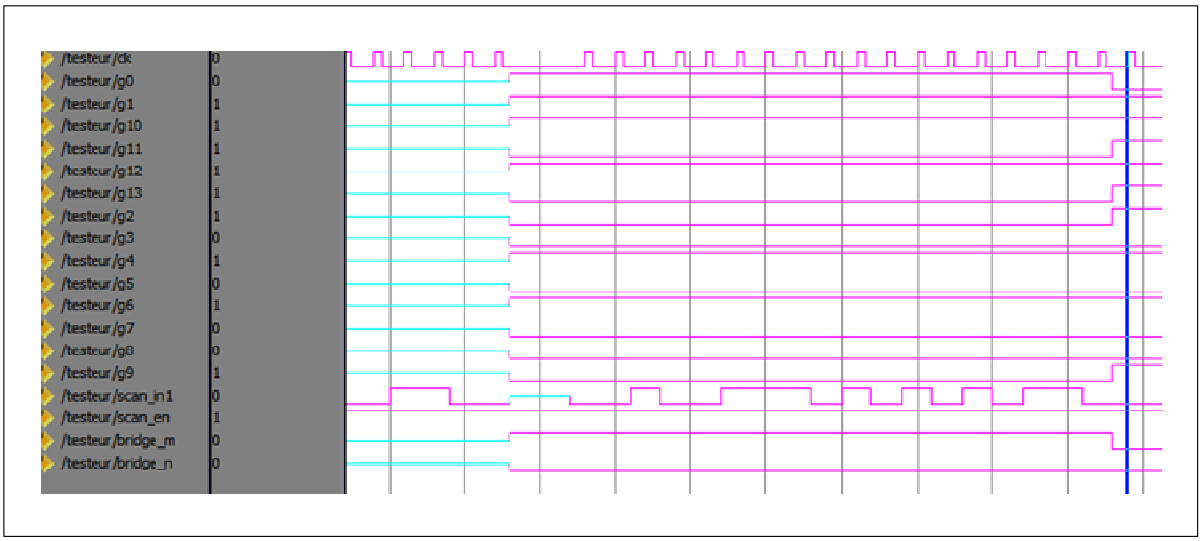

Figure 2.12 Simulation logique de Vz pour le patron 17

On distingue dans ces figures trois types de signaux. Le premier signal est l'horloge ck qui attaque les 18 bascules de la chaîne de balayage. Pour cela, on trouve dans les figures des séquences de 19 fronts d'horloge. Les dix-huit premiers fronts permettent le balayage des bits de scan et le dix-neuvième permet d'intégrer les bits des entrées primaires. Le deuxième type de signal correspond aux bits des entrées. Ces entrées sont composées des entrées primaires (g0 à g13), de l'entrée de scan (scan-in1) et de l'entrée d'activation (scan-en). Le dernier type

de signal correspond aux sorties auxiliaires des nœuds du court-circuit choisi. Théoriquement, suite à l'application de dernier front d'horloge de la séquence de test, un couple de nœuds de court-circuit suit les valeurs logiques suivantes (1010, 0110). Le Tableau 2.2 présente une récapitulation des valeurs logiques obtenues en appliquant le patron 17. Il est clair que ces valeurs respectent la séquence attendue, ainsi, nous pouvons valider logiquement notre testeur CDI_{DDQ}.

Tableau 2.2 Valeur logique des nœuds de court-circuit M et N

	Vw	**Vx**	**Vy**	**Vz**
nœud M	1	0	1	0
nœud N	0	1	1	0

2.5.4 Validation expérimentale

L'objectif de cette phase consiste à prendre les mesures du courant consommé par la puce FPGA pour chaque vecteur de test. Par la suite, la combinaison des courants obtenus permet de valider expérimentalement l'approche CDI_{DDQ}. Pour ce faire, nous avons programmé la plateforme pour lancer la séquence de test. Cette séquence s'arrête automatiquement au dernier front de chaque vecteur pour prélever les mesures en question de l'analyseur de puissance. Ces mesures sont présentées au tableau 2.3 pour le patron 17.

Dans ce tableau, deux cas se présentent : le premier cas présente les mesures prélevées en l'absence de court-circuit et le deuxième cas présente les mesurés prélevés avec l'insertion d'un court-circuit de plusieurs valeurs différentes, allant de 1k à 80kΩ dans la sortie observable du nœuds M et N. D'après les résultats obtenus (en lisant l'affichage du bloc d'alimentation et en notant pour chaque cas la valeur moyenne), il s'avère que les courts-circuits dont la résistance est inférieure ou égale à environ 80kΩ sont détectables, ce qui équivaut à un courant supplémentaire total ($C\Delta I_{DDQ} = I_{br,01} + I_{br,10}$) d'environ 70μA, alors que le pire cas recensé jusqu'ici sans défectuosité est $C\Delta I_{DDQ}$ = 30 μA. Nous pouvons confirmer à travers les résultats obtenus la validité de la technique $C\Delta I_{DDQ}$ pour tester les courts-circuits

des ASIC émulés dans les FPGA. Afin de s'assurer de la validation expérimentale, nous avons accompagné ces mesures d'un enregistrement automatique des séquences logiques appliquées dans la phase de test. Ces séquences ont été prises par l'analyseur logique. Comme le montrent les figures (2.13, 2.14 et 2.15), ces séquences correspondent exactement aux séquences prédites par le simulateur logique Modelsim.

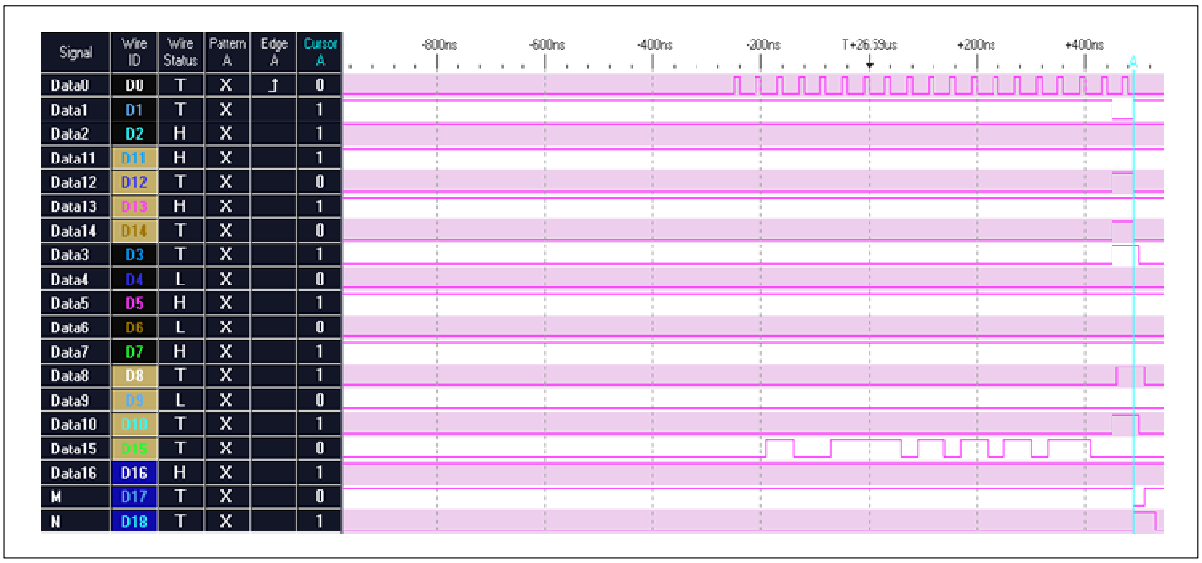

Figure 2.13 Séquence expérimentale de Vw_Vx_p17

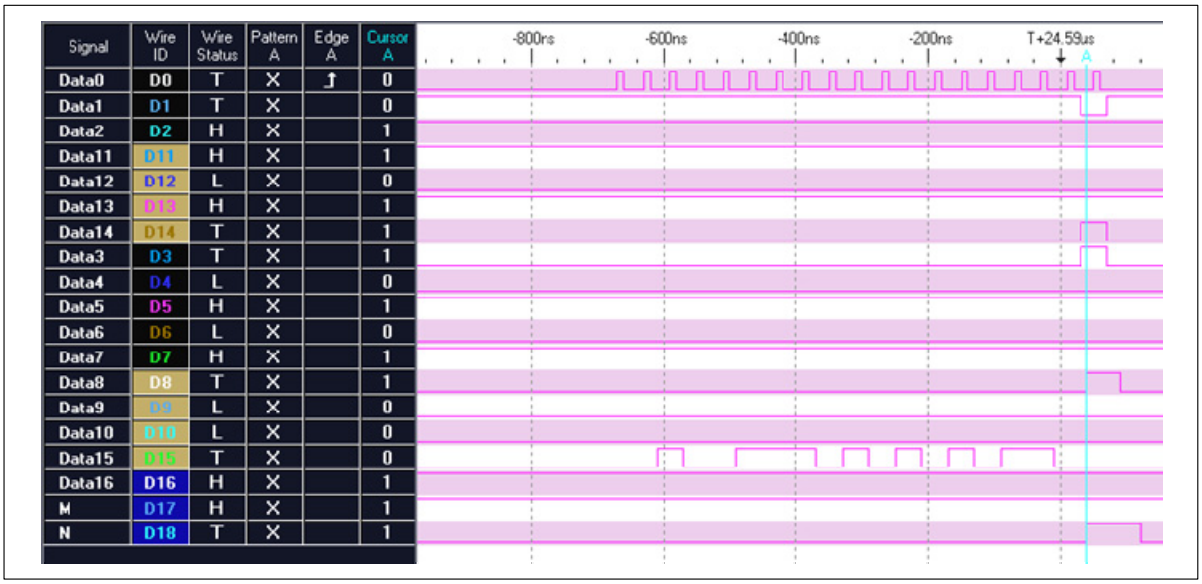

Figure 2.14 Séquence expérimentale de Vy_p17

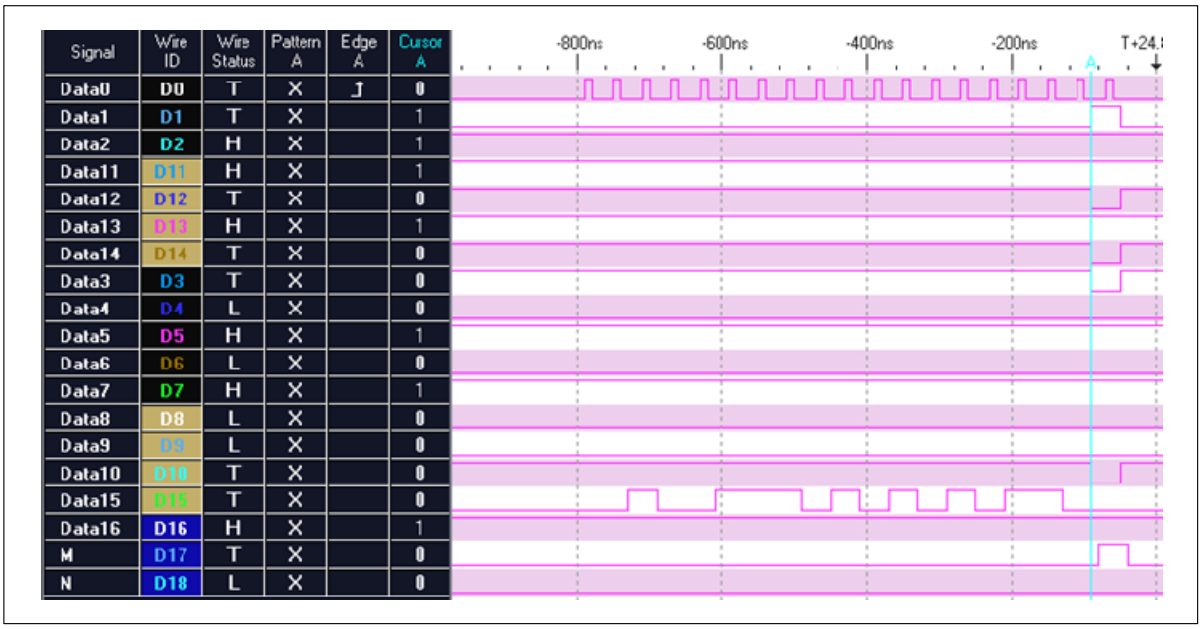

Figure 2.15 Séquence expérimentale de Vz_p17

Tableau 2.3 Vérification expérimentale de l'approche CDI_{DDQ} pour le patron 17

R (kohms)	**0**		**0,1**		**1**		**2**		**13**	
Unité	**I (mA)**	**V (V)**	**I (mA)**	**V (V)**	**I (mA)**	**V (V)**	**I (mA)**	**V (V)**	**I (mA)**	**V (V)**
Vw/0-0	80,14	0	80,14	0	80,15	0	80,14	0	80,15	0
Vx/1-1	80,15	0	80,17	0	80,17	0	80,18	0	80,17	0
Vy/1-0	80,15	3,272	84,34	0,728	84,11	1,236	83,73	1,705	80,24	2,795
Vz/0-1	80,16	-3,272	84,35	-0,728	84,12	-1,236	83,75	-1,705	80,25	-2,795
CDIDDQ	**0,02**		**8,38**		**7,91**		**7,16**		**0,17**	
R (kohms)	**17,8**		**39,03**		**71,2**		**80,2**		**84,2**	
Unité	**I (mA)**	**V (V)**	**I (mA)**	**V (V)**	**I (mA)**	**V (V)**	**I (mA)**	**I (mA)**	**V (V)**	**V (V)**
Vw/0-0	80,15	0	80,14	0	80,14	0	80,14	0	80,14	0
Vx/1-1	80,17	0	80,17	0	80,17	0	80,17	0	80,15	0
Vy/1-0	80,22	3,007	80,19	3,122	80,19	3,167	80,19	3,187	80,15	3,272
Vz/0-1	80,23	-3,007	80,2	-3,122	80,19	-3,167	80,19	-3,187	80,16	-3,272
CDIDDQ	**0,13**		**0,08**		**0,07**		**0,07**		**0,02**	

2.6 Conclusion

Dans ce chapitre, nous avons pu déterminer la description RTL du testeur CDI_{DDQ} approprié pour tester un ASIC émulé dans un FPGA. Autrement dit, nous avons adapté l'approche de test $C\Delta I_{DDQ}$ aux tests dépendants de l'application pour les FPGA de Xilinx. Pour ce faire, il a fallu que nous fassions une révision globale du processus de génération des vecteurs de test de la technique en question dans le domaine ASIC. Ainsi, à chaque étape de processus, nous avons exploité les différentes variables utilisées dans ce domaine et nous les avons adaptées au domaine cible. Une plateforme de test a été développée. Cette plateforme a permis la prise de mesures du courant en régime permanent, mesures qui ont validé expérimentalement l'approche de test CDI_{DDQ}.

CHAPITRE 3

STRATÉGIE DE CONFIGURATION POUR TESTER LES LIGNES D'INTERCONNECTIONS DES FPGAs AVEC CDI_{DDQ}

3.1 Introduction

Dans ce chapitre, nous nous intéressons au test des ressources de routage globales dans le FPGA. Nous mettrons plus précisément l'accent sur les pannes de courts-circuits qui affectent une partie de ces ressources. Suite aux résultats obtenus montrant l'efficacité de la méthode de test CDI_{DDQ} pour la détection de ce type de pannes dans un ASIC émulé dans un FPGA (Chapitre 2), nous proposons de généraliser l'application de la technique au test de différentes ressources du FPGA indépendamment des applications. Le travail consiste à proposer une stratégie de configuration pour mettre sous test certaines interconnections, à savoir les lignes horizontales doubles nord du FPGA Spartan 3E de la compagnie Xilinx. Dans ce cadre, nous mettons l'emphase sur l'aspect expérimental de la technique en donnant les démarches à suivre en premier lieu. Par la suite, nous définissons une nouvelle structure de configuration qui consiste à mettre les lignes ciblées sous la forme de peignes entrelacés. Nous proposons notamment un outil de configuration automatique de la structure pour ces lignes.

