INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI®

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UNIVERSITÉ DE MONTRÉAL

ANALYSE DE TESTABILITÉ AU NIVEAU TRANSFERT DE REGISTRES

SAMIR BOUBEZARI

DÉPARTEMENT DE GÉNIE ÉLECTRIQUE
ET DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION
DU DIPLÔME DE PHILOSPHIÆ DOCTOR (Ph.D.)
(GÉNIE ÉLECTRIQUE)
MARS 1998

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée:

ANALYSE DE TESTABILITÉ AU NIVEAU TRANSFERT DE REGISTRES

présenté par: BOUBEZARI Samir

en vue de l'obtention du diplôme de: Philosophiae Doctor (Ph.D.)

a été dûment acceptée par le jury d'examen constitué de:

M. DAGENAIS Michel, Ph.D., président

Mme KAMINSKA Bozena, Ph.D., membre et directeur de recherche

M. CERNY Eduard, PhD., membre et co-directeur de recherche

M. ABRAHAM Jacob A., Ph.D., membre externe

M. ABOULHAMID ElMostapha, Ph.D., membre

À la mémoire de mon père et à ma mère,

à toute ma famille ...

Remerciements

Nous tenons à remercier ceux et celles qui, de près ou de loin, ont contribué à ce travail, en particulier: Mon directeur de recherche Mme Bozena Kaminska pour sa confiance dont elle a fait preuve à mon égard tout au long de cette recherche ainsi que pour le soutien financier qu'elle m'a accordé durant mes études. Mon codirecteur M. Eduard Cerny qui m'a guidé et motivé lors de discussions intéressantes que nous avons eues ensemble tout au long de mes études de Doctorat. Son expertise dans le domaine m'a été d'une aide inestimable. M. Benoit-Nadeau Dostie de la compagnie LogicVision, Ottawa, pour ses conseils et sa collaboration durant ce travail de recherche, M. Bechir Ayari de sa collaboration dans ce projet en tant que postdoctorat ainsi que les membres de jury qui ont accepté d'évaluer le contenu de cette thèse.

Je remercie la compagnie LogicVision à Ottawa ainsi que le GRIAO (Groupe de Recherche Interuniversitaire en Architecture des Ordinateurs) pour leurs soutiens financier durant mes études de doctorat.

Mes remerciements vont également à ma femme pour son support, ses multiples encouragements et la patience dont elle fait preuve tout au long de cette thèse. Mes remerciements vont de même à l'ensemble de ma famille pour leur soutien, leurs encouragements et leurs prières tout au long de mes études; ainsi qu'à tous ceux et celles qui ont eu de la gentillesse, aux moments critiques, de céder une station, répondre à une question, etc...

Résumé

Actuellement, la synthèse au niveau transfert de registres (Register Transfer Level - RTL) ou la synthèse logique, représente une étape importante dans le processus de conception des circuits intégrés numériques. Elle a été intégrée dans un certain nombre d'outils de Conception Assisté par Ordinateur (Computer-Aided-Design - CAD) tels que Synopsys, Cadence, Mentor Graphics, etc. La synthèse au niveau RTL est un processus d'optimisation permettant de générer à partir d'une description au niveau RTL, une description au niveau portes logiques en respectant certaines contraintes telles que la surface et la vitesse du circuit. Une description au niveau RTL consiste en un réseau de modules combinatoires (réalisant des fonctions élémentaires telles que l'addition, la multiplication, etc.) et d'éléments mémoires ou registres.

Parmi les avantages de la synthèse au niveau RTL est qu'elle permet de décrire le circuit à haut niveau d'abstraction, ce qui est nécessaire avec l'augmentation de la complexité des circuits intégrés. Aussi, le circuit généré au niveau portes logiques après la synthèse est généralement non redondant, et peut être implanté en plusieurs technologies différentes. Enfin, la description au niveau RTL est plus facile à lire, à comprendre et à analyser par comparaison à une description au niveau portes logiques.

L'évolution de la technologie des semi-conducteurs vers des niveaux d'intégration de plus en plus élevés et le progrès dans le développement des outils CAD pour la conception des circuits ont rendu le test une tâche de plus en plus complexe et coûteuse. Des techniques de conception orientée pour la testabilité (Design-For-Testability - DFT) ont été développées afin de réduire le coût du test et améliorer la qualité du test du circuit intégré.

La plupart des techniques DFT proposées dans la littérature utilisent l'analyse de testabilité pour estimer la difficulté du test d'un circuit intégré dans le but de le modifier pour le rendre facilement testable. La modification du circuit consiste généralement à insérer un ensemble de points de test. Cependant, cette modification affecte certains paramètres tels que la surface et la vitesse du circuit déjà optimisés durant le processus de la synthèse au niveau RTL. D'où l'avantage de considérer la testabilité du circuit avant la synthèse.