3.2 Mise en contexte

De point de vue du test des FPGA indépendant de l'application, la détection d'une panne suppose en général de placer tout d'abord le circuit dans un état précis (ou configuration) permettant d'activer la panne et la propager vers une ou des sorties observables le cas échéant. Ainsi, la réponse à l'application éventuelle d'un certain nombre de vecteurs aux entrées permet de détecter, pour une configuration donnée, un certain nombre de pannes parmi toutes celles pouvant affecter les ressources ciblées d'un FPGA. Par conséquent, l'atteinte d'une couverture de pannes acceptable peut nécessiter la reprogrammation (à plusieurs reprises) du FPGA et l'application d'autres vecteurs. Le tout se traduit par une

augmentation du temps de test. Ceci cause un vrai défi pour avoir un FPGA suffisamment testé. L'objectif est donc de minimiser le nombre de configurations et de vecteurs pour garder le temps de test à un niveau acceptable, tout en atteignant un niveau acceptable de couverture.

Pour atteindre cet objectif, nous avons adapté l'approche de test CDI_{DDQ} pour le test des lignes adjacentes. Le test envisagé des lignes horizontales ciblées constitue une preuve de concept du test des FPGA avec l'approche CDI_{DDQ}. Notons que nous posons les mêmes hypothèses de base pour cette approche, à savoir qu'il n'y a pas de changement de l'environnement (i.e. tension et température), ce qui signifie que le courant mesuré n'est pas affecté par de tels changements. Nous supposons également l'absence de ligne située en aval du site de transition qui reçoit les transitions extérieures. Le concept de décomposition d'une panne de transition dans le contexte CDI_{DDQ} est illustré à la Figure 3.1. De par la nature de la structure à tester, la décomposition est beaucoup plus simple que pour les ASIC. Les nœuds A et B représentent deux fils adjacents, la résistance (Rb) un site potentiel de panne de type court-circuit, et le quadruplet (Vw, Vx, Vy, Vz) les quatre vecteurs à appliquer. Vw et Vx constituent la première paire de vecteurs TDF générant une transition montant 0 -> 1 sur les 2 nœuds adjacents, alors que Vy et Vz sont les deux vecteurs transitoires complémentaires résultant de la décomposition : Vy est tel que seule une transition montante sur B serait créée si Vy était appliqué après Vw tandis que Vz est tel que seule une transition montante sur A serait créée si Vz était appliqué après Vw. Si les conditions précédentes sont remplies, il peut être démontré que, sans défectuosité, la combinaison CDI_{DDQ} du courant de fuite de ces quatre mesures combinées, $C\Delta I_{DDQ}$ (w, x, y, z), est égal à 0 (en théorie). L'équation de test $C\Delta I_{DDQ}$ peut être exprimée comme suit

$$C\Delta IDDQ(w,x,y,z) = I(V_Y) + I(V_Z) - I(V_W) - I(V_X) = 0 \tag{3.1}$$

Où I (V_i) est le courant de fuite mesuré suite à l'application du vecteur V_i.

Lorsque présent, le court-circuit est activé quand A ≠ B. Dans cet exemple, l'activation a lieu aux vecteurs Vy et Vz. L'impact du court-circuit sur la combinaison CDIDDQ (w, x, y, z) devient donc,

$$\left|C\Delta IDDQ(w,x,y,z)\right| = I_{br,01} + I_{br,10} \tag{3.2}$$

Le terme $I_{br,ab}$ représente la consommation de courant tiré par le court-circuit.

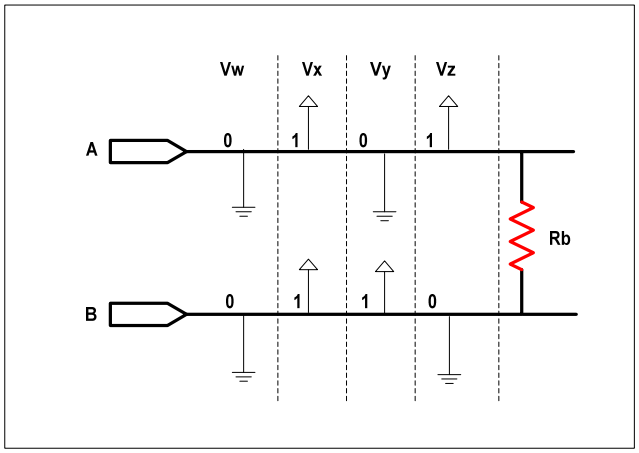

Figure 3.1 Détection d'un court-circuit dans une paire de lignes

Pour tester les interconnections ciblées, nous supposons également que :

- les interconnections sont constituées de *n* fils parallèles, où *n* est un entier positif;
- un seul court-circuit peut affecter à la fois l'ensemble des fils ciblés;
- un court-circuit se présente seulement entre deux lignes adjacentes.

Pour garder le même nombre de mesures pour la détection de courts-circuits dans un regroupement de *n* fils parallèles, nous proposons la solution présentée à la Figure 3.2. Elle consiste à diviser les *n* fils en deux groupes, un groupe G1 pour les fils impairs et l'autre G2 pour les fils pairs. Finalement, la séquence de vecteurs utilisés pour le fil A de la Fig. 3.1 (0101) est associée aux fils impairs (G1) tandis que celle des vecteurs utilisés pour le fil B de la Fig. 3.1 (0110) est associée aux fils pairs (G2). De cette façon, nous confirmons que nous pouvons détecter un court-circuit dans un type bien défini d'interconnection avec seulement 4 mesures $C\Delta I_{DDQ}$.

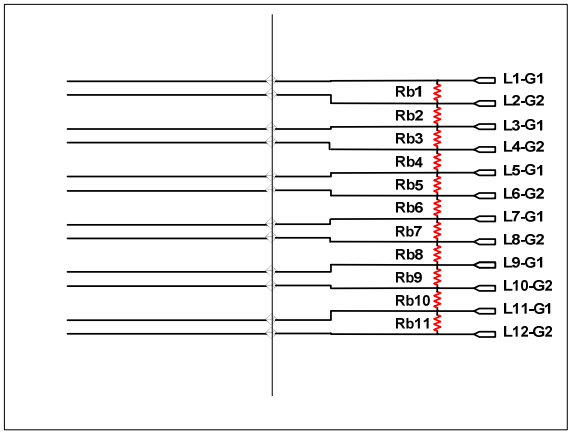

Figure 3.2 Division des lignes en deux groupes G1 et G2

3.3 Structure de peigne

Il est indispensable d'avoir des configurations adéquates pour satisfaire les hypothèses posées pour l'adaptation de l'approche CDI_{DDQ} au test des interconnections. Le rôle de chaque configuration est de créer un environnement propice pour un ensemble de fils constituant les interconnections sous test (*wire under test*, WUT). Puisque la méthode $C\Delta I_{DDQ}$ ne souffre pas des limitations imposées par l'observabilité, le seul point de contrôle de WUT est l'entrée de ces fils. Ceci augmente le nombre des WUTs qui peuvent être testés en parallèle et réduit alors le nombre de configurations nécessaire pour avoir une couverture de pannes la plus complète possible. Cependant, le rapport du nombre de fils à tester au nombre d'E/S des FPGAs contemporains est très important. Donc, le nombre de configurations, qui constitue un facteur important dans le temps de test, pourrait demeurer significatif. La solution préconisée consiste à concaténer les WUTs pour en former des chaînes traversant une grande partie du FPGA. Dans notre revue de littérature (chapitre 1), nous avons recensé deux structures qui, à première vue, avaient le potentiel d'être adaptées à l'approche $C\Delta I_{DDQ}$. La première structure, en serpentins, a été développée par (Renovell, Portal et al. 1998) pour la première génération de Spartan, et est illustrée à la Figure 3.3. Toutefois, l'adaptation de cette structure présente plusieurs limitations à cause de changement structurel des matrices d'interconnection dans les nouvelles générations de FPGA (les anciennes SM sont constituées avec des points d'interconnections horizontaux et

verticaux alors que les nouvelles SM contiennent des points d'interconnections diagonaux) et pose ainsi beaucoup de défis.

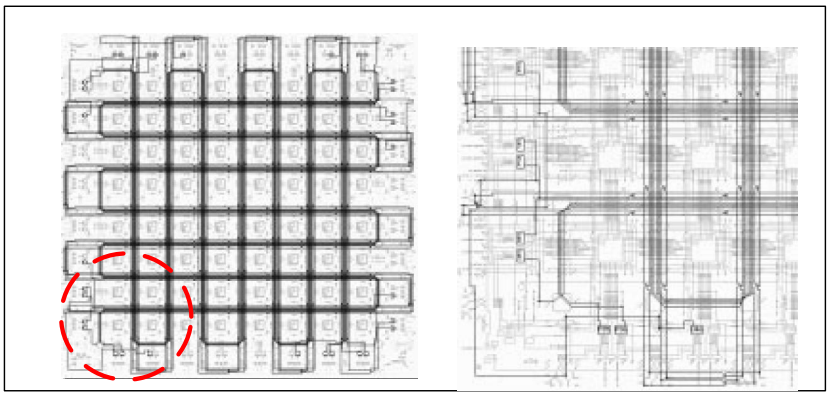

Figure 3.3 Structure de configuration en serpentins
Tirée de MICHEL RENOVELL et al. (1998, p. 49)

Le concept des bus globaux orthogonaux, verticaux et horizontaux développé par (Tahoori and Mitra 2005) est l'autre structure d'intérêt. Le passage par les registres est indispensable pour concaténer les chaines des bus d'interconnections, comme le présente la figure 3.4. La présence de ces registres rend le test séquentiel (alors que CDI_{DDQ} est une technique purement combinatoire), ce qui augmenterait de manière significative le nombre de vecteurs à appliquer pour conserver la propriété de combinaison résultante nulle des courants en régime permanent. Le nombre de mesures I_{DDQ} à prendre augmenterait de manière significative le temps de test, sachant qu'il faut environ 20 ms par mesure I_{DDQ} d'après (Chmelar and Toutounchi 2004).

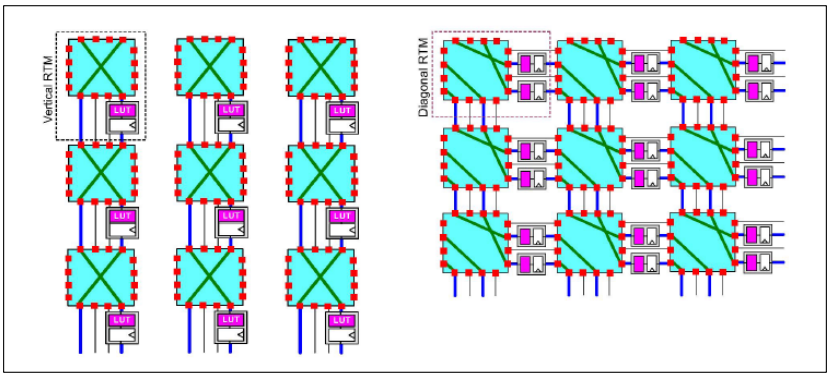

Figure 3.4 Concept des bus globaux
Tirée de Tahoori et Mitra (2005, p. 5)

Dans ce cadre, et en nous basant sur la régularité du FPGA, nous avons opté pour une structure bidimensionnelle pour minimiser de façon efficace le nombre de configurations et de mesures. La structure proposée consiste à concaténer les WUT selon une structure en peignes entrelacés, comme l'indique la Figure 3.5. Les couples de peignes entrelacés visent à tester un ensemble des WUT de même type et direction dans toutes les rangées de la puce. Dans chaque rangée, les deux doigts entrelacés du couple de peignes ciblent deux lignes adjacentes de même type (dans notre cas, *DOUBLE*). De cette façon, on aura deux groupes de couples de peignes inter-digités en parallèle pour couvrir la totalité d'un ensemble de WUT. La jonction des doigts de chaque rangée se fait par le dos du peigne. Il s'agit de doigts extrêmes droit ou gauche de peignes de directions opposées.

Le routage des lignes de dos de peigne garantit une grande souplesse pour accéder aux doigts. Elle permet aussi d'étendre la couverture des lignes cibles aux quatre côtés du FPGA, tout en ayant la même longueur de fil à tester. Théoriquement, deux configurations sont suffisantes pour avoir une couverture totale des lignes d'interconnections. La première configuration utilise les lignes verticales pour former les doigts et les ressources horizontales pour former le dos et inversement pour la deuxième configuration. En outre, cette structure évite le passage par les BLC et permet ainsi d'obtenir un temps de test minimum pour la technique de test $C\Delta I_{DDQ}$.

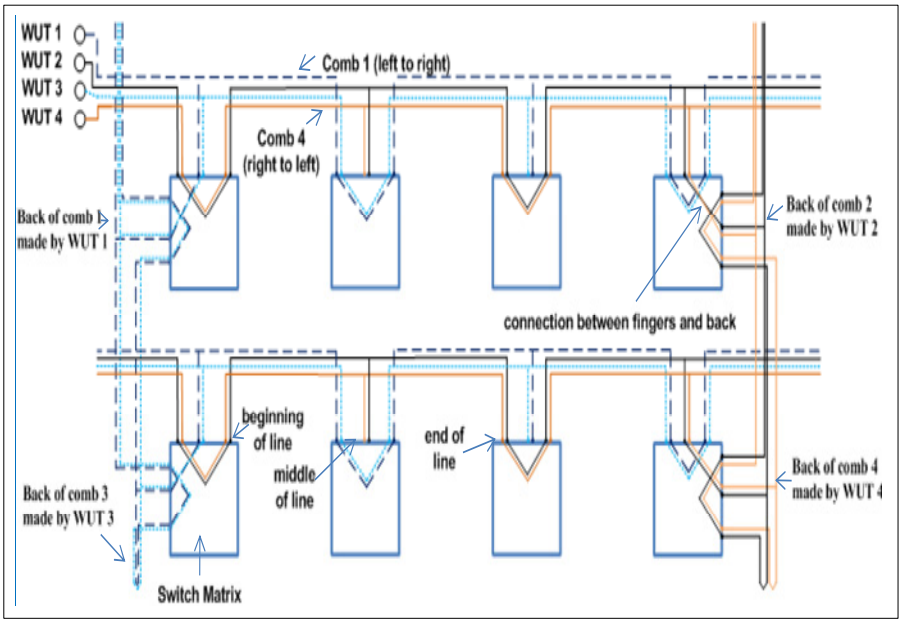

Figure 3.5 Structure des peignes entrelacés

3.4 Algorithme d'auto-routage

3.4.1 Stratégie de routage

Après avoir défini la structure des configurations, l'étape suivante consiste à définir la stratégie adéquate pour l'implémenter dans la puce. Pour ce faire, il est absolument indispensable de modifier la configuration des ressources du FPGA au niveau le plus bas permis par les outils de conception, à savoir le routage direct par l'usager. Au meilleur de nos connaissances, il y a deux méthodes qui offrent la possibilité de modifier le FPGA à bas niveau: le traitement du fichier XDL (*Xilinx Design Language*) et l'outil FPGA Editor. Le fichier XDL donne une description physique du réseau de routage et de la structure logique. C'est un langage puissant écrit en format humainement lisible (ASCII) permettant ainsi au concepteur de manipuler le routage des différentes ressources du FPGA. Cependant, ce langage souffre de l'absence de documentation à part quelques informations rudimentaires dans (Lavin, Padilla et al. 2010) et (Xilinx 2009). Nous nous sommes donc tournées vers la configuration des circuits FPGA via l'outil FPGA Editor. C'est un outil fourni par Xilinx qui

permet de visualiser les composants du circuit FPGA et d'apporter des modifications au placement et au routage. En fait, le FPGA, comme on l'a vu précédemment, est une matrice de composants et d'interconnections programmables. FPGA Editor permet de voir exactement lesquels de ces composants sont en cours d'utilisation, la façon dont ils sont utilisés, et comment ils ont été connectés. FPGA Editor prend un fichier **NCD** formaté comme entrée. Plus précisément, le fichier **NCD** représente un circuit qui est associé à des ressources et des composants spécifiques. Ce qui est important dans notre étude est que FPGA Editor permet de modifier le fichier **NCD** en fonction des configurations qu'on décide de réaliser. Cette modification peut se faire manuellement, soit par la souris, soit par l'écriture des commandes appropriées. Cependant, FPGA Editor offre la possibilité de faire la modification automatiquement en exécutant les commandes à partir d'un script **SCR** humainement lisible.

Vu la documentation détaillée de FPGA Editor (Xilinx 2009), ainsi que la flexibilité offerte par cet outil, nous avons choisi de l'utiliser pour aboutir à la configuration désirée. De ce fait, nous devons générer un circuit constitué uniquement des lignes d'interconnections, ce qui signifie l'utilisation de trois types de ressources du FPGA :

- les IOBs : l'outil permet de définir la position, le placement et le routage des blocs IOBs à utiliser avec la commande : select pin IOB 'position';
- les lignes d'interconnection : l'outil distingue les différents types de lignes (SIMPLE, DOUBLE,…) et leurs coordonnées dans la puce (x, y), où x et y sont des entiers naturels allant de -100000 à 100000. La commande de sélection pour le placement et le routage est la suivante : select ligne 'type de ligne (x, y) ';
- les points d'interconnections programmables (PIPs) : FPGA Editor considère que ces points d'interconnections sont des arcs de connection entre des lignes de départ et des lignes d'arrivée. Le symbole ---> indique le sens de l'arc. Ainsi, la commande menant à la programmation d'un tel PIP est : select arc 'type de ligne de départ (x, y) ---> type de ligne d'arrivée (x', y') '.

Ci-dessous apparaît une partie de script pour le routage automatique de deux lignes adjacentes.

```
/*debut script*/
select pin 'B4.I'
select wire 'IOBOUTPUT(-49896,80904)'
select wire 'DUMMYESC(-50055,80904)'
select arc 'IOBOUTPUT(-49896,80904)--->DUMMYESC(-50055,80904)'
select wire 'DOUBLE(-62515,82072)'
select arc 'DOUBLE(-62515,82072)--->DOUBLE(-61416,81975)'
select wire 'DOUBLE(-61416,81975)'
route
/*fin script.*/
```

La Figure 3.6 présente la configuration obtenue après l'exécution du script pour le routage de deux lignes adjacentes.

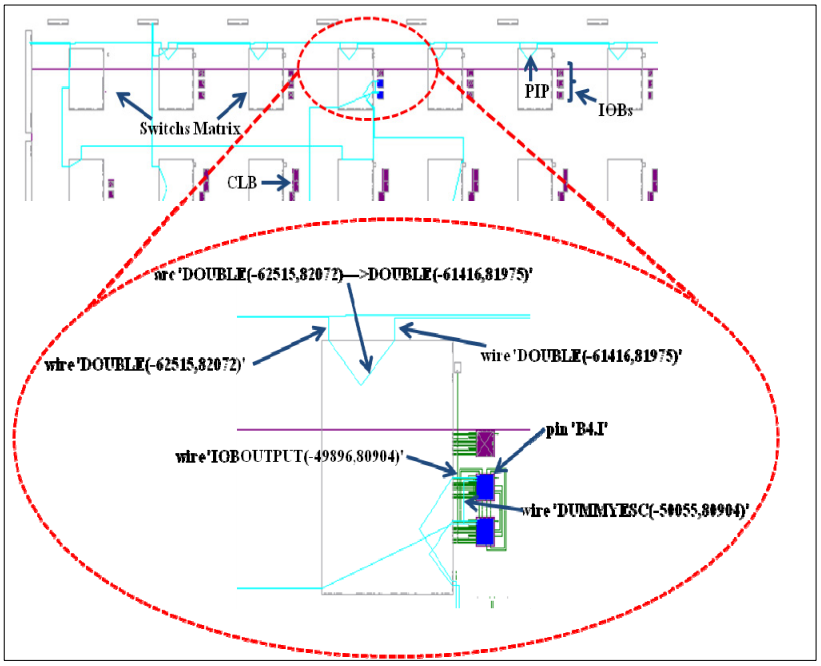

Figure 3.6 Routage automatique de deux lignes adjacentes

En conservant la syntaxe du script et en changeant seulement les coordonnées (x, y) des ressources utilisées par d'autres coordonnés (x', y') valides, on obtient une configuration de deux autres lignes adjacentes, parallèles aux deux lignes originales et qui appartiennent toujours à la même famille d'interconnections. Cette famille est caractérisée par le type des interconnections (par exemple *SIMPLE, DOUBLE, HEX, LONG*) et la direction (verticale, horizontale). Ceci implique que la génération d'un script exécutable pour configurer des peignes entrelacés d'une famille donnée de ressources de routage nécessite seulement la génération d'un script de base pour un seul couple de peignes. Par la suite, un simple changement des coordonnées nous permet d'obtenir d'autres couples. Le même raisonnement nous permet de supposer qu'il suffit de générer le script des deux premiers doigts du couple de peignes pour obtenir le script des autres doigts. Le choix de lignes adéquates pour former le doigt de chaque peigne du couple dépend de la position du peigne. Comme présenté à la figure 3.5, s'il s'agissait du peigne à droite, il faut que le sens du routage soit de la droite vers la gauche. Donc, il faut vérifier l'existence physique de l'arc de liaison dans ce sens. Dans le cas contraire, on affecte ce doigt au peigne gauche et vice versa. Reste alors la configuration de dos de peigne, et faire la liaison avec le doigt via le milieu de la première ligne qui croise le dos.

Le pseudo-code de l'Algorithme 3.1 présente le processus de génération de scripts pour l'obtention de la configuration désirée. En fait, cet algorithme s'appuie sur l'hypothèse de la présence de toutes les informations physiques des ressources à utiliser, à savoir le nombre de rangées et de colonnes, le nombre de lignes par rangée et leurs adresses (x, y). L'exécution du code au complet dans l'environnement C nous permet d'avoir les scripts de commande nécessaires pour la configuration ciblée (Voir ANNEXE V pour les scripts générés).

```
while (all T_line in line_i != visited ){
     create first T_line of i_comb;
     T_line = {simple, double, hex, long};
     line_i = {vertical_line, horizontal_line};
     while (all T_line in first line_i!= visited ){
             select wire 'T_line(x(i),y(i))' ;
             select arc 'T_line(x(i),y(i))--->T_line(x(i+1),y(i+1))';
             select wire 'T_line(x(i+1),y(i+1))' ;
             route;
             d_direction = 'left_to_right'; i++;
             if ( arc 'T_lyne(x(i),y(i))--->T_line(x(i+1),y(i+1))'
                    not exist){
                    select wire 'T_line(x(n),y(n))' ;
                    select arc 'T_line(x(n),y(n))--->T_line(x(n-1),y(n-1))'
                    select wire 'T_line(x(n-1),y(n-1))' ;
                    route;
                    d_direction = 'right_to_left'; n--;
             }
             mark first finger of i_comb routed in d_direction;
       }//Routing_Algo_1
       switch (d_direction){
             create back of first comb;
             create links between back and other fingers in first comb;
             while ( j row != visited ){
                    create remains fingers of i_comb;// using Routing_Algo_1 and
                    //only changing the coordinates of used resources (x,y) by (x',y');
                    j++;
             }
     }
       mark used T_line visited; i_comb++;
}
```

Algorithme 3.1 Auto-routage de Spartan3e

3.4.2 Générateur des informations physiques

Pour mettre en œuvre l'algorithme développé pour la configuration, il est nécessaire de lui fournir les informations physiques nécessaires. Avoir accès à ces informations présente certains défis. Premièrement, le fichier qui contient ces informations est propriétaire à la compagnie Xilinx et n'est pas accessible. Deuxièmement, ces informations varient d'un FPGA à l'autre, même à l'intérieur d'une même famille. Face à cette situation, nous avons opté pour le développement d'un outil pour générer automatiquement les adresses, nécessitant au départ une investigation manuelle de la nomenclature de ces adresses et ce, pour une ressource de routage en particulier (des lignes d'interconnections doubles nord horizontales de circuit FPGA Spartan 3e de Xilinx). Le FPGA cible est composé de 48 rangées, où chaque rangée contient environ 400 lignes doubles. Il s'agit donc de générer les

adresses d'environ 20000 lignes doubles réparties dans le FPGA. Notons toutefois que la répartition de ces lignes n'est pas régulière (voir les axes et les zones vides à la Figure 3.7).

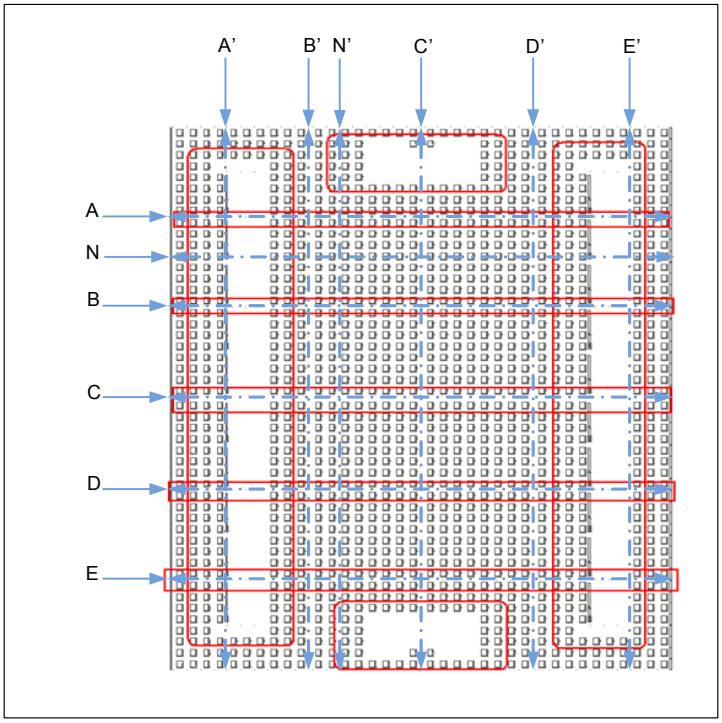

Figure 3.7 Axes de symétrie dans Spartan 3e

Cette non-régularité cause un défi supplémentaire pour l'automatisation de processus. Cette non-régularité présente par contre une certaine symétrie. Il existe en effet des axes de symétrie qui permettent de trouver plus facilement les adresses des ressources de connexions parallèles à ces axes.

On distingue deux types d'axes, montrés à la figure 3.7 :

- les axes réguliers, horizontaux (N) et verticaux (N'). tel que N et N' sont des entiers positifs représentant respectivement le rang d'une rangée horizontale et le rang d'une rangée verticale dans la puce : ces axes se trouvent entre toutes les rangées séparées par des distances uniformes. Ainsi, pour une ligne d'interconnection d'adresse (x, y) qui se trouve dans une rangée N, l'adresse de la ligne dans la rangé symétrique N+1 est $(x+\alpha, y+\beta)$, où α et β sont des constantes;

- les axes irréguliers, horizontaux (A, B, C, D et E) et verticaux (A', B', C', D' et E'): ces axes se trouvent entre les rangées séparées par des zones blanches et des distances irrégulières. Pour une ligne d'interconnection d'adresse (x, y) qui se trouve dans une rangée au dessus de l'un de ces axes, l'adresse de la ligne dans la rangée symétrique est (x+α(i), y+β(i)), où les valeurs α(i) et β(i) dépendent de l'axe ciblé.

Ces observations nous permettent de conclure que la génération des adresses de la première rangée de Spartan 3e permet de trouver les adresses de la majorité des rangées. L'adresse des rangées d'exception, causées par les zones vides et irrégulières, peuvent être déduites en examinant un nombre réduit de rangées (ici 7 rangées ont suffi). Ci-dessous est présenté le pseudo-code de l'algorithme qui résume le processus de génération des adresses.

```
create first row of regular double line database address;
create seven first row of irregular double line database address;
for (i=1; i <= N ; i++){
     while (all double line in i row != visited ){
      address double line number j = (x-a, y-ß);
        j++; (x,y) = (x-a, y-ß);
      if ( i=A||i=B||i=C||i=D||i=E){
            adress double line number j = (x-a(i), y-ß(i));
            j++; (x,y) = (x-a(i), y-ß(i));
        }
      if ( double line number j in i row is irregular){
            replace address double line number j = appropriate
            irregular address;
            j++;
        }
  }
  mark i row visited;
  j = 0;
}
```

Algorithme 3.2 Générateur des adresses physiques des interconnections

L'outil développé est un programme écrit en langage C. Une fois lancé, il permet de générer les adresses requises (voir ANNEXE VI pour les adresses générées). Ainsi, l'injection de ces adresses dans l'outil de configuration nous permet d'avoir la configuration requise. Cette configuration consiste à concaténer toutes les lignes horizontales doubles nord de la puce pour avoir un seul ensemble des lignes adjacentes comme la présente la figure 3.2. Reste

alors la vérification expérimentale de détection des pannes de courts-circuits entre les lignes couvertes dans le chapitre suivant.

3.5 Conclusion

Dans ce chapitre, nous avons présenté une stratégie d'adaptation de la technique de test CDIDDQ pour détecter les pannes de courts-circuits dans les interconnections d'un FPGA. Nous avons utilisé la structure en peignes entrelacés pour mettre les interconnections sous test. Nous avons présenté un algorithme simple pour configurer la couverture de toutes les pannes de courts-circuits affectant les interconnections adjacentes ciblées en deux configurations. En outre, nous avons introduit un outil générateur de scripts pour configurer les lignes horizontales doubles nord de Spartan 3e de la compagnie Xilinx. Ce générateur peut facilement être étendu pour terminer la configuration des lignes restantes afin de tester tout le réseau de routage et de travailler avec d'autres générations contemporaines de FPGA.

CHAPITRE 4

IMPLÉMENTATION ET RÉSULTATS EXPÉRIMENTAUX

4.1 Introduction

L'architecture matérielle utilisée au chapitre 2 offre la possibilité d'implémenter et de valider les techniques de test des puces FPGA. Dans ce chapitre, nous proposons l'utilisation de cette même architecture matérielle pour appliquer la technique de test CDI_{DDQ} indépendamment d'une application. Nous nous limiterons à l'implémentation et à la validation des approches développées dans le chapitre 3. Nous présentons notamment les résultats obtenus suite à l'application des approches en question sur la plateforme conçue. Nous présentons également l'outil de configuration automatique pour les lignes horizontales doubles nord de Spartan 3e de la compagnie Xilinx mentionné au chapitre 3.

4.2 Configuration de plateforme pour les applications indépendantes

Nous avons déjà présenté dans le troisième chapitre l'aspect théorique de l'approche de l'adaptation de la technique de test CDI_{DDQ} indépendant de l'application de l'FPGA. Afin de prouver expérimentalement l'efficacité de cette approche, nous avons conçu en premier lieu un outil de configuration automatique de la puce. En second lieu, nous avons appliqué la séquence de test appropriée à la configuration obtenue automatiquement.

4.2.1 Configuration de la puce FPGA

La configuration de la puce FPGA en forme des peignes entrelacés est générée à partir d'un script exécuté par FPGA Editor. Ce script est constitué d'une structure plus ou moins régulière de commandes permettant de couvrir toutes les lignes horizontales doubles nord de Spartan 3e afin de pouvoir les tester a posteriori. Nous avons développé un outil écrit en langage C pour satisfaire cette tâche. La Figure 4.1 présente l'interface DOS de cet outil. Cette interface nous permet de vérifier les différentes étapes de configuration.

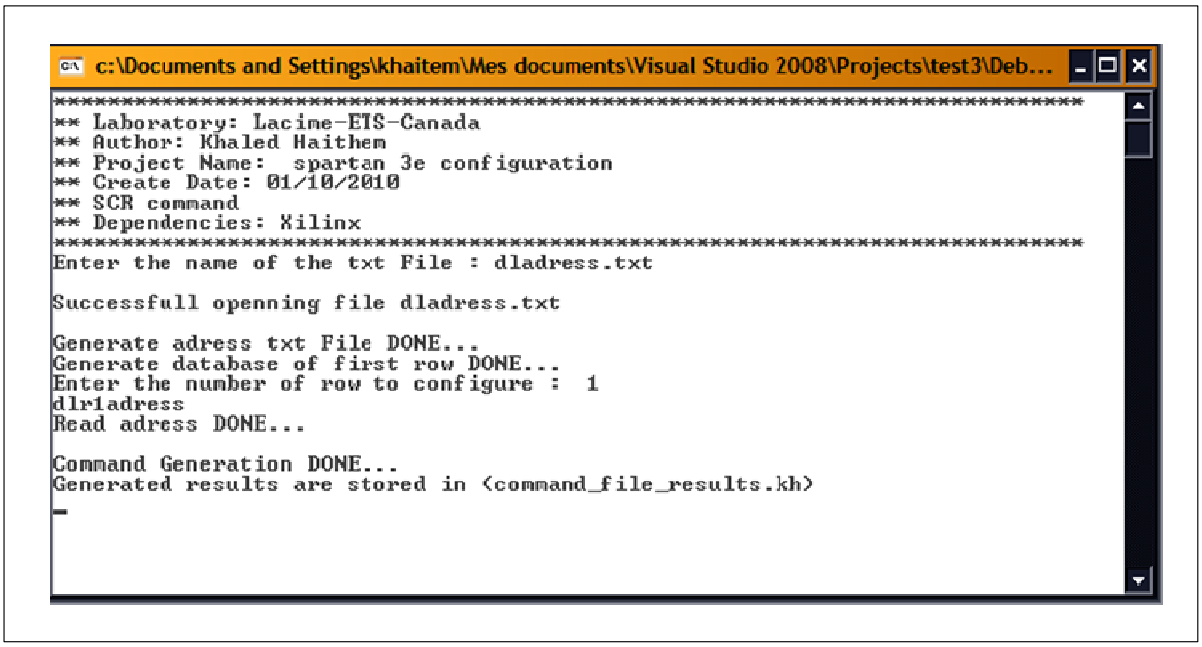

Figure 4.1 Interface de l'outil de configuration

Une fois lancé, l'outil de configuration permet en quatre étapes de générer les commandes nécessaires :

1- Extraction des adresses des lignes doubles à partir d'un fichier texte nommé dladress.txt généré en premier lieu avec FPGA EDITOR, décrit au chapitre 3 (exemple (-49896,80896))

2- Classement des adresses selon le type d'interconnection come la présente la figure 4.2 (voir ANNEXE VII pour les adresses obtenus).