L'objectif de cette thèse est de considérer la testabilité d'un circuit à un haut niveau d'abstraction, avant la synthèse au niveau RTL. Nous proposons une nouvelle méthode d'analyse de testabilité et d'insertion de points de test des circuits intégrés numériques décrits au niveau RTL et spécifiés en langage VHDL. Nous supposons la méthode de balayage complet (full scan) dans un environnement de test aléatoire intégré (Built-In Self-Test -BIST).

La méthode de balayage complet associée au test aléatoire est devenue la norme dans l'industrie des dispositifs à semi-conducteurs. Cependant, la présence des pannes qui résistent au test aléatoire limite le succès de cette méthode. Deux solutions sont possibles pour résoudre ce problème. La première solution consiste à modifier les vecteurs de test générés par le générateur de vecteurs de test tandis que la deuxième solution consiste à utiliser des techniques DFT pour modifier le circuit en insérant un ensemble de points de test. Dans notre méthode, nous nous intéressons à la deuxième solution. C'est à dire, à l'analyse de testabilité et à l'insertion de points de test.

Dans la méthode proposée, la spécification VHDL du circuit est d'abord analysée et convertie en un graphe acyclique dirigé (Direct Acyclic Graph - DAG). Les noeuds internes du graphe représentent les différentes opérations et les arcs représentent les signaux et les variables de la spécification VHDL. Tous les signaux et variables sont convertis au

niveau bit ou vecteurs de bits. Les noeuds sources (puits) du graphe représentent les entrées (sorties) primaires et pseudo-primaires du circuit. Les entrées/sorties pseudo-primaires sont représentées par les registres qui sont synthétisés à partir de la spécification VHDL du circuit. Des mesures de testabilité sont ensuite définies et calculées au niveau fonctionnel pour les différentes types d'opérations VHDL sans connaître aucun détail de leur implantation au niveau portes logiques. Ceci nous permet de réduire les erreurs de calcul dues aux problèmes de reconvergence dans les circuits décrits au niveau portes logiques et aussi réduire la complexité de la construction du DAG. Les mesures de testabilité consistent à calculer la contrôlabilité et l'observabilité de chaque bit de chaque signal et variable. Certains signaux internes des modules fonctionnels (implantations des opérations VHDL telles que l'addition, la comparaison, etc.) sont aussi analysés pour déterminer leurs contrôlabilité et observabilité. Les signaux internes sont obtenus en décomposant les modules fonctionnels en sous-modules fonctionnels élémentaires.

Les valeurs de contrôlabilité (d'observabilité) sont propagées à partir des noeuds sources (puits) vers les noeuds puits (sources) du DAG afin d'identifier les bits des signaux et variables, difficiles à contrôler et/ou à observer. Un ensemble points de test est inséré dans la spécification VHDL afin d'améliorer la contrôlabilité et/ou l'observabilité de ces bits. Pour chaque point de test, on lui associe une étiquette indiquant son chemin hiérarchique de sa position au niveau du code VHDL. Au niveau VHDL, chaque point de test est défini par une fonction pour le point de contrôle, ou par une procédure pour le point d'observation. Un package a été défini incluant toutes les définitions des fonctions et procédures utilisées pour l'insertion de points de test au niveau VHDL. Cette insertion nous permet d'optimiser simultanément, la logique fonctionnelle du circuit et la logique additionnelle des points de test. Enfin, la dernière étape consiste à valider notre méthode par des résultats expérimentaux. Des circuits-test (benchmark) décrits au niveau RTL et

spécifiés en langage VHDL sont utilisés pour montrer l'efficacité de notre méthode en termes de surface, vitesse du circuit et testabilité.

Abstract

Nowadays, RTL (Register-Transfer-Level) synthesis or logic synthesis has become an integral part of a design process of digital circuits. It was integrated into several Computer-Aided-Design (CAD) tools such as Synopsys, Cadence, Mentor Graphics, etc. RTL synthesis is an optimization process which transforms an RTL description into a netlist of logic gates and registers by satisfying a set of constraints such as area and delay. An RTL description consists of an interconnection of combinational blocks (implementing functions such as addition, multiplication, etc.) and registers. Some advantages of RTL synthesis are that the design specification can be described at a high level of abstraction which is necessary with increasing design complexity, the design can be easily mapped to different technologies, the RTL description is more readable, and the gate level combinational circuits implemented by synthesis tools usually contain no redundancy.

The evolution of semiconductor technology at higher level of integration and the progress of CAD tools for designing integrated circuits have made testing a difficult and very costly task. Design-For Testability (DFT) techniques have been developed as a way to reduce the cost and improve the quality of testing of integrated circuits. Most of the DFT techniques proposed in the literature use testability analysis to estimate the difficulty of circuit testing and then the circuit is modified in order to make it easy to test. Circuit modification generally consists of inserting a set of test points. However, circuit modification affects some parameters such as area and delay which are already optimized during the RTL synthesis process. Thus, the advantage of considering testability of a circuit before the synthesis process.