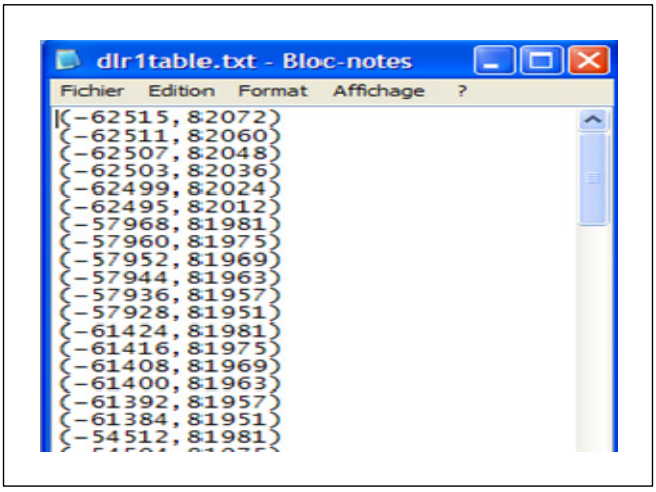

Figure 4.2 Liste des adresses des lignes doubles (extrait)

3- Choix de la rangée à configurer et génération de la base de données des lignes à configurer à partir de dlr1table.txt, et classement de cette base de données dans dlr%iadress tel que %i est le numéro de la rangée choisie.

4- Génération des commandes pour configurer les rangées choisies comme présenté dans la figure 4.3 (voir ANNEXE VIII pour les scripts générés).

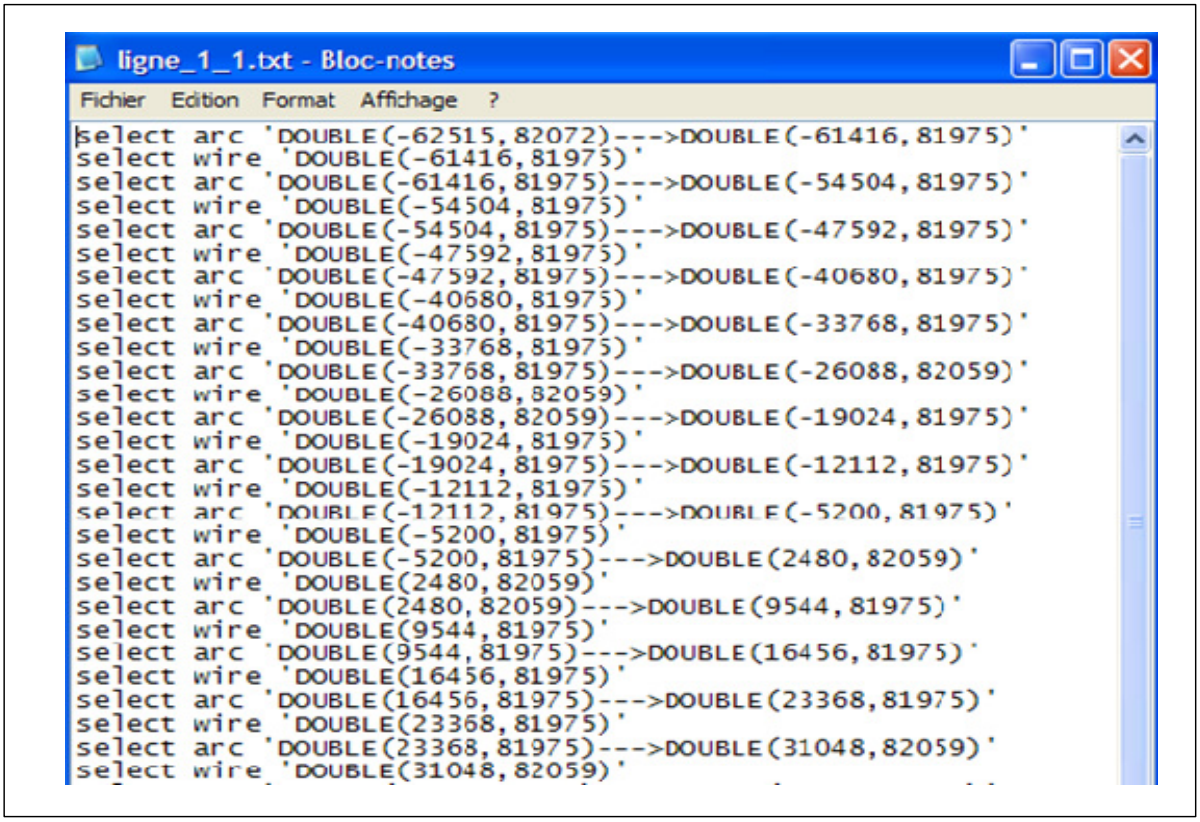

Figure 4.3 Script exécutable pour la configuration (extrait)

L'exécution de générateur de script de configuration permet de fournir 9000 lignes de commandes dans un fichier en format SCR. La lecture automatique de ce fichier par FPGA Editor de Xilinx permet d'avoir le fichier de configuration de peignes entrelacés sous le format **NCD**. La

Figure 4.4 présente l'implémentation de la configuration dans le Spartan 3e. À partir de FPGA Editor, on peut visualiser les lignes couvertes par la configuration. La vérification

automatique du taux de couverture se fait en comparant les adresses générées par l'outil d'adressage aux adresses générées dans les scripts de configuration. En plus de la vérification automatique du taux de couverture, on peut confirmer graphiquement que le taux de couverture des lignes horizontales doubles nord de Spartan 3e est de 100% pour une seule configuration. L'outil ISE de Xilinx permet de générer le fichier de configuration à partir du fichier .NCD. Il suffit donc de configurer la puce sous test par le fichier de configuration pour entamer l'opération de test.

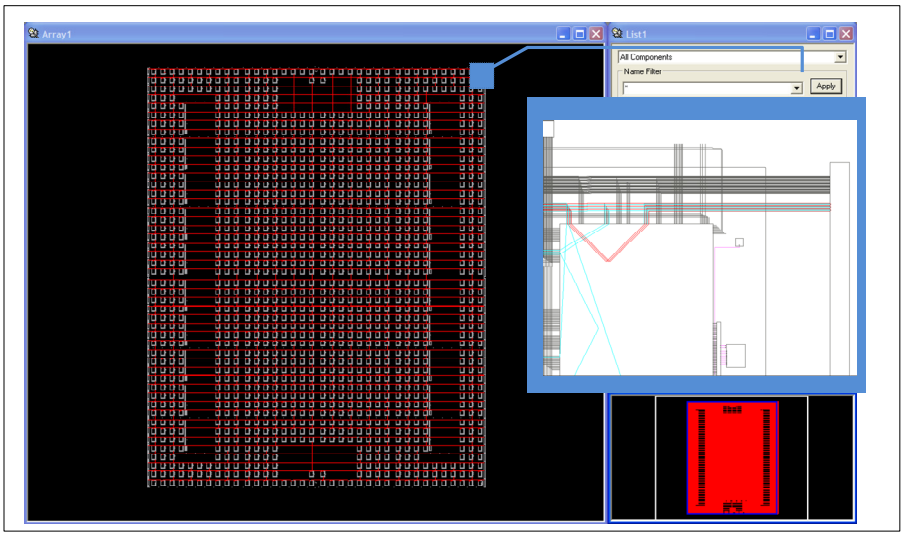

Figure 4.4 Visualisation de la configuration completée

4.2.2 Validation expérimentale

La validation expérimentale consiste à appliquer des vecteurs de test déterminés par la carte qui contient le testeur et à visualiser l'impact de ces vecteurs sur la puce sous test. Dans ce cas, les vecteurs visent à tester tous les courts-circuits qui peuvent affecter les 12 lignes horizontales nord de Spartan 3e (voir Figure 3.2). Tel que décrit au chapitre 3, nous utilisons pour les quatre vecteurs (Vw, Vx, Vy et Vz) les valeurs suivantes :

Vw= {L1,L2,L3,L4,L5,L6,L7,L8,L9,L10,L11,L12}={0,0,0,0,0,0,0,0,0,0,0,0}

Vx= {L1,L2,L3,L4,L5,L6,L7,L8,L9,L10,L11,L12}={1,1,1,1,1,1,1,1,1,1,1,1}

Vy= {L1,L2,L3,L4,L5,L6,L7,L8,L9,L10,L11,L12}={1,0,1,0,1,0,1,0,1,0,1,0}

Vz= {L1,L2,L3,L4,L5,L6,L7,L8,L9,L10,L11,L12}={0,1,0,1,0,1,0,1,0,1,0,1}

Notons que les courts-circuits sont émulés par des résistances placées à l'extérieur du FPGA sous test, plus précisément connectées, une à la fois, aux broches d'entrée des signaux alimentant chacune des paires des lignes adjacentes par un test particulier. Le Tableau 4.1 présente la valeur des courts-circuits injectés mesurés entre les lignes L1 et L2. La première ligne donne l'étiquette utilisée pour identifier l'expérience. La deuxième ligne présente la valeur de la résistance associée au court-circuit. Les quatre lignes suivantes présentent la valeur de mesure du courant consommé par la carte sous test suite à l'application des vecteurs identifiés par la première colonne. Et la dernière ligne présente le résultat de la combinaison CDI_{DDQ} des courants mesurés. Notons que la valeur des résistances a été augmentée afin de déterminer la limite de détection des courts-circuits. À partir de ces résultats, nous concluons que la technique CDI_{DDQ} permet la détection de courant supplémentaire de l'ordre de 0.070mA sur un courant total de plus de 86mA, sachant que la plus grande valeur de $C\Delta I_{DDQ}$ mesurée jusqu'ici sans court-circuit est de 0.030mA.

Une fois que la limite de détection est connue, nous avons fait les mesures avec la résistance de cette limite (7.9 k) pour toutes les positions de court-circuit du Tableau 4.2. La deuxième ligne de ce tableau présente la position de court-circuit ciblée. Sur la base de ces résultats, nous pouvons conclure que la technique est capable de détecter 100% des courts-circuits actifs dans les lignes doubles horizontales nord de Spartan 3e à l'intérieur de la limite de détection établie.

Tableau 4.1 Limite de détection de court-circuit dans les interconnections

Court-circuit	**sans Rb**	**Rb1**	**Rb 2**	**Rb 3**	**Rb 4**	**Rb 5**	**Rb 6**
Resistance (ohm)		**1k**	**2k**	**3,937k**	**6,91k**	**7,9k**	**7,98k**
Vw (mA) 0-0	80,14	80,15	80,15	80,14	80,14	80,14	80,14
Vx (mA) 1-1	80,15	80,17	80,17	80,17	80,17	80,17	80,17
Vy (mA) 1-0	80,15	80,24	80,21	80,19	80,19	80,19	80,19
Vz (mA) 0-1	80,16	80,25	80,22	80,2	80,19	80,19	80,19
$C\Delta I_{DDQ}$ (mA)	**0,02**	**0,17**	**0,11**	**0,08**	**0,07**	**0,07**	**0,07**

Tableau 4.2 Mesures appliquées à tous les positions du court-circuit

Court-circuit	**Sans Rb**	**Rb6**	**Rb 6**	**Rb 6**	**Rb 6**	**Rb6**	**Rb 6**	**Rb6**	**Rb 6**	**Rb6**	**Rb 6**	**Rb 6**
Lignes associés		**L1-L2**	**L2-L3**	**L3-L4**	**L4-L5**	**L5-L6**	**L6-L7**	**L7-L8**	**L8-L9**	**L9-L10**	**L10-L11**	**L11-L12**
Vw (mA) 0-0	80,14	80,14	80,14	80,14	80,14	80,14	80,14	80,14	80,14	80,14	80,14	80,14
Vx (mA) 1-1	80,14	80,17	80,17	80,17	80,17	80,17	80,17	80,17	80,17	80,17	80,17	80,17
Vy (mA) 1-0	80,15	80,19	80,19	80,19	80,19	80,2	80,21	80,2	80,19	80,19	80,19	80,19
Vz (mA) 0-1	80,15	80,19	80,19	80,19	80,2	80,2	80,21	80,2	80,19	80,2	80,19	80,19
$C\Delta I_{DDQ}$(mA)	**0,02**	**0,07**	**0,07**	**0,07**	**0,08**	**0,09**	**0,11**	**0,09**	**0,07**	**0,08**	**0,07**	**0,07**

4.3 Conclusion

Dans ce chapitre, nous avons utilisé la plateforme présentée précédemment pour appliquer la technique de test CDI_{DDQ} aux circuits FPGA. Nous avons prouvé que la technique est fonctionnelle pour le test indépendant de l'application. En outre, un outil générateur de script a été introduit pour les tests indépendants de l'application. Cet outil permet de configurer les lignes doubles horizontales nord de Spartan 3e de Xilinx. Ce générateur peut être facilement étendu pour terminer la configuration des lignes restantes afin de tester tout le réseau de routage, ainsi que pour son application avec d'autres générations contemporaines de FPGA. Les résultats obtenus donnent un taux de couverture de 100% pour les lignes ciblées pour une seule configuration et quatre mesures de courant.

CONCLUSION

Le fait que les FPGA partagent les mêmes technologies de fabrication que les ASIC a incité les chercheurs à utiliser les mêmes techniques de test manufacturier pour les deux types de circuits. Ce projet s'inscrit dans la même ligne de pensée et a conduit à l'adaptation de la technique de test CDI_{DDQ}, développée au départ pour les ASIC, aux circuits programmables FPGA.

Le premier objectif de ce mémoire était l'adaptation de la technique aux tests dépendants de l'application. En d'autres termes, dans cette partie, nous avons pu appliquer les vecteurs de test CDI_{DDQ} à un circuit ASIC émulé dans un FPGA. L'accomplissement de cette tâche a nécessité une révision générale du processus de génération des vecteurs de test dans le domaine ASIC. Ainsi, à chaque étape de processus, il a fallu décrypter les différentes variables utilisées dans ce domaine pour les adapter convenablement au domaine FPGA. Le décryptage réalisé nous a permis de bénéficier des hautes performances offertes par les simulateurs logiques des circuits ASIC, et d'intégrer ces performances dans le domaine des FPGA. Ceci est expliqué par le même taux de couverture des courts-circuits obtenu dans l'environnement ASIC et dans l'environnement FPGA, qui a dépassé 90%.

Le deuxième objectif du mémoire était l'introduction d'une nouvelle stratégie de configuration des circuits FPGA pour appliquer la technique CDI_{DDQ} à des lignes configurées comme des peignes entrelacés. Cette configuration permet de détecter les pannes de courts-circuits dans des ressources de routage ciblées, à savoir les lignes horizontales doubles nord de Spartan 3e de la compagnie Xilinx. Le choix de la structure de configuration a été basé sur deux critères principaux. Premièrement, nous avons visé l'obtention d'un nombre minimum de configurations et de vecteurs de test pour un taux de couverture élevé de test indépendant de l'application. Deuxièmement, nous avons pensé à la possibilité d'automatiser la configuration et de l'étendre facilement pour couvrir les autres types d'interconnections et même de pouvoir tester d'autres générations contemporaines des FPGAs. Nous avons obtenu un taux de couverture de 100% pour les lignes ciblées pour une seule configuration et quatre

mesures de courant. Ceci permet d'avoir un temps de test raisonnable, inférieur à 100 ms selon nos estimations. Selon le meilleur de nos connaissances, ces résultats présentent un facteur important dans l'optimisation de temps de test, donc de la minimisation de l'opération de test dans le coût de fabrication de ces circuits intégrés.

Finalement, dans le but de valider expérimentalement les approches discutées, nous avons développé une plateforme de test à faible coût pour adapter la technique de test CDI_{DDQ} aux circuits FPGA. Nous avons prouvé que la technique était fonctionnelle pour les deux types de test, dépendants et indépendants de l'application.

En conclusion, nous pouvons affirmer que l'utilisation de la technique de test CDI_{DDQ} pour les circuits programmables FPGA peut être considérée comme une solution efficace face à la diminution de la taille des transistors.

RECOMMANDATIONS

Les recommandations et les travaux futurs peuvent cibler les tests dépendants et indépendants de l'application. Premièrement, il serait intéressant de maîtriser davantage la technique de conversion des netlists du domaine ASIC vers le domaine FPGA. Dans ce contexte, nous recommandons le développement d'un convertisseur automatique capable de fonctionner avec des circuits de haute performance et de l'intégrer dans les outils utilisés par la compagnie Xilinx.

La deuxième recommandation est au niveau du test indépendant de l'application. Nous proposons en premier lieu d'étendre l'algorithme de configuration pour couvrir toutes les lignes d'interconnections de la puce ciblée. En second lieu, nous proposons de cibler les matrices de connexion qui font partie du réseau de routage. Dans ce cadre, le choix de l'algorithme de routage adéquat avec la technique de test constituera un vrai défi.

Une dernière recommandation est au niveau du test des tranches. Trouver le scénario de configuration convenable aux tranches est essentiel pour l'adaptation de la technique de test $C\Delta I_{DDQ}$. Nous recommandons de concevoir un circuit qui utilise dans son architecture toutes les composantes d'une tranche.

ANNEXE I

EXTRAIT DE NETLIST DE S1196 VERSION ASIC

```
//  file :s1196.v
//# 14 inputs
//# 14 outputs
//# 18 D-type flipflops
//# 141 inverters
//# 388 gates (118 ANDs + 119 NANDs + 101 ORs + 50 NORs)
`timescale 1ns / 1ns
module s1196(INIT,CK,G0,G1,G10,G11,G12,G13,G2,G3,G4,G45,G5,G530,G532,G535,
G537,G539,
G542,G546,G547,G548,G549,G550,G551,G552,G6,G7,G8,G9);
input INIT,CK,G0,G1,G2,G3,G4,G5,G6,G7,G8,G9,G10,G11,G12,G13;
output
G546,G539,G550,G551,G552,G547,G548,G549,G530,G45,G542,G532,G535,G537;

  wdrp_2 DFF_0 (  VDD, VSS,  G29, QB1, G502, CK,   INIT);
  wdrp_2 DFF_1 (  VDD, VSS,  G30, QB2,  G503, CK,  INIT);
  wdrp_2 DFF_2 (  VDD, VSS,  G31, QB3,  G504, CK,  INIT);
  wdrp_2 DFF_3 (  VDD, VSS,  G32, QB4,  G505, CK,  INIT);
  wdrp_2 DFF_4 (  VDD, VSS,  G33, QB5,  G506, CK,  INIT);
  wdrp_2 DFF_5 (  VDD, VSS,  G34, QB6,  G507, CK,  INIT);
  wdrp_2 DFF_6 (  VDD, VSS,  G35, QB7,  G508, CK,  INIT);
  wdrp_2 DFF_7 (  VDD, VSS,  G36, QB8,  G509, CK,  INIT);
  wdrp_2 DFF_8 (  VDD, VSS,  G37, QB9,  G510, CK,  INIT);
  wdrp_2 DFF_9 (  VDD, VSS,  G38, QB10,  G511, CK,  INIT);
  wdrp_2 DFF_10 (  VDD, VSS, G39, QB11,  G512, CK,  INIT);
  wdrp_2 DFF_11 (  VDD, VSS, G40, QB12,  G513, CK,  INIT);
  wdrp_2 DFF_12 (  VDD, VSS, G41, QB13,  G514, CK,  INIT);
  wdrp_2 DFF_13 (  VDD, VSS, G42, QB14,  G515, CK,  INIT);
  wdrp_2 DFF_14 (  VDD, VSS, G43, QB15,  G516, CK,  INIT);
  wdrp_2 DFF_15 (  VDD, VSS, G44, QB16,  G517, CK,  INIT);
  wdrp_2 DFF_16 (  VDD, VSS, G45, QB17,  G518, CK,  INIT);
  wdrp_2 DFF_17 (  VDD, VSS, G46, QB18,  G519, CK,  INIT);

  winv_1 NOT_0 (  VDD, VSS, G520, G0 );
  winv_1 NOT_1 (  VDD, VSS, G521, G1 );
  winv_1 NOT_2 (  VDD, VSS, G522, G2 );
  winv_1 NOT_3 (  VDD, VSS, G524, G3 );

  winv_1 NOT_90 (  VDD, VSS, G107, G456 );
  winv_1 NOT_91 (  VDD, VSS, G207, G208 );
  winv_1 NOT_92 (  VDD, VSS, G167, G168 );
  winv_1 NOT_93 (  VDD, VSS, G124, G206 );
  winv_1 NOT_94 (  VDD, VSS, G203, G204 );
  winv_1 NOT_95 (  VDD, VSS, G489, G273 );
  winv_1 NOT_96 (  VDD, VSS, G495, G273 );
  winv_1 NOT_97 (  VDD, VSS, G177, G357 );
  winv_1 NOT_98 (  VDD, VSS, G212, G213 );
  winv_1 NOT_99 (  VDD, VSS, II493, G218 );
```

```
winv_1 NOT_100 (  VDD, VSS, G404, II493 );
winv_1 NOT_101 (  VDD, VSS, II502, G124 );
winv_1 NOT_102 (  VDD, VSS, G468, II502 );
wand2_1 AND2_78 (  VDD, VSS, G443, G47, G162 );
wand2_1 AND2_79 (  VDD, VSS, G416, G61, G167 );
wand3_1 AND3_32 (  VDD, VSS, G427, G541, G95, G165 );
wand2_1 AND2_80 (  VDD, VSS, G442, G541, G121 );
wand2_1 AND2_81 (  VDD, VSS, G423, G541, G128 );
wand2_1 AND2_82 (  VDD, VSS, G448, G139, G153 );
wor2_1 OR2_0 (  VDD, VSS, G419, G3, G5 );
wor2_1 OR2_1 (  VDD, VSS, G193, G6, G30 );
wor2_1 OR2_2 (  VDD, VSS, G394, G5, G58 );
wor2_1 OR2_3 (  VDD, VSS, G407, G6, G117 );
wor2_1 OR2_4 (  VDD, VSS, G314, G527, G57 );
wor2_1 OR2_5 (  VDD, VSS, G395, G4, G134 );
wor2_1 OR2_6 (  VDD, VSS, G288, G1, G528 );
wor2_1 OR2_7 (  VDD, VSS, G302, G4, G529 );
wor2_1 OR2_8 (  VDD, VSS, G224, G533, G31 );
wor2_1 OR2_9 (  VDD, VSS, G355, G11, G116 );
wor2_1 OR2_10 (  VDD, VSS, G316, G531, G536 );
wor2_1 OR2_11 (  VDD, VSS, G350, G6, G536 );
wor2_1 OR2_12 (  VDD, VSS, G368, G533, G536 );
wor2_1 OR2_13 (  VDD, VSS, G381, G7, G71 );
wor2_1 OR2_14 (  VDD, VSS, G384, G529, G71 );
wor2_1 OR2_15 (  VDD, VSS, G389, G9, G274 );
wor2_1 OR2_16 (  VDD, VSS, G374, G536, G538 );
wor2_1 OR2_17 (  VDD, VSS, G286, G9, G540 );
wor2_1 OR2_18 (  VDD, VSS, G293, G7, G540 );
wor2_1 OR2_19 (  VDD, VSS, G375, G10, G540 );
wnand2_1 NAND2_48 (  VDD, VSS, G278, G332, G333 );
wnand3_1 NAND3_5 (  VDD, VSS, G255, G309, G2, G529 );
wnand3_1 NAND3_6 (  VDD, VSS, G69, G419, G420, G233 );
wnand2_1 NAND2_49 (  VDD, VSS, G512, G310, G233 );
wnand2_1 NAND2_50 (  VDD, VSS, G181, G2, G78 );
wnand3_1 NAND3_7 (  VDD, VSS, G277, G394, G395, G81 );
wnand2_1 NAND2_51 (  VDD, VSS, G151, G305, G200 );
wnand3_1 NAND3_8 (  VDD, VSS, G48, G407, G408, G409 );
wnand2_1 NAND2_52 (  VDD, VSS, G264, G227, G241 );
wnand2_1 NAND2_53 (  VDD, VSS, G208, G68, G229 );
wnand2_1 NAND2_54 (  VDD, VSS, G168, G75, G221 );
wnand2_1 NAND2_55 (  VDD, VSS, G84, G369, G370 );
wnand3_1 NAND3_9 (  VDD, VSS, G258, G464, G103, G223 );
wnand2_1 NAND2_56 (  VDD, VSS, G166, G7, G50 );
wnand2_1 NAND2_57 (  VDD, VSS, G259, G130, G225 );
wnand2_1 NAND2_58 (  VDD, VSS, G504, G292, G293 );
wnand2_1 NAND2_59 (  VDD, VSS, G217, G50, G230 );
wnand2_1 NAND2_60 (  VDD, VSS, G257, G538, G230 );
wnand3_1 NAND3_10 (  VDD, VSS, G260, G528, G529, G191 );
wnand2_1 NAND2_61 (  VDD, VSS, G266, G524, G96 );
wnand2_1 NAND2_62 (  VDD, VSS, G262, G527, G278 );
wnand2_1 NAND2_63 (  VDD, VSS, G138, G465, G263 );
wnand2_1 NAND2_64 (  VDD, VSS, G256, G4, G69 );
wnand2_1 NAND2_65 (  VDD, VSS, G82, G334, G335 );
wnand2_1 NAND2_66 (  VDD, VSS, G109, G269, G219 );
```

```
  wnor2_1 NOR2_36 (  VDD, VSS, G186, G282, G501 );
  wnor4_1 NOR4_1 (  VDD, VSS, G247, G471, G472, G473, G474 );
  wnor2_1 NOR2_37 (  VDD, VSS, G179, G541, G280 );
  wnor2_1 NOR2_38 (  VDD, VSS, G188, G543, G493 );
  wnor2_1 NOR2_39 (  VDD, VSS, G154, G12, G488 );
  wnor3_1 NOR3_3 (  VDD, VSS, G184, G541, G13, G499 );
  wnor2_1 NOR2_40 (  VDD, VSS, G506, G311, G312 );
  wnor2_1 NOR2_41 (  VDD, VSS, G155, G13, G480 );
  wnor2_1 NOR2_42 (  VDD, VSS, G162, G185, G498 );
  wnor3_1 NOR3_4 (  VDD, VSS, G514, G372, G373, G478 );
endmodule
```