The objective of this thesis to consider testability at a high level of abstraction, before RTL synthesis. We propose a new testability analysis and test point insertion method which is applicable at the RT-Level assuming full scan and pseudorandom Built-In Self-Test (BIST) environment. Full scan in combination with random patterns is widely adopted in the industry due to its ease of implementation. Unfortunately, the presence of random pattern resistant faults in many practical circuits poses a serious limitation to its success. The solutions to tackle this limitation can be broadly classified as those that modify the input patterns or those that modify the circuit-under test. In this paper, we are interested in the second class of solutions, circuit modifications, that introduce test points to improve the random pattern testability of a circuit.

The proposed approach is based on the analysis of a design specification given as a synthesizable RTL VHDL model. An intermediate representation is first obtained from the specification and then transformed in a Directed Acyclic Graph (DAG). The internal nodes of the graph correspond to operations and the edges represent signals/variables in the VHDL specification. All signals and variables are converted to bits or vectors of bits. The source (sink) nodes of the graph represent the pseudo-primary inputs (outputs) and primary inputs (outputs). The pseudo-primary inputs/outputs are given by the registers that are synthesized from the VHDL specification.

Using the DAG, Testability Measures (TMs) are defined and computed at a functional level rather than the gate level, to reduce or eliminate errors introduced by ignoring reconvergent fanout in the gate network, and to reduce the complexity of the graph construction. The TMs are the Controllability and the Observability for each bit of each signal/variable. Some internal signals of functional modules (implementation of VHDL operations) are also analyzed to determine their Controllability and Observability. Internal

signals are obtained by decomposing functional modules into elementary functional sub-modules. The Controllability (Observability) values are propagated from the source (sink) nodes to the sink (sources) nodes of the graph, in order to identify bits of signals/variables having too low Controllability or Observability. Based on the computed TM values, test point insertion is performed on each bit of each signal/variable that is considered as having too low Controllability and/or Observability. Each test point is identified by a label, that is the name of the hierarchical path and the line number in the VHDL specification. Each test point is inserted at the VHDL level by a function for a control point or a procedure for an observation point. A package has been defined to include the definitions of these functions and procedures. The test points become a part of the VHDL specification, unlike in other existing methods. This allows full use of RTL synthesis optimization concurrently on both the functional and the test logic within the design constraints such as area and delay. A number of benchmark circuits were used to demonstrate the effectiveness and the viability of the proposed method in terms of the resulting circuit area, delay, and testability.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES SYMBOLES

CAD	*Computer-Aided-Design*
RTL	*Register Transfer Level*
DFT	*Design For Testability*
BIST	*Built-In Self-Test*
DAG	*Direct Acyclic Graph*
TMs	*Testability Measures*
SHN	*Synthèse de haut niveau*
ATPG	*Automatic Test Pattern Generation*
VHDL	*Very High Speed Integration Circuit Hardware Description Language*
LFSR	*Linear Feedback Shift Register*
CA	*Cellular Automata*
COP	*Controllability Observability Program*
CDFG	*Control Data Flow Graph*
DFG	*Data Flow Graph*
CFG	*Control Flow Graph*
VIF	*VHDL Intermediate Format*
FM	*Functional Module*

LISTE DES ANNEXES

CHAPITRE 1

Introduction générale

1.1 Introduction

Les techniques modernes de conception assistée par ordinateur (Computer-Aided-design - CAD) et le progrès technologique réalisé dans le domaine de la physique des semi-conducteurs permettent aujourd'hui la réalisation de circuits intégrés numériques extrêmement complexes à base de plusieurs centaines de milliers de portes logiques. La plupart des circuits de par leur complexité, peuvent être affectés par des défauts qui sont dûs: soit à une mauvaise conception, soit à une mauvaise implantation du circuit sur le silicium. Ces défauts donnent lieu à des circuits ne fonctionnant pas du tout (cas des pannes permanentes), ou dans certains cas fonctionnant par moments et ne fonctionnant pas à d'autres moments (cas des pannes intermittentes).

La complexité croissante des circuits intégrés, associée à une augmentation du rapport entre le nombre de composants logiques et le nombre de broches, a fait augmenter de manière importante la difficulté de tester adéquatement ces circuits. Par conséquent, le test des circuits intégrés devient de plus en plus complexe. De nombreuses techniques de test ont été développées afin de mettre en évidence les différents types de défauts qui peuvent survenir dans un circuit intégré. En particulier, le test fonctionnel permettant de mettre en évidence tout défaut physique qui modifie la table de vérité du circuit lorsque celui-ci apparaît.

Dans cette thèse, nous proposons une solution originale au problème du test des circuits intégrés (nous nous intéressons au cas des circuits intégrés numériques). La suite de ce chapitre se divise en quatres parties décrivant successivement: la synthèse automatique des circuits intégrés et la formulation du problème que nous voulons résoudre (Section 1.2); les objectifs de la thèse (Section 1.3); la méthodologie de travail (section 1.4) et le plan de la thèse (section 1.5).

1.2 Synthèse automatique et analyse de testabilité

L'apparition des outils de synthèse automatique des circuits intégrés à très grande échelle, a permis de réduire de façon considérable le temps requis pour réaliser des systèmes de plus en plus complexes.

La synthèse automatique d'un circuit intégré est un processus permettant de générer à partir d'une description décrite à un certain niveau d'abstraction, une description structurelle d'une qualité acceptable, voire optimale, en respectant certaines contraintes telles que la surface résultante et la vitesse du circuit. Autrement dit, la synthèse automatique consiste surtout en un problème d'optimisation. Une description du circuit peut être raffinée manuellement, cependant, la qualité du résultat est fortement reliée à l'expérience du concepteur et au temps alloué à cette tâche. Dans la littérature, la synthèse automatique, est aussi nommée compilation du matériel par analogie avec le logiciel. Elle permet de réduire de beaucoup le temps de conception tout en générant des solutions d'une qualité supérieure, dans certains cas, à celles obtenues manuellement.

Les étapes principales de la synthèse d'un circuit intégré sont décrites dans la figure 1.1. La synthèse de haut niveau (SHN), appelée aussi synthèse comportementale, consiste à transformer une spécification d'un niveau comportemental à un niveau plus détaillé

transfert de registres (Register Transfer Level - RTL). Une description au niveau RTL est une description dans laquelle le circuit est décrit sous forme d'un chemin de données et d'un contrôleur. Le chemin de données est composé de modules fonctionnels combinatoires et de registres. Les modules fonctionnels réalisent des fonctions telles que l'addition, la soustraction, la multiplication, etc. Le contrôleur implanté sous forme d'une machine à états finis est utilisé pour synchroniser les opérations des modules fonctionnels et les transferts de données. La synthèse au niveau RTL ou la synthèse logique transforme une description au niveau RTL en un réseau optimisé d'éléments combinatoires (portes logiques) et d'éléments séquentiels (bascules élémentaires). Finalement, dépendamment de la technologie utilisée, les générateurs de dessins de masques ou d'outils de placement et routage permettent de générer des descriptions au niveau de dessins de masques du circuit en vue de sa fabrication. En passant d'une description comportementale d'un circuit jusqu'à sa fabrication, on traverse différents niveau d'abstraction (voir la figure 1.1). Les niveaux comportementaux et RTL sont indépendants de la technologie de fabrication du circuit. Ceci constitue un avantage important de l'utilisation des outils de SHN et au niveau RTL. En effet, le niveau de description du circuit généré après l'étape de la SHN et la synthèse au niveau RTL sont indépendants de la technologie avec laquelle le circuit sera fabriqué.

La synthèse au niveau RTL a été intégrée dans un certain nombre d'outils CAD tels que Synopsys, Cadence, Mentor Graphics, etc. Actuellement, elle est devenue une étape importante dans le processus de conception des circuits intégrés.

Cependant, une conséquence directe de l'évolution de la technologie des semi-conducteurs et des progrès dans le développement des outils CAD pour la conception est la difficulté d'assurer un test adéquat pour des circuits intégrés. En effet, la technologie des

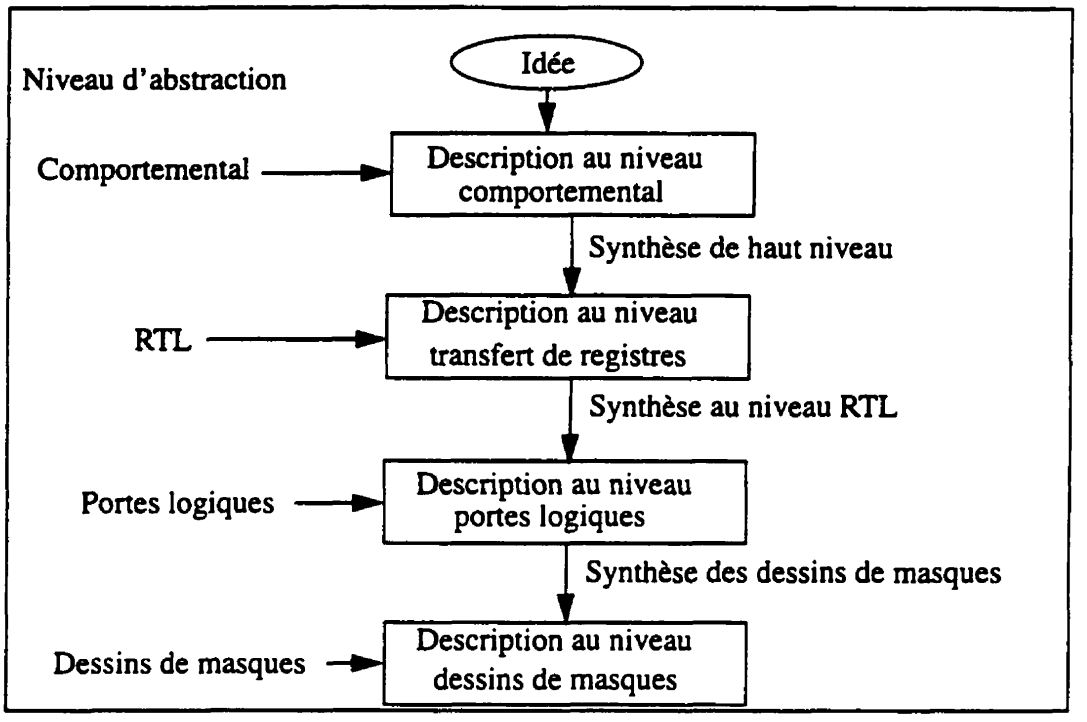

Figure 1.1 Étapes de la synthèse d'un circuit intégré.

semi-conducteurs a rendu possible l'intégration de plusieurs centaines de milliers de portes dans la même puce [101]. Entre temps, le nombre des broches de circuits n'a augmenté que comme une fonction de la racine carrée du nombre de nombre de portes (règle de Rent) [93]. Cette situation rend de plus en plus complexe le problème du test des circuits intégrés modernes.

La testabilité d'un circuit est déterminée par le temps de développement du test d'un produit. Le développement d'un test pour un circuit intégré consiste principalement à accomplir les trois tâches suivantes:

1. La génération des ensembles de vecteurs de test;
2. L'application de ces ensembles et l'observation de la réponse du circuit;
3. La comparaison de la réponse reçue avec la réponse attendue (réponse du bon circuit).

Les stratégies de test présentement disponibles sont essentiellement destinées à réduire la complexité des trois tâches mentionnées ci-haut. En général, les solutions et les techniques de test développées jusqu'à présent émergent principalement de deux visions complémentaires:

1. Développer des techniques de test de plus en plus sophistiquées, sans avoir à changer la structure du circuit à tester;
2. Agir sur la structure du circuit pour le rendre plus facile à tester.

En réalité, ces deux visions ne sont pas contradictoires. Au contraire, tout développement dans l'une, peut affecter l'autre. Les techniques de génération de vecteurs de test avec les outils automatiques de génération de vecteurs de test (Automatic Test Pattern Generation - ATPG) et la génération de vecteurs de test pondérés, par exemples, s'inscrivent dans le cadre de la première vision. Durant plusieurs années, les chercheurs ont mis sur pied un ensemble de techniques de génération de vecteurs de test qui deviennent actuellement des outils importants dans les méthodologies de conception des circuits intégrés modernes. Par ailleurs, avec des niveaux d'intégration de plus en plus élevés, il est souvent très difficile de résoudre les problèmes de test exclusivement avec des techniques de génération de vecteurs de test. De nombreux chercheurs ont ainsi proposé d'agir sur la structure du circuit lui-même pour en faciliter le test. C'est l'idée principale des techniques dites de conception orientée pour la testabilité (Design-For-Testability - DFT). Généralement, ces technique se basent sur l'analyse de testabilité du circuit dans le but de le modifier pour le rendre facilement testable.

L'analyse de testabilité est un processus qui quantifie les problèmes de test dans un circuit donné. L'idée principale de l'analyse de testabilité est l'utilisation d'un ensemble de mesures faciles à calculer. Ceci nous permet, tout en évitant la complexité des méthodes exactes, d'estimer la difficulté de tester un circuit donné et de localiser les sites et la

nature du problème de test. L'analyse de la testabilité est essentiellement basée sur deux notions principales qui reflètent la difficulté de tester une panne [26, 53] à savoir:

1. La contrôlabilité, et
2. L'observabilité.

La contrôlabilité est une mesure de la difficulté à contrôler un noeud du circuit à partir des entrées primaires pour faire apparaître (sur ce même noeud) l'effet de la panne. L'observabilité quant à elle est une mesure de la difficulté à observer cet effet sur les sorties primaires. De ce point de vue, la technique DFT peut être considérée comme une discipline destinée à améliorer la contrôlabilité et l'observabilité. Une de ces techniques est l'insertion de points de test. Un point de test consiste en une logique additionnelle ajoutée au circuit sous-test permettant d'améliorer la contrôlabilité et l'observabilité d'un noeud donné.

Généralement, l'analyse de testabilité et l'insertion de points de test est souvent considérée après la synthèse du circuit au niveau portes logiques, c'est à dire après l'étape de la synthèse au niveau RTL. En plus, l'utilisation des outils de synthèse automatique au niveau RTL a rendu difficile, voire complexe et moins bénéfique l'analyse de testabilité et l'insertion de points de test au niveau portes logiques et cela pour plusieurs raisons: Premièrement, une modification du circuit après la synthèse peut dégrader les performances du circuit déjà optimisées durant le processus de synthèse telles que la surface résultante et la vitesse du circuit. Deuxièmment, il suffit de modifier les contraintes d'optimisation pour obtenir une autre structure du circuit, qui réalise la même fonction. Par conséquent, une autre analyse de testabilité est nécessaire. Enfin, la complexité des circuits intégrés au niveau portes logiques augmente la complexité d'analyse de testabilité et d'insertion de points de test.