ANNEXE II

EXTRAIT DE NETLIST DE S1196 VERSION FPGA

```
module s1196 ( INIT , CK , G0 , G1 , G10 , G11 , G12 , G13 , G2 , G3 , G4
, G45 ,
               G5 , G530 , G532 , G535 , G537 , G539 , G542 , G546 , G547
, G548 ,
               G549 , G550 , G551 , G552 , G6 , G7 , G8 , G9 , scan_in1 ,
               scan_en, scan_out1 );
input   INIT , CK , G0 , G1 , G10 , G11 , G12 , G13 , G2 , G3 , G4 , G5 ,
G6 ,
        G7 , G8 , G9 , scan_in1 , scan_en ;
output  G45 , G530 , G532 , G535 , G537 , G539 , G542 , G546 , G547 , G548
,
        G549 , G550 , G551 , G552, scan_out1 ;

  FDE  DFF_0 ( .Q ( G29 ) ,  .C ( CK ) ,  .D (f0 ) , .CE ( INIT ) );
  FDE  DFF_1 ( .Q ( G30 ) ,  .C ( CK ) ,  .D (f1 ) , .CE ( INIT ) );
  FDE  DFF_2 ( .Q ( G31 ) ,  .C ( CK ) ,  .D (f2 ) , .CE ( INIT ) );
  FDE  DFF_3 ( .Q ( G32 ) ,  .C ( CK ) ,  .D (f3 ) , .CE ( INIT ) );
  FDE  DFF_4 ( .Q ( G33 ) ,  .C ( CK ) ,  .D (f4 ) , .CE ( INIT ) );
  FDE  DFF_5 ( .Q ( G34 ) ,  .C ( CK ) ,  .D (f5 ) , .CE ( INIT ) );
  FDE  DFF_6 ( .Q ( G35 ) ,  .C ( CK ) ,  .D (f6 ) , .CE ( INIT ) );
  FDE  DFF_7 ( .Q ( G36 ) ,  .C ( CK ) ,  .D (f7 ) , .CE ( INIT ) );
  FDE  DFF_8 ( .Q ( G37 ) ,  .C ( CK ) ,  .D (f8 ) , .CE ( INIT ) );
  FDE  DFF_9 ( .Q ( G38 ) ,  .C ( CK ) ,  .D (f9 ) , .CE ( INIT ) );
  FDE  DFF_10 ( .Q ( G39 ) , .C ( CK ) , .D (f10 ) , .CE ( INIT ) );
  FDE  DFF_11 ( .Q ( G40 ) , .C ( CK ) , .D (f11 ) , .CE ( INIT ) );
  FDE  DFF_12 ( .Q ( G41 ) , .C ( CK ) , .D (f12 ) , .CE ( INIT ) );
  FDE  DFF_13 ( .Q ( G42 ) , .C ( CK ) , .D (f13 ) , .CE ( INIT ) );
  FDE  DFF_14 ( .Q ( G43 ) , .C ( CK ) , .D (f14 ) , .CE ( INIT ) );
  FDE  DFF_15 ( .Q ( G44 ) , .C ( CK ) , .D (f15 ) , .CE ( INIT ) );
  FDE  DFF_16 ( .Q ( G45 ) , .C ( CK ) , .D (f16 ) , .CE ( INIT ) );
  FDE  DFF_17 ( .Q ( G46 ) , .C ( CK ) , .D (f17 ) , .CE ( INIT ) );


  INV  INV_DFF_0 ( .O ( scan_out1) , .I ( G29 ) );
  INV  INV_DFF_1 ( .O ( QB2 ) , .I ( G30 ) );
  INV  INV_DFF_2 ( .O ( QB3 ) , .I ( G31 ) );
  INV  INV_DFF_3 ( .O ( QB4 ) , .I ( G32 ) );
  INV  INV_DFF_4 ( .O ( QB5 ) , .I ( G33 ) );
  INV  INV_DFF_5 ( .O ( QB6 ) , .I ( G34 ) );
  INV  INV_DFF_6 ( .O ( QB7 ) , .I ( G35 ) );
  INV  INV_DFF_7 ( .O ( QB8 ) , .I ( G36 ) );
  INV  INV_DFF_8 ( .O ( QB9 ) , .I ( G37 ) );
  INV  INV_DFF_9 ( .O ( QB10 ), .I ( G38 ) );
  INV  INV_DFF_10 ( .O ( QB11 ) , .I ( G39 ) );
  INV  INV_DFF_11 ( .O ( QB12 ) , .I ( G40 ) );
  INV  INV_DFF_12 ( .O ( QB13 ) , .I ( G41 ) );
  INV  INV_DFF_13 ( .O ( QB14 ) , .I ( G42 ) );
  INV  INV_DFF_14 ( .O ( QB15 ) , .I ( G43 ) );
```

```
  INV  INV_DFF_15 ( .O ( QB16 ) , .I ( G44 ) );
  INV  INV_DFF_16 ( .O ( QB17 ) , .I ( G45 ) );
  INV  INV_DFF_17 ( .O ( QB18 ) , .I ( G46 ) );

  MUXCY  MUXCY_DFF_0 (    .O(f0) , .CI ( G502 ),  .DI ( QB2 ) , .S(
scan_en ));
  MUXCY  MUXCY_DFF_1 (    .O(f1) , .CI ( G503 ),  .DI ( QB3 ) , .S(
scan_en ));
  MUXCY  MUXCY_DFF_2 (    .O(f2) , .CI ( G504 ),  .DI ( QB4 ) , .S(
scan_en ));
  MUXCY  MUXCY_DFF_3 (    .O(f3) , .CI ( G505 ),  .DI ( QB5 ) , .S(
scan_en ));
  MUXCY  MUXCY_DFF_4 (    .O(f4) , .CI ( G506 ),  .DI ( QB6 ) , .S(
scan_en ));
  MUXCY  MUXCY_DFF_5 (    .O(f5) , .CI ( G507 ),  .DI ( QB7 ) , .S(
scan_en ));
  MUXCY  MUXCY_DFF_6 (    .O(f6) , .CI ( G508 ),  .DI ( QB8 ) , .S(
scan_en ));
  MUXCY  MUXCY_DFF_7 (    .O(f7) , .CI ( G509 ),  .DI ( QB9 ) , .S(
scan_en ));
  MUXCY  MUXCY_DFF_8 (    .O(f8) , .CI ( G510 ),  .DI ( QB10 ) , .S(
scan_en ));
  MUXCY  MUXCY_DFF_9 (    .O(f9) , .CI ( G511 ),  .DI ( QB11 ) , .S(
scan_en ));
  MUXCY  MUXCY_DFF_10 (   .O(f10) , .CI ( G512 ),  .DI ( QB12 ) , .S(
scan_en ));
  MUXCY  MUXCY_DFF_11 (   .O(f11) , .CI ( G513 ),  .DI ( QB13 ) , .S(
scan_en ));
  MUXCY  MUXCY_DFF_12 (   .O(f12) , .CI ( G514 ),  .DI ( QB14 ) , .S(
scan_en ));
  MUXCY  MUXCY_DFF_13 (   .O(f13) , .CI ( G515 ),  .DI ( QB15 ) , .S(
scan_en ));
  MUXCY  MUXCY_DFF_14 (   .O(f14) , .CI ( G516 ),  .DI ( QB16 ) , .S(
scan_en ));
  MUXCY  MUXCY_DFF_15 (   .O(f15) , .CI ( G517 ),  .DI ( QB18 ) , .S(
scan_en ));
  MUXCY  MUXCY_DFF_16 (   .O(f16) , .CI ( G518 ),  .DI ( QB1 )  , .S(
scan_en ));
  MUXCY  MUXCY_DFF_17 (   .O(f17) , .CI ( G519 ),  .DI ( scan_in1 ) , .S(
scan_en ));

  NOR2  NOR2_0 ( .O ( G163 ) , .I0 ( G0 ) ,
         .I1 ( G4 ));
  NOR2  NOR2_1 ( .O ( G216 ) , .I0 ( G4 ) ,
         .I1 ( G5 ));
  NOR2  NOR2_2 ( .O ( G169 ) , .I0 ( G5 ) ,
         .I1 ( G7 ));
  NOR2  NOR2_3 ( .O ( G225 ) , .I0 ( G7 ) ,
         .I1 ( G8 ));
  NOR2  NOR2_4 ( .O ( G190 ) , .I0 ( G7 ) ,
         .I1 ( G11 ));
  NOR2  NOR2_5 ( .O ( G241 ) , .I0 ( G10 ) ,
         .I1 ( G11 ));
  NOR2  NOR2_6 ( .O ( G198 ) , .I0 ( G520 ) ,
         .I1 ( G3 ));
```

```
NOR2  NOR2_7 ( .O ( G178 ) , .I0 ( G521 ) ,
       .I1 ( G4 ));
NOR2  NOR2_8 ( .O ( G229 ) , .I0 ( G1 ) ,
       .I1 ( G522 ));
NOR2  NOR2_9 ( .O ( G209 ) , .I0 ( G1 ) ,
       .I1 ( G524 ));
NOR2  NOR2_10 ( .O ( G195 ) , .I0 ( G521 ) ,
       .I1 ( G134 ));
NOR2  NOR2_11 ( .O ( G189 ) , .I0 ( G522 ) ,
       .I1 ( G54 ));
NOR2  NOR2_12 ( .O ( G201 ) , .I0 ( G528 ) ,
       .I1 ( G54 ));
NOR2  NOR2_13 ( .O ( G164 ) , .I0 ( G531 ) ,
       .I1 ( G10 ));
NOR2  NOR2_14 ( .O ( G211 ) , .I0 ( G6 ) ,
       .I1 ( G274 ));
NOR2  NOR2_15 ( .O ( G156 ) , .I0 ( G12 ) ,
       .I1 ( G543 ));
NOR2  NOR2_16 ( .O ( G205 ) , .I0 ( G529 ) ,
       .I1 ( G122 ));
NOR2  NOR2_17 ( .O ( G227 ) , .I0 ( G5 ) ,
       .I1 ( G200 ));
NOR2  NOR2_18 ( .O ( G230 ) , .I0 ( G8 ) ,
       .I1 ( G490 ));
NOR2  NOR2_19 ( .O ( G191 ) , .I0 ( G9 ) ,
       .I1 ( G482 ));
NOR3  NOR3_0 ( .O ( G196 ) , .I0 ( G5 ) ,
       .I1 ( G540 ) , .I2 ( G86 ));
NOR2  NOR2_20 ( .O ( G197 ) , .I0 ( G540 ) ,
       .I1 ( G232 ));
NOR2  NOR2_21 ( .O ( G202 ) , .I0 ( G10 ) ,
       .I1 ( G63 ));
NOR2  NOR2_22 ( .O ( G502 ) , .I0 ( G436 ) ,
       .I1 ( G437 ));
NOR2  NOR2_23 ( .O ( G218 ) , .I0 ( G528 ) ,
       .I1 ( G217 ));
NOR3  NOR3_1 ( .O ( G516 ) , .I0 ( G410 ) ,
       .I1 ( G411 ) , .I2 ( G412 ));
NOR2  NOR2_24 ( .O ( G515 ) , .I0 ( G387 ) ,
       .I1 ( G388 ));
NOR2  NOR2_25 ( .O ( G509 ) , .I0 ( G331 ) ,
       .I1 ( G5 ));
NOR2  NOR2_26 ( .O ( G513 ) , .I0 ( G360 ) ,
      .I1 ( G536 ) , .I2 ( G222 ));
OR2  OR2_29 ( .O ( G464 ) , .I0 ( G72 ) ,
      .I1 ( G536 ));
OR2  OR2_30 ( .O ( G391 ) , .I0 ( G74 ) ,
      .I1 ( G220 ));
OR2  OR2_31 ( .O ( G292 ) , .I0 ( G538 ) ,
      .I1 ( G75 ));
OR2  OR2_32 ( .O ( G345 ) , .I0 ( G529 ) ,
      .I1 ( G226 ));
OR2  OR2_33 ( .O ( G465 ) , .I0 ( G524 ) ,
      .I1 ( G210 ));
OR2  OR2_34 ( .O ( G454 ) , .I0 ( G122 ) ,
```

```
      .I1 ( G77 ));
OR2  OR2_35 ( .O ( G269 ) , .I0 ( G362 ) ,
      .I1 ( G529 ));
OR2  OR2_36 ( .O ( G287 ) , .I0 ( G522 ) ,
      .I1 ( G81 ));
OR3  OR3_2 ( .O ( G318 ) , .I0 ( G6 ) ,
      .I1 ( G8 ) , .I2 ( G232 ));
OR2  OR2_37 ( .O ( G326 ) , .I0 ( G533 ) ,
      .I1 ( G232 ));
OR2  OR2_38 ( .O ( G390 ) , .I0 ( G89 ) ,
      .I1 ( G50 ));
OR2  OR2_39 ( .O ( G298 ) , .I0 ( G5 ) ,
      .I1 ( G497 ));
OR2  OR2_40 ( .O ( G300 ) , .I0 ( G87 ) ,
      .I1 ( G97 ));
OR2  OR2_41 ( .O ( G261 ) , .I0 ( G283 ) ,
      .I1 ( G528 ));
OR2  OR2_42 ( .O ( G301 ) , .I0 ( G122 ) ,
      .I1 ( G486 ));
OR2  OR2_43 ( .O ( G92 ) , .I0 ( G351 ) ,
      .I1 ( G352 ));
OR2  OR2_44 ( .O ( G47 ) , .I0 ( G440 ) ,
       .I1 ( G6 ));
NAND2  NAND2_9 ( .O ( G116 ) , .I0 ( G6 ) ,
       .I1 ( G9 ));
NAND2  NAND2_10 ( .O ( G71 ) , .I0 ( G8 ) ,
       .I1 ( G10 ));
NAND2  NAND2_11 ( .O ( G274 ) , .I0 ( G7 ) ,
       .I1 ( G10 ));
NAND2  NAND2_12 ( .O ( G74 ) , .I0 ( G9 ) ,
       .I1 ( G11 ));
NAND2  NAND2_13 ( .O ( G112 ) , .I0 ( G8 ) ,
       .I1 ( G31 ));
NAND2  NAND2_14 ( .O ( G245 ) , .I0 ( G8 ) ,
       .I1 ( G34 ));
NAND2  NAND2_15 ( .O ( G122 ) , .I0 ( G522 ) ,
       .I1 ( G3 ));
NAND2  NAND2_16 ( .O ( G238 ) , .I0 ( G2 ) ,
       .I1 ( G524 ));
NAND2  NAND2_17 ( .O ( G129 ) , .I0 ( G527 ) ,
       .I1 ( G5 ));
NAND2  NAND2_18 ( .O ( G240 ) , .I0 ( G4 ) ,
       .I1 ( G134 ));
NAND4  NAND4_0 ( .O ( G252 ) , .I0 ( G3 ) ,
       .I1 ( G11 ) , .I2 ( G35 ) , .I3 ( G216 ));
NAND2  NAND2_19 ( .O ( G77 ) , .I0 ( G4 ) ,
       .I1 ( G528 ));
NAND3  NAND3_0 ( .O ( G103 ) , .I0 ( G529 ) ,
       .I1 ( G7 ) , .I2 ( G30 ));
NAND2  NAND2_20 ( .O ( G200 ) , .I0 ( G527 ) ,
       .I1 ( G529 ));
NAND2  NAND2_21 ( .O ( G248 ) , .I0 ( G529 ) ,
       .I1 ( G36 ));
NAND2  NAND2_22 ( .O ( G89 ) , .I0 ( G531 ) ,
       .I1 ( G8 ));
```

```
  NAND2  NAND2_23 ( .O ( G222 ) , .I0 ( G533 ) ,
          .I1 ( G10 ));
  NAND2  NAND2_57 ( .O ( G259 ) , .I0 ( G130 ) ,
          .I1 ( G225 ));
  NAND2  NAND2_58 ( .O ( G504 ) , .I0 ( G292 ) ,
          .I1 ( G293 ));
  NAND2  NAND2_59 ( .O ( G217 ) , .I0 ( G50 ) ,
          .I1 ( G230 ));
  NAND2  NAND2_60 ( .O ( G257 ) , .I0 ( G538 ) ,
          .I1 ( G230 ));

  AND3  AND3_0 ( .O ( G385 ) , .I0 ( G529 ) ,
         .I1 ( G7 ) , .I2 ( G49 ));
  AND2  AND2_9 ( .O ( G376 ) , .I0 ( G533 ) ,
         .I1 ( G75 ));
) ,
         .I1 ( G154 ));
  AND2  AND2_68 ( .O ( G449 ) , .I0 ( G88 ) ,
         .I1 ( G154 ));
  AND2  AND2_69 ( .O ( G452 ) , .I0 ( G526 ) ,
         .I1 ( G184 ));
  AND2  AND2_70 ( .O ( G329 ) , .I0 ( G150 ) ,
         .I1 ( G156 ));
  AND2  AND2_71 ( .O ( G291 ) , .I0 ( G138 ) ,
         .I1 ( G155 ));
         .I1 ( G167 ));
  AND3  AND3_32 ( .O ( G427 ) , .I0 ( G541 ) ,
         .I1 ( G95 ) , .I2 ( G165 ));
  AND2  AND2_80 ( .O ( G442 ) , .I0 ( G541 ) ,
         .I1 ( G121 ));
  AND2  AND2_81 ( .O ( G423 ) , .I0 ( G541 ) ,
         .I1 ( G128 ));
  AND2  AND2_82 ( .O ( G448 ) , .I0 ( G139 ) ,
         .I1 ( G153 ));

  INV  NOT_108 ( .O ( G446 ) , .I ( II536 ));
  INV  NOT_109 ( .O ( G494 ) , .I ( G173 ));
  INV  NOT_110 ( .O ( G500 ) , .I ( G173 ));
  INV  NOT_111 ( .O ( G214 ) , .I ( G215 ));
  INV  NOT_112 ( .O ( G492 ) , .I ( G62 ));
  INV  NOT_113 ( .O ( G483 ) , .I ( G182 ));
  INV  NOT_114 ( .O ( G282 ) , .I ( G281 ));
  INV  NOT_115 ( .O ( II573 ) , .I ( G176 ));
  INV  NOT_116 ( .O ( G403 ) , .I ( II573 ));
  INV  NOT_117 ( .O ( II576 ) , .I ( G175 ));
  INV  NOT_118 ( .O ( G447 ) , .I ( II576 ));
  INV  NOT_119 ( .O ( G479 ) , .I ( G194 ));
  INV  NOT_120 ( .O ( G491 ) , .I ( G194 ));
  INV  NOT_121 ( .O ( G554 ) , .I ( G553 ));
  INV  NOT_122 ( .O ( G170 ) , .I ( G171 ));
  INV  NOT_123 ( .O ( G172 ) , .I ( G171 ));
  INV  NOT_124 ( .O ( G525 ) , .I ( G526 ));
  INV  NOT_125 ( .O ( G493 ) , .I ( G544 ));
  INV  NOT_126 ( .O ( G545 ) , .I ( G544 ));
  INV  NOT_127 ( .O ( G488 ) , .I ( G172 ));
```

```
  INV  NOT_128 ( .O ( G499 ) , .I ( G280 ));
  INV  NOT_129 ( .O ( II624 ) , .I ( G120 ));
  INV  NOT_130 ( .O ( G303 ) , .I ( II624 ));
  INV  NOT_131 ( .O ( G480 ) , .I ( G179 ));
  INV  NOT_132 ( .O ( II631 ) , .I ( G188 ));
  INV  NOT_133 ( .O ( G336 ) , .I ( II631 ));
  INV  NOT_134 ( .O ( G496 ) , .I ( G188 ));
  INV  NOT_135 ( .O ( G174 ) , .I ( G496 ));
  INV  NOT_136 ( .O ( II662 ) , .I ( G174 ));
  INV  NOT_137 ( .O ( G405 ) , .I ( II662 ));
  INV  NOT_138 ( .O ( G478 ) , .I ( G279 ));
  INV  NOT_139 ( .O ( II692 ) , .I ( G145 ));
  INV  NOT_140 ( .O ( G432 ) , .I ( II692 ));

endmodule
```

ANNEXE III

FICHIER DE SÉQUENCE GÉNÉRÉ PAR FASTSCAN

```
/
//  FastScan  v8.2003_5.10
//
//  Design  =
/home/projet/prj_0091/Mentor/TDF_IDDQ/s1196x/s1196x_pads_syn_fs.v
//  Created = Mon Dec 11 15:35:19 2006
//  Format  = scan v1.0
//
//  Statistics:
//      Test Coverage    =  99.17%
//      Total Faults     =  6054
//          DS (det_simulation)     =  3367
//          DI (det_implication)    =  118
//          UU (unused)             =  2540
//          AU (atpg_untestable)    =  29
//      Total             Patterns  =  420
//
//  Settings:
//      Simulation Mode =  combinational, seq_depth = 0
//      Fault Type      =  transition
//      Fault Mode      =  uncollapsed
//      Pos_Det Credit  =  50%
//      Z external      =  X
//      Z internal      =  X
//      wired_net       =  WIRE
//
//  Warnings:
//      circuit contains 1 wired nets
//
//  Clock Information:
//      /INIT  off-state = 1  type = reset-MASTER  poflag = 0
//      /CK  off-state = 0  type = shift-MASTER  poflag = 0
//

SETUP =

    declare input bus "PI" = "/INIT", "/CK", "/G0", "/G1", "/G10",
         "/G11", "/G12", "/G13", "/G2", "/G3", "/G4",
         "/G5", "/G6", "/G7", "/G8", "/G9", "/scan_in1",
         "/scan_en";

    declare output bus "PO" = "/G45", "/G530", "/G532", "/G535", "/G537",
         "/G539", "/G542", "/G546", "/G547", "/G548", "/G549",
         "/G550", "/G551", "/G552", "/scan_out1";

    clock "/INIT" =
        off_state   = 1;
        pulse_width = 1;
```

```
    end;

    clock "/CK" =
        off_state   = 0;
        pulse_width = 1;
    end;

    scan_group "grp1" =
        scan_chain "chain1" =
            scan_in = "/scan_in1";
            scan_out = "/scan_out1";
            length = 18;
        end;
        procedure shift "grp1_load_shift" =
            force_sci "chain1" 0;
            force "/CK" 1 20;
            force "/CK" 0 30;
            period 40;
        end;
        procedure shift "grp1_unload_shift" =
            measure_sco "chain1" 10;
            force "/CK" 1 20;
            force "/CK" 0 30;
            period 40;
        end;
        procedure load "grp1_load" =
            force "/CK" 0 0;
            force "/INIT" 1 0;
            force "/scan_en" 1 0;
            apply "grp1_load_shift" 18 40;
        end;
        procedure unload "grp1_unload" =
            force "/CK" 0 0;
            force "/INIT" 1 0;
            force "/scan_en" 1 0;
            apply "grp1_unload_shift" 18 40;
        end;
    end;

end;

CHAIN_TEST =

    pattern = 0;
    apply "grp1_load" 0 =
        chain "chain1" = "0011001100110011100";
    end;
    apply "grp1_unload" 1 =
        chain "chain1" = "0011001100110011100";
    end;

end;

SCAN_TEST =
```

```
pattern = 0;
force   "PI" "1000110101100110X1" 0;
apply "grp1_load" 1 =
    chain "chain1" = "0001010010110000100";
end;
force   "PI" "1010011000011101100" 2;
measure "PO" "1000000011000000" 3;
pulse "/CK" 4;
apply "grp1_unload" 5 =
    chain "chain1" = "0010101000100100011";
end;

pattern = 1;
force   "PI" "1000011010110011X1" 0;
apply "grp1_load" 1 =
    chain "chain1" = "1111010110100111101";
end;
force   "PI" "1011001100001101110" 2;
measure "PO" "1000000010000001" 3;
pulse "/CK" 4;
apply "grp1_unload" 5 =
    chain "chain1" = "0010111010110100001";
end;

pattern = 2;
force   "PI" "1010111000111111X1" 0;
apply "grp1_load" 1 =
    chain "chain1" = "1011101100110111111";
end;
force   "PI" "1001111110011101111" 2;
measure "PO" "0000001011000001" 3;
pulse "/CK" 4;
apply "grp1_unload" 5 =
    chain "chain1" = "0111011001101111111";
end;

pattern = 3;
force   "PI" "1001101001111001X1" 0;
apply "grp1_load" 1 =
    chain "chain1" = "0001110001111111110";
end;
force   "PI" "1000100111010101111" 2;
measure "PO" "1010001010010000" 3;
pulse "/CK" 4;
apply "grp1_unload" 5 =
    chain "chain1" = "0011100011111111101";
end;

...
...

pattern = 14;
force   "PI" "1010100111001001X1" 0;
apply "grp1_load" 1 =
    chain "chain1" = "1101011000111000011";
```

```
    end;
    force   "PI" "101101101100110110" 2;
    measure "PO" "011000111111001" 3;
    pulse "/CK" 4;
    apply "grp1_unload" 5 =
        chain "chain1" = "0010111111011100011";
    end;
SCAN_CELLS =

    scan_group "grp1" =
        scan_chain "chain1" =
            scan_cell =    0  MASTER  TFFT  "/DFF_0"  "_i2/mlc_dff2"  "SD"
"qb";
            scan_cell =    1  MASTER  FTFT  "/DFF_1"  "_i2/mlc_dff2"  "SD"
"qb";
            scan_cell =    2  MASTER  TFFT  "/DFF_6"  "_i2/mlc_dff2"  "SD"
"qb";
            scan_cell =    3  MASTER  FTFT  "/DFF_7"  "_i2/mlc_dff2"  "SD"
"qb";
            scan_cell =    4  MASTER  TFFT  "/DFF_9"  "_i2/mlc_dff2"  "SD"
"qb";
            scan_cell =    5  MASTER  FTFT  "/DFF_11"  "_i2/mlc_dff2"
"SD"  "qb";
            scan_cell =    6  MASTER  TFFT  "/DFF_16"  "_i2/mlc_dff2"
"SD"  "qb";
            scan_cell =    7  MASTER  FTFT  "/DFF_10"  "_i2/mlc_dff2"
"SD"  "qb";
            scan_cell =    8  MASTER  TFFT  "/DFF_2"  "_i2/mlc_dff2"  "SD"
"qb";
            scan_cell =    9  MASTER  FTFT  "/DFF_3"  "_i2/mlc_dff2"  "SD"
"qb";
            scan_cell =   10  MASTER  TFFT  "/DFF_8"  "_i2/mlc_dff2"  "SD"
"qb";
            scan_cell =   11  MASTER  FTFT  "/DFF_17"  "_i2/mlc_dff2"
"SD"  "qb";
            scan_cell =   12  MASTER  TFFT  "/DFF_4"  "_i2/mlc_dff2"  "SD"
"qb";
            scan_cell =   13  MASTER  FTFT  "/DFF_12"  "_i2/mlc_dff2"
"SD"  "qb";
            scan_cell =   14  MASTER  TFFT  "/DFF_15"  "_i2/mlc_dff2"
"SD"  "qb";
            scan_cell =   15  MASTER  FTFT  "/DFF_5"  "_i2/mlc_dff2"  "SD"
"qb";
            scan_cell =   16  MASTER  TFFT  "/DFF_13"  "_i2/mlc_dff2"
"SD"  "qb";
            scan_cell =   17  MASTER  FTFT  "/DFF_14"  "_i2/mlc_dff2"
"SD"  "qb";
        end;
    end;

end;
```

ANNEXE IV

FICHIER DE VALEUR GÉNÉRÉ PAR FASTSCAN

```
--
-- VHDL format test patterns produced by FastScan v8.2008_2.10
-- Filename        :
/home/ens/khaitem/Desktop/S1196/s1196_pads_fs_vec.v.0.vec
-- Scan operation : SERIAL
-- Idstamp        : v8.2008_2.10:59b8:bad3:270:158a
-- Fault          : TRANSITION
-- Coverage       : 99.40(TC) 58.20(FC)
-- Date           : Wed Mar 24 10:38:02 2010
--
-- Pattern fields:
--  pattern number
--  timeplate number
--  repeat_count
--  input vector
--  output vector
--  output mask vector
--
 0 1 1 10XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXX 000000000000000

-- Chain test block

-- Chain Pattern 0
-- Begin chain test
 0 1 1 10XXXXXXXXXXXXXXX1 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 11XXXXXXXXXXXXXX01 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 11XXXXXXXXXXXXXX01 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 11XXXXXXXXXXXXXX11 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 11XXXXXXXXXXXXXX11 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 11XXXXXXXXXXXXXX01 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 11XXXXXXXXXXXXXX01 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 11XXXXXXXXXXXXXX11 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 11XXXXXXXXXXXXXX11 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 11XXXXXXXXXXXXXX01 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 11XXXXXXXXXXXXXX01 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 11XXXXXXXXXXXXXX11 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 11XXXXXXXXXXXXXX11 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 11XXXXXXXXXXXXXX01 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 11XXXXXXXXXXXXXX01 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 11XXXXXXXXXXXXXX11 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 11XXXXXXXXXXXXXX11 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 11XXXXXXXXXXXXXX01 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 11XXXXXXXXXXXXXX01 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 10XXXXXXXXXXXXXX01 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 11XXXXXXXXXXXXXX01 XXXXXXXXXXXXXX0 000000000000001
 0 1 1 11XXXXXXXXXXXXXX01 XXXXXXXXXXXXXX0 000000000000001
 0 1 1 11XXXXXXXXXXXXXX11 XXXXXXXXXXXXXX1 000000000000001
 0 1 1 11XXXXXXXXXXXXXX11 XXXXXXXXXXXXXX1 000000000000001
```

```
 0 1 1 11XXXXXXXXXXXXXX01 XXXXXXXXXXXXXX0 000000000000001
 0 1 1 11XXXXXXXXXXXXXX01 XXXXXXXXXXXXXX0 000000000000001
 0 1 1 11XXXXXXXXXXXXXX11 XXXXXXXXXXXXXX1 000000000000001
 0 1 1 11XXXXXXXXXXXXXX11 XXXXXXXXXXXXXX1 000000000000001
 0 1 1 11XXXXXXXXXXXXXX01 XXXXXXXXXXXXXX0 000000000000001
 0 1 1 11XXXXXXXXXXXXXX01 XXXXXXXXXXXXXX0 000000000000001
 0 1 1 11XXXXXXXXXXXXXX11 XXXXXXXXXXXXXX1 000000000000001
 0 1 1 11XXXXXXXXXXXXXX11 XXXXXXXXXXXXXX1 000000000000001
 0 1 1 11XXXXXXXXXXXXXX01 XXXXXXXXXXXXXX0 000000000000001
 0 1 1 11XXXXXXXXXXXXXX01 XXXXXXXXXXXXXX0 000000000000001
 0 1 1 11XXXXXXXXXXXXXX11 XXXXXXXXXXXXXX1 000000000000001
 0 1 1 11XXXXXXXXXXXXXX11 XXXXXXXXXXXXXX1 000000000000001
 0 1 1 11XXXXXXXXXXXXXX01 XXXXXXXXXXXXXX0 000000000000001
 0 1 1 11XXXXXXXXXXXXXX01 XXXXXXXXXXXXXX0 000000000000001
-- End chain test

-- Scan test block

-- Pattern 0
 0 1 1 10XXXXXXXXXXXXXXX1 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 11XXXXXXXXXXXXXX11 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 11XXXXXXXXXXXXXX11 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 11XXXXXXXXXXXXXX11 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 11XXXXXXXXXXXXXX01 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 11XXXXXXXXXXXXXX11 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 11XXXXXXXXXXXXXX01 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 11XXXXXXXXXXXXXX11 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 11XXXXXXXXXXXXXX01 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 11XXXXXXXXXXXXXX01 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 11XXXXXXXXXXXXXX01 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 11XXXXXXXXXXXXXX11 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 11XXXXXXXXXXXXXX01 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 11XXXXXXXXXXXXXX11 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 11XXXXXXXXXXXXXX11 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 11XXXXXXXXXXXXXX11 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 11XXXXXXXXXXXXXX11 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 11XXXXXXXXXXXXXX01 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 11XXXXXXXXXXXXXX11 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 101001111100010011 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 101111011100111100 000001000000001 111111111111111

-- Pattern 1 Type clock_sequential
 1 1 1 101111011100111101 XXXXXXXXXXXXXXX 000000000000000
 0 1 1 111111011100111111 XXXXXXXXXXXXXX1 000000000000001
 0 1 1 111111011100111111 XXXXXXXXXXXXXX1 000000000000001
 0 1 1 111111011100111101 XXXXXXXXXXXXXX1 000000000000001
 0 1 1 111111011100111111 XXXXXXXXXXXXXX0 000000000000001
 0 1 1 111111011100111101 XXXXXXXXXXXXXX1 000000000000001
 0 1 1 111111011100111101 XXXXXXXXXXXXXX0 000000000000001
 0 1 1 111111011100111101 XXXXXXXXXXXXXX1 000000000000001
 0 1 1 111111011100111101 XXXXXXXXXXXXXX0 000000000000001
 0 1 1 111111011100111111 XXXXXXXXXXXXXX0 000000000000001
 0 1 1 111111011100111111 XXXXXXXXXXXXXX0 000000000000001
 0 1 1 111111011100111101 XXXXXXXXXXXXXX1 000000000000001
 0 1 1 111111011100111101 XXXXXXXXXXXXXX0 000000000000001
```

```
0 1 1 111111011100111101 XXXXXXXXXXXXXXX1 000000000000001
0 1 1 111111011100111101 XXXXXXXXXXXXXXX1 000000000000001
0 1 1 111111011100111101 XXXXXXXXXXXXXXX1 000000000000001
0 1 1 111111011100111111 XXXXXXXXXXXXXXX1 000000000000001
0 1 1 111111011100111101 XXXXXXXXXXXXXXX0 000000000000001
0 1 1 111111011100111101 XXXXXXXXXXXXXXX1 000000000000001
1 1 1 111111010010001001 XXXXXXXXXXXXXXXX 000000000000000
        1  1 1 101111101111111110 100110010000001 111111111111111
```

ANNEXE V

DÉFINITION DES PARAMTERES DU VECTEUR DE TEST

Tableau-A V-1 Définition des paramètres du vecteur de test
Tiré de Jorge Soto (2009, p. 26)

Nom	Bits	Description
PI_bits	52:35	Cette chaîne de bits contient les valeurs des PI à appliquer sur le circuit sous test.
PO_bits	34:20	Valeurs attendues aux sorties du circuit, lorsque soumis au vecteur de test pour le patron courant.
MASK_bits	19:5	Masque à appliquer sur les bits de sorties (*PO_bits*) du vecteur secondaire afin de déterminer les valeurs attendues pour le mot actuel.
Increment_bit	4	Bit indiquant au testeur d'incrémenter le numéro de patrons pour les vecteurs suivants.
Timeplate_bits	3	Ce bit est utilisé pour signaler la validité du vecteur. Il devrait toujours être égal à 1 au risque de terminer précocement la simulation.
PatType_bits	2:0	Chaînes de bits définissant le type du vecteur étant analysé : soit un vecteur de fin, un vecteur de scan, un vecteur secondaire, un vecteur de shift ou un vecteur d'état. Tout autre valeur signifie que le patron est invalide ou corrompu, et termine la simulation.

ANNEXE VI

DÉFINITION DES PARAMÈTRES DU VECTEUR SCAN

Tableau-A VI-1 Définition des paramètres du vecteur scan
Tiré de Jorge Soto (2009, p. 26)

Nom	Bits	Description
SI_bits	56:39	Vecteur de scan utilisé pour initialiser les valeurs dans les registres internes du CUT. Ces bits seront affectés à l'entrée de scan du circuit.
SO_bis	38:21	Valeurs attendues à la sortie de scan du circuit. Cette chaîne sera comparée à celle obtenue lors de la simulation.
MASK_bits	20:3	Cette chaîne de bits au même principe que ceux du vecteur secondaire. Elle sert de masque pour les bits à la sortie du vecteur de balayage (*SO_bits*).
PatType_bits	2:0	Même définition que pour le vecteur secondaire.