On peut donc déduire que la description du circuit au niveau portes logiques n'est pas toujours une étape appropriée pour considérer l'analyse de testabilité et l'insertion de points de test d'un circuit intégré généré par les outils de synthèse automatique. En effet, considérer l'analyse de testabilité et l'insertion de points de test avant le processus de synthèse, permet d'optimiser le circuit avec d'autres contraintes de synthèse. De cette manière, l'analyse de testabilité devient une partie de la spécification du circuit avant la synthèse et indépendante de la technologie de fabrication. Ainsi, le but de notre thèse se résume:

La proposition d'une nouvelle méthode d'analyse de la testabilité et d'insertion de points de test à une étape avancée du processus de conception des circuits intégrés, avant la synthèse au niveau RTL.

1.3 Objectifs de la thèse

Les objectifs de cette thèse sont l'analyse de testabilité et l'insertion de points de test dans les circuits intégrés décrits au niveau RTL et spécifiés en langage VHDL. On suppose la technique de balayage complet (full scan) dans un environnement de test aléatoire intégré (Built-In Self Test - BIST). Le but principal est de générer des circuits au niveau portes logiques par les outils de synthèse au niveau RTL, facilement testables dans un environnement de test aléatoire avec la méthode BIST en supposant la méthode de balayage complet. Un outil logiciel a été développé en langage C d'une complexité d'environ 14,000 lignes de code.

1.4 Méthodologie de travail

Afin d'atteindre nos objectifs de notre méthode, nous avons procédé selon les étapes suivantes:

• **Étape 1**: *Analyse et conversion de la spécification VHDL*

Dans cette étape, la spécification VHDL du circuit est analysée et convertie en un graphe acyclique dirigé (Direct Acyclic Graph - DAG). Les noeuds internes du graphe représentent les opérations (arithmétiques, logiques, relationnelles et de transfert de données), et les arcs du graphe représentent les signaux et les variables qui sont déclarés ou peuvent être induits dans la spécification VHDL. Les noeuds sources (puits) du graphe représentent les entrées (sorties) primaires et pseudo-primaires du circuit. Les entrées/sorties pseudo-primaires sont représentées par les registres qui sont synthétisés de la spécification VHDL du circuit. Tous les signaux et variables sont converties au niveau bit ou vecteur de bits.

• **Étape 2**: *Calcul et propagation des mesures de testabilité*

Des mesures de testabilité sont définies et calculées pour chaque bit de chaque signal/variable de la spécification VHDL. Ces mesures consistent à calculer la contrôlabilité à 0 et à 1, et l'observabilité de chaque bit de chaque signal/variable. Ces mesures sont calculées au niveau fonctionnel pour les différents types d'opérations VHDL (addition, comparaison, etc.) sans connaître aucun détail de leur implantation au niveau portes logiques. Ces mesures de testabilité sont aussi calculées pour certains signaux internes des modules fonctionnels (implantation des opérations VHDL) dans la spécification VHDL. Les valeurs de contrôlabilité (observabilité) sont propagées à partir des entrées (sorties) primaires et pseudo-primaires vers les sorties (entrées) primaires et pseudo-primaires afin d'affecter les valeurs de contrôlabilité à 0 et à 1, et de l'observabilité à chaque bit des signaux et des variables de la spécification.

• **Étape 3**: *Sélection des points test*

En utilisant les résultats du calcul des mesures de testabilité de l'étape 2, on identifie le(s) bit(s) de chaque signal/variable qui sont difficiles à contrôler ou à observer. À chaque point de test, on associe une étiquette indiquant son chemin hiérarchique de sa position au niveau du code VHDL original.

- **Étape 4**: *Insertion des points de test au niveau du VHDL*

On insère les points de test sélectionnés durant l'étape 3 dans la spécification VHDL. Chaque point de test consiste en un point de contrôle ou d'observation. Il est défini par une fonction (point de contrôle) ou par une procédure (point d'observation) en langage VHDL synthétisable. À cet effet, un package a été défini incluant toutes les définitions des fonctions et procédures utilisées pour l'insertion de points de test au niveau VHDL. Des circuits-test spécifiés en langage VHDL synthétisable sont utilisés pour valider notre méthode. On utilise les outils de Synopsys pour synthétiser les circuits au niveau portes logiques. Certains paramètres sont alors notés tels que la surface résultante, la vitesse du circuit et la couverture de pannes correspondante à l'application d'une séquence de vecteurs de test aléatoires.

L'originalité de la présente thèse réside premièrement dans l'analyse de la testabilité d'une description au niveau RTL d'un circuit spécifié en langage VHDL dans un environnement de test aléatoire et de BIST en supposant la méthode de balayage complet. On identifie le(s) bit(s) des signaux et variables de la spécification VHDL qui sont difficiles à contrôler et/ou à observer. Des signaux internes des modules fonctionnels (additionneurs, comparateurs, etc.) qui ne sont pas accessibles dans la spécification VHDL sont aussi analysés pour déterminer leurs contrôlabilité et observabilité. Deuxièmment, un certain nombre de points de test décrits en code VHDL synthétisable sont insérés dans la spécification originale du circuit afin d'améliorer la testabilité aléatoire du circuit résultant après la synthèse. Troisièmment, les points de test sont inclus dans la spécification VHDL originale du circuit indépendamment de la technologie d'implantation. De cette manière, on applique

toutes les étapes de la synthèse au niveau RTL à une spécification VHDL incluant un ensemble de points de test. Cela permet en effet, d'optimiser simultanément et la logique fonctionnelle du circuit original et la logique additionnelle des points de test insérés en fonction de toutes les contraintes du concepteur (surface, vitesse du circuit, etc.). En effet, les performances du circuit peuvent ne pas changer avec ou sans l'insertion de points de test. Enfin, un outil logiciel codé en C a été développé pour implémenter toutes les étapes de notre méthode.

1.5 Plan de la thèse

Le reste de la présente thèse est organisé comme suit:

- Le chapitre 2 présente quelques définitions de base et une revue de littérature des méthodes qui ont abordé le problème d'analyse de testabilité à haut niveau d'abstraction.
- Le chapitre 3 présente la méthode d'analyse de testabilité et d'insertion de points de test dans les circuits décrits au niveau RTL spécifiés en langage VHDL. Cette méthode utilise les mesures de testabilité présentées en annexe 1.
- Le chapitre 4 décrit l'implantation de l'outil logiciel et présente des résultats expérimentaux où certains exemples illustrent l'insertion de points de test au niveau VHDL.
- Le chapitre 5 conclut la présente thèse en évoquant les travaux futurs.

CHAPITRE 2

Définitions et revue de littérature

2.1 Introduction

Nous avons formulé en introduction, la difficulté et la complexité d'analyse de la testabilité au niveau de portes logiques, c'est à dire, après la synthèse du circuit. Un certain nombre de méthodes ont été proposées dans la littérature considérant l'analyse de testabilité à un haut niveau d'abstraction de la description du circuit dans le but de générer des circuits facilement testables. Plusieurs définitions du mot "testable" ont été utilisées par les chercheurs dans le domaine de la synthèse [102]. Un circuit qui est évalué avec un outil ATPG (séquentiel ou combinatoire) est considéré comme facilement testable lorsque le coût de la génération de vecteurs de test est faible (nombre de vecteurs de test faible, couverture de pannes élevée, etc.). L'augmentation de la contrôlabilité et de l'observabilité des noeuds d'un circuit permettent de rendre le circuit facilement testable par les outils ATPG. Une autre manière de générer un circuit facilement testable est de synthétiser un circuit qui peut être testé par la méthode BIST. Les méthodes proposées dans la littérature diffèrent l'une de l'autre par l'objectif du test ciblé à savoir: la génération des circuits facilement testables par les outils ATPG (séquentiel ou combinatoire) ou la génération des circuits facilement testables par la méthode du BIST [62].

Dans le reste de ce chapitre, nous abordons brièvement les notions de base des techniques de test des circuits intégrés. Par la suite, nous présentons des méthodes qui ont été

proposées dans la littérature concernant l'analyse de testabilité et l'insertion de points de test à un haut niveau d'abstraction (le niveau RTL et comportemental).

2.2 Les définitions de base

Dans cette section, nous introduisons le principe de modélisation de pannes ainsi que le modèle considéré dans notre méthode. Ensuite, nous introduisons les mesures de testabilité approximatives. Ces mesures ont une grande importance dans les techniques de génération de vecteurs de test, ainsi que dans les techniques de DFT. Enfin, nous abordons brièvement quelques techniques de base de DFT et du BIST.

2.2.1 La modélisation de pannes

Les dispositifs semi-conducteurs peuvent être sujets à différentes défectuosités. Ces défectuosités peuvent être dues à des erreurs de conception (c.a.d., spécifications incomplètes, inconsistantes ou violations des règles de conception) comme elles peuvent être dues aussi à des défectuosités de fabrication ou à des défaillances physiques. Les défectuosités de fabrication ne sont pas directement attribuables à une erreur humaine, elles résultent plutôt d'une imperfection dans le processus de fabrication. On peut citer l'exemple des courts circuits et des circuits ouverts qui sont des défectuosités de fabrication très fréquentes. En général, les défectuosités ne permettent pas une modélisation facile sur laquelle on pourrait baser le développement des tests. La modélisation par des modèles de pannes logiques s'est donc imposée comme un moyen convenable pour faciliter le développement des tests. En effet, avec un modèle de panne logique, le problème d'analyse des défectuosités et des défaillances devient un problème logique plutôt que physique. La complexité de la génération des vecteurs de test est alors réduite, puisque plusieurs défectuosités différentes peuvent être modélisées par une même panne. De plus, dans certains

cas, le même modèle peut être applicable à plusieurs technologies différentes, rendant ainsi le problème du test des circuits intégrés indépendant de la technologie.

En général, les modèles de pannes logiques supposent que tous les composants fonctionnent correctement, autrement dit, ils ne contiennent aucune défectuosité, et que seules leurs interconnexions peuvent être affectées. Les pannes typiques affectant les interconnexions sont les courts circuits et les circuits ouverts. Un court circuit est formé lorsque se connectent des lignes qui ne sont pas supposées l'être, tandis qu'un circuit ouvert résulte d'une connexion brisée.

Par exemple, dans plusieurs technologies, un court-circuit entre la masse ou l'alimentation et une ligne quelconque du circuit peut amener la ligne à prendre une valeur logique fixe. Autrement dit, la ligne est collée à une valeur logique 0 ou 1.

Depuis quelques décennies, plusieurs modèles de pannes ont vu le jour. Le modèle "collé-à" (stuck-at) est de loin le plus utilisé malgré ses limites. Pour ce modèle, chaque défectuosité se manifeste au niveau de la structure, par une ligne "collé-à 1 ou à 0". De plus, dans la plupart des approches du test, une seule panne à la fois est supposée être présente dans le circuit, lors du développement du test. Ceci est généralement nécessaire pour éviter une explosion de la complexité de la génération des vecteurs de test. Un test pour un circuit donné est considéré comme étant bon s'il couvre la majeure partie des pannes "collé-à" simples. Dans le présent document, nous considérons le modèle "collé-à" simple pour des pannes uniques, car il a été démontré suffisant par la pratique.

2.2.2 Les mesures de testabilité approximatives

Il existe dans la littérature deux grandes catégories de mesures de testabilité:

1. Les mesures de testabilité déterministes, et

2. Les mesures de testabilité probabilistes.

Dans la première catégorie, nous retrouvons les travaux pionniers de Ruthman [109], Stephenson et Grason [131], et Breuer [24]. Ces travaux ont culminé, peu de temps après, par des outils de calcul des mesures de testabilité les plus populaires: SCOAP [53, 54]. Plus récemment, d'autres méthodes aussi basées sur SCOAP, ont vu le jour [104, 20].

La procédure de calcul des nombres SCOAP est basée sur le rang logique des noeuds du circuit. En partant des entrées primaires se dirigeant vers les sorties primaires, les contrôlabilités sont calculées selon le rang de chaque noeud. Une fois que toutes les contrôlabilités sont calculées, la procédure de calcul des nombres SCOAP commence le calcul des observabilités, en partant des sorties primaires pour aller vers les entrées primaires. Pour de plus amples détails sur le calcul des nombres SCOAP, veuillez consulter les références spécialisées [53, 54].

Avec les mesures de testabilité probabilistes, on s'intéresse plutôt à la proportion des vecteurs d'entrées qui peuvent détecter une panne. Autrement dit, les mesures de testabilité probabilistes, comme leur nom l'indique, donnent une estimation de la probabilité de détection d'une panne. Les techniques de calcul des mesures de testabilité probabilistes que l'on retrouve dans la littérature peuvent être réparties en deux groupes.

1. Les techniques probabilistes: basées sur l'étude probabiliste du circuit sans considération des vecteurs d'entrée;
2. Les techniques statistiques: basées sur des simulations logiques d'un ensemble fini de vecteurs de test.

Une étude comparative des différentes techniques de calcul des mesures de testabilité fut publiée dans [61]. Les techniques statistiques semblent être efficaces si nous assurons un nombre de vecteurs de test adéquat. Malheureusement, les simulations logiques, qu'utilisent les techniques statistiques, sont relativement coûteuses et chaque vecteur de

test additionnel contribue à augmenter ces coûts. De plus, la précision de calcul dépend fortement du nombre d'échantillons (vecteurs) utilisés.

Les techniques probabilistes [44, 64, 116] furent développées pour les circuits combinatoires. Par exemple, les nombres COP [27, 28], a été une des premières méthodes proposées utilisant la technique probabiliste.

Les nombres COP consistent principalement en trois nombres qui peuvent être calculés pour chaque noeud du circuit. De la même manière qu'avec la procédure de calcul des nombres SCOAP, la procédure de calcul des nombres COP traverse le circuit en partant des entrées primaires vers les sorties primaires pour calculer d'abord les contrôlabilités de tous les noeuds du circuit. Pour chaque