ANNEXE VII

ADRESSE OBTENUE PAR LE GÉNÉRATEUR D'ADRESSES

```
----------------------------------------------------------------------------
----
-- Laboratory: Lacime-ETS-CANADA
-- Engineer: Khaled Haithem
-- Supervisor: Claude Thibeault
-- Create Date:    29/09/2010
-- Project Name:  spartan 3e configuration

-- Dependencies: Xilinx
----------------------------------------------------------------------------
------

------------------------sm1/1-----------------------------------
select wire 'DOUBLE(-62515,82072)'
select wire 'DOUBLE(-62511,82060)'
select wire 'DOUBLE(-62507,82048)'
select wire 'DOUBLE(-62503,82036)'
select wire 'DOUBLE(-62499,82024)'
select wire 'DOUBLE(-62495,82012)'
------------------------sm1/2-----------------------------------
select wire 'DOUBLE(-57968,81981)'
select wire 'DOUBLE(-57960,81975)'
select wire 'DOUBLE(-57952,81969)'
select wire 'DOUBLE(-57944,81963)'
select wire 'DOUBLE(-57936,81957)'
select wire 'DOUBLE(-57928,81951)'
------------------------sm2/1-----------------------------------
select wire 'DOUBLE(-61424,81981)'
select wire 'DOUBLE(-61416,81975)'
select wire 'DOUBLE(-61408,81969)'
select wire 'DOUBLE(-61400,81963)'
select wire 'DOUBLE(-61392,81957)'
select wire 'DOUBLE(-61384,81951)'
------------------------sm2/2-----------------------------------
select wire 'DOUBLE(-54512,81981)'
select wire 'DOUBLE(-54504,81975)'
select wire 'DOUBLE(-54496,81969)'
select wire 'DOUBLE(-54488,81963)'
select wire 'DOUBLE(-54480,81957)'
select wire 'DOUBLE(-54472,81951)'
------------------------sm3/1-----------------------------------
select wire 'DOUBLE(-57968,81981)'
select wire 'DOUBLE(-57960,81975)'
select wire 'DOUBLE(-57952,81969)'
select wire 'DOUBLE(-57944,81963)'
select wire 'DOUBLE(-57936,81957)'
select wire 'DOUBLE(-57928,81951)'
```

```
----------------------sm3/2----------------------------------
select wire 'DOUBLE(-51056,81981)'
select wire 'DOUBLE(-51048,81975)'
select wire 'DOUBLE(-51040,81969)'
select wire 'DOUBLE(-51032,81963)'
select wire 'DOUBLE(-51024,81957)'
select wire 'DOUBLE(-51016,81951)'
----------------------sm4/1----------------------------------
select wire 'DOUBLE(-54512,81981)'
select wire 'DOUBLE(-54504,81975)'
select wire 'DOUBLE(-54496,81969)'
select wire 'DOUBLE(-54488,81963)'
select wire 'DOUBLE(-54480,81957)'
select wire 'DOUBLE(-54472,81951)'
----------------------sm4/2----------------------------------
select wire 'DOUBLE(-47600,81981)'
select wire 'DOUBLE(-47592,81975)'
select wire 'DOUBLE(-47584,81969)'
select wire 'DOUBLE(-47576,81963)'
select wire 'DOUBLE(-47568,81957)'
select wire 'DOUBLE(-47560,81951)'
----------------------sm5/1----------------------------------
select wire 'DOUBLE(-51056,81981)'
select wire 'DOUBLE(-51048,81975)'
select wire 'DOUBLE(-51040,81969)'
select wire 'DOUBLE(-51032,81963)'
select wire 'DOUBLE(-51024,81957)'
select wire 'DOUBLE(-51016,81951)'
----------------------sm5/2----------------------------------
select wire 'DOUBLE(-44144,81981)'
select wire 'DOUBLE(-44136,81975)'
select wire 'DOUBLE(-44128,81969)'
select wire 'DOUBLE(-44120,81963)'
select wire 'DOUBLE(-44112,81957)'
select wire 'DOUBLE(-44104,81951)'
----------------------sm6/1----------------------------------
select wire 'DOUBLE(-47600,81981)'
select wire 'DOUBLE(-47592,81975)'
select wire 'DOUBLE(-47584,81969)'
select wire 'DOUBLE(-47576,81963)'
select wire 'DOUBLE(-47568,81957)'
select wire 'DOUBLE(-47560,81951)'
----------------------sm6/2----------------------------------
select wire 'DOUBLE(-40688,81981)'
select wire 'DOUBLE(-40680,81975)'
select wire 'DOUBLE(-40672,81969)'
select wire 'DOUBLE(-40664,81963)'
select wire 'DOUBLE(-40656,81957)'
select wire 'DOUBLE(-40648,81951)'
----------------------sm7/1----------------------------------
select wire 'DOUBLE(-44144,81981)'
select wire 'DOUBLE(-44136,81975)'
select wire 'DOUBLE(-44128,81969)'
select wire 'DOUBLE(-44120,81963)'
select wire 'DOUBLE(-44112,81957)'
```

```
select wire 'DOUBLE(-44104,81951)'
-----------------------sm7/2----------------------------------
select wire 'DOUBLE(-37232,81981)'
select wire 'DOUBLE(-37224,81975)'
select wire 'DOUBLE(-37216,81969)'
select wire 'DOUBLE(-37208,81963)'
select wire 'DOUBLE(-37200,81957)'
select wire 'DOUBLE(-37192,81951)'
-----------------------sm8/1----------------------------------
select wire 'DOUBLE(-40688,81981)'
select wire 'DOUBLE(-40680,81975)'
select wire 'DOUBLE(-40672,81969)'
select wire 'DOUBLE(-40664,81963)'
select wire 'DOUBLE(-40656,81957)'
select wire 'DOUBLE(-40648,81951)'
-----------------------sm8/2----------------------------------
select wire 'DOUBLE(-33776,81981)'
select wire 'DOUBLE(-33768,81975)'
select wire 'DOUBLE(-33760,81969)'
select wire 'DOUBLE(-33752,81963)'
select wire 'DOUBLE(-33744,81957)'
select wire 'DOUBLE(-33736,81951)'
-----------------------sm9/1----------------------------------
select wire 'DOUBLE(-37232,81981)'
select wire 'DOUBLE(-37224,81975)'
select wire 'DOUBLE(-37216,81969)'
select wire 'DOUBLE(-37208,81963)'
select wire 'DOUBLE(-37200,81957)'
select wire 'DOUBLE(-37192,81951)'
-----------------------sm9/2----------------------------------
select wire 'DOUBLE(-30320,81981)'
select wire 'DOUBLE(-30312,81975)'
select wire 'DOUBLE(-30304,81969)'
select wire 'DOUBLE(-30296,81963)'
select wire 'DOUBLE(-30288,81957)'
select wire 'DOUBLE(-30280,81951)'
-----------------------sm10/1---------------------------------
select wire 'DOUBLE(-33776,81981)'
select wire 'DOUBLE(-33768,81975)'
select wire 'DOUBLE(-33760,81969)'
select wire 'DOUBLE(-33752,81963)'
select wire 'DOUBLE(-33744,81957)'
select wire 'DOUBLE(-33736,81951)'
-----------------------sm10/2---------------------------------
select wire 'DOUBLE(-26088,82071)'
select wire 'DOUBLE(-26088,82059)'
select wire 'DOUBLE(-26088,82047)'
select wire 'DOUBLE(-26088,82035)'
select wire 'DOUBLE(-26088,82023)'
select wire 'DOUBLE(-26088,82011)'
```

ANNEXE VIII

SCRIPT POUR LA COUFIGURATION DE LA PREMIÈRE LIGNE

```
unselect -all
select pin 'B4.I'
select wire 'IOBOUTPUT(-49896,80904)'
select wire 'DUMMYESC(-50055,80904)'
select arc 'IOBOUTPUT(-49896,80904)--->DUMMYESC(-50055,80904)'
select wire 'DOUBLE(-62515,82072)'
select arc 'DOUBLE(-62515,82072)--->DOUBLE(-61416,81975)'
select wire 'DOUBLE(-61416,81975)'
select arc 'DOUBLE(-61416,81975)--->DOUBLE(-54504,81975)'
select wire 'DOUBLE(-54504,81975)'
select arc 'DOUBLE(-54504,81975)--->DOUBLE(-47592,81975)'
select wire 'DOUBLE(-47592,81975)'
select arc 'DOUBLE(-47592,81975)--->DOUBLE(-40680,81975)'
select wire 'DOUBLE(-40680,81975)'
select arc 'DOUBLE(-40680,81975)--->DOUBLE(-33768,81975)'
select wire 'DOUBLE(-33768,81975)'
select arc 'DOUBLE(-33768,81975)--->DOUBLE(-26088,82059)'
select wire 'DOUBLE(-26088,82059)'
select arc 'DOUBLE(-26088,82059)--->DOUBLE(-19024,81975)'
select wire 'DOUBLE(-19024,81975)'
select arc 'DOUBLE(-19024,81975)--->DOUBLE(-12112,81975)'
select wire 'DOUBLE(-12112,81975)'
select arc 'DOUBLE(-12112,81975)--->DOUBLE(-5200,81975)'
select wire 'DOUBLE(-5200,81975)'
select arc 'DOUBLE(-5200,81975)--->DOUBLE(2480,82059)'
select wire 'DOUBLE(2480,82059)'
select arc 'DOUBLE(2480,82059)--->DOUBLE(9544,81975)'
select wire 'DOUBLE(9544,81975)'
select arc 'DOUBLE(9544,81975)--->DOUBLE(16456,81975)'
select wire 'DOUBLE(16456,81975)'
select arc 'DOUBLE(16456,81975)--->DOUBLE(23368,81975)'
select wire 'DOUBLE(23368,81975)'
select arc 'DOUBLE(23368,81975)--->DOUBLE(31048,82059)'
select wire 'DOUBLE(31048,82059)'
select arc 'DOUBLE(31048,82059)--->DOUBLE(38112,81975)'
select wire 'DOUBLE(38112,81975)'
select arc 'DOUBLE(38112,81975)--->DOUBLE(45024,81975)'
select wire 'DOUBLE(45024,81975)'
select arc 'DOUBLE(45024,81975)--->DOUBLE(51936,81975)'
select wire 'DOUBLE(51936,81975)'
select arc 'DOUBLE(51936,81975)--->DOUBLE(58848,81975)'
select wire 'DOUBLE(58848,81975)'
select arc 'DOUBLE(58848,81975)--->DOUBLE(63425,82056)'
select wire 'DOUBLE(63425,82056)'
select arc 'DOUBLE(63425,82056)--->DOUBLE(62152,82071)'
select wire 'DOUBLE(62152,82071)'
select arc 'DOUBLE(62152,82071)--->DOUBLE(55384,81981)'
select wire 'DOUBLE(55384,81981)'
```

```
select arc 'DOUBLE(55384,81981)--->DOUBLE(48472,81981)'
select wire 'DOUBLE(48472,81981)'
select arc 'DOUBLE(48472,81981)--->DOUBLE(41560,81981)'
select wire 'DOUBLE(41560,81981)'
select arc 'DOUBLE(41560,81981)--->DOUBLE(34648,81981)'
select wire 'DOUBLE(34648,81981)'
select arc 'DOUBLE(34648,81981)--->DOUBLE(26816,81981)'
select wire 'DOUBLE(26816,81981)'
select arc 'DOUBLE(26816,81981)--->DOUBLE(19904,81981)'
select wire 'DOUBLE(19904,81981)'
select arc 'DOUBLE(19904,81981)--->DOUBLE(12992,81981)'
select wire 'DOUBLE(12992,81981)'
select arc 'DOUBLE(12992,81981)--->DOUBLE(6080,81981)'
select wire 'DOUBLE(6080,81981)'
select arc 'DOUBLE(6080,81981)--->DOUBLE(-1752,81981)'
select wire 'DOUBLE(-1752,81981)'
select arc 'DOUBLE(-1752,81981)--->DOUBLE(-8664,81981)'
select wire 'DOUBLE(-8664,81981)'
select arc 'DOUBLE(-8664,81981)--->DOUBLE(-15576,81981)'
select wire 'DOUBLE(-15576,81981)'
select arc 'DOUBLE(-15576,81981)--->DOUBLE(-22488,81981)'
select wire 'DOUBLE(-22488,81981)'
select arc 'DOUBLE(-22488,81981)--->DOUBLE(-30320,81981)'
select wire 'DOUBLE(-30320,81981)'
select arc 'DOUBLE(-30320,81981)--->DOUBLE(-37232,81981)'
select wire 'DOUBLE(-37232,81981)'
select arc 'DOUBLE(-37232,81981)--->DOUBLE(-44144,81981)'
select wire 'DOUBLE(-44144,81981)'
select arc 'DOUBLE(-44144,81981)--->DOUBLE(-51056,81981)'
select wire 'DOUBLE(-51056,81981)'
select arc 'DOUBLE(-51056,81981)--->DOUBLE(-57968,81981)'
select wire 'DOUBLE(-57968,81981)'
route
```

LISTE DE RÉFÉRENCES BIBLIOGRAPHIQUES

Bennetts, R. G. (1984). Design of testable logic circuits. London ; Reading, Mass., Addison-Wesley Pub. Co.

Breuer, M. A. and A. D. Friedman (1976). Diagnosis & reliable design of digital systems. Woodland Hills, Calif., Computer Science Press.

Bushnell, M. L. and V. D. Agrawal (2000). Essentials of electronic testing for digital, memory, and mixed-signal VLSI circuits. Boston, Kluwer Academic.

Chakravarty, S. and P. J. Thadikaran (1997). Introduction to IDDQ testing. Boston, Kluwer Academic Publishers.

Chmelar, E. (2004). "Minimizing the number of test configurations for FPGAs." ICCAD-2004: International Conference on Computer Aided Design, Ieee/Acm Digest of Technical Papers: pages 899-902.

Chmelar, E. and S. Toutounchi (2004). "FPGA bridging fault detection and location via differential I-DDQ." 22nd IEEE VLSI Test Symposium, Proceedings: pages 109-114.

Das, D. and N. A. Touba (1999). "A low cost approach for detecting, locating, and avoiding interconnect faults in FPGA-based reconfigurable systems." VLSI Design, 1999. Proceedings. Twelfth International Conference On: pages 266-269.

FPGA Editor (2000). FPGA Editor 2.11, Xilinx Inc.

Gkatziani, M., R. Kapur, et al. (2007). "Accurately determining bridging defects from layout." Proceedings of the 2007 IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems: pages 87-90.

Harris, I. G. and R. Tessier (2002). "Testing and diagnosis of interconnect faults in cluster-based FPGA architectures." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 21(11): pages 1337-1343.

Ken Jaramillo (2000)."10 tips for successful scan design: part one" Design Feature Conference, 2000. Proceedings. 17th 2: pages 68- 71.

Krasniewski, A. (1999). "Application-dependent testing of FPGA delay faults." EUROMICRO Conference, 1999. Proceedings. 25th 1: pages 260-267.

Maxfield, C. (2004). The Design Warrior's Guide to FPGAs: Devices, Tools and Flows, Newnes

Mourad, S. and Y. Zorian (2000). Principles of testing electronic systems. New York, John Wiley & Sons.

Rajsuman, R. (2000). "Iddq testing for CMOS VLSI." Proceedings of the Ieee 88(4): pages 544-566.

Renovell, M., P. Faure, et al. (2001). "IS-FPGA : a new symmetric FPGA architecture with implicit scan." Test Conference, 2001. Proceedings. International: pages 924-931.

Renovell, M., J. M. Portal, et al. (1998). "Testing the interconnect of RAM-based FPGAs." IEEE Design & Test of Computers 15(1): pages 45-50.

Tahoori, M. B., E. J. McCluskey, et al. (2004). "A multi-configuration strategy for an application dependent testing of FPGAs." 22nd IEEE VLSI Test Symposium, Proceedings: pages 154-159.

Tahoori, M. B. and S. Mitra (2005). "Application-independent testing of FPGA interconnects." Ieee Transactions on Computer-Aided Design of Integrated Circuits and Systems 24(11): pages 1774-1783.
.

Tahoori, M. B. and S. Mitra (2006). "Test compression for FPGAs." 2006 IEEE International Test Conference, Vols 1 and 2: pages 540-548.

Thibeault, C. (1998). "Increasing current testing resolution." 1998 IEEE International Symposium on Defect and Fault Tolerance in Vlsi Systems, Proceedings: pages 126-134.

Thibeault, C., Hariri, Y (2009). «CDIDDQ: Improving Current-Based Testing and Diagnosis through Adapted Test Pattern Generation», IEEE Trans. on VLSI Systems, vol.19, no.1, janv. 2011. (CRSNG)

Wang, C. C., J. J. Liou, et al. (2005). "A BIST scheme for FPGA interconnect delay faults." 23rd IEEE VLSI Test Symposium, Proceedings: pages 201-206.

Xilinx (2003). The Programmable Logic Data book 2003, Xilinx Inc.

ZHAO, L., W. D. M. H., et al. (1998). "Bridging fault detection in FPGA interconnects using IDDQ." FPGA '98 Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays(0-89791-978-5): pages 95 - 104.

ZHAO, L., W. D. M. H., et al. (1998). "IDDQ testing of bridging faults in logic resources of reconfigurable Field Programmable Gate Arrays." IEEE Transactions on Computers 47(10): pages 1136-1152.

ZHAO, L., W. D. M. H., et al. (1999). "IDDQ Testing of Input/Output Resources of SRAM-Based FPGAs." Proceeding ATS '99 Proceedings of the 8th Asian Test Symposium: pages 375-379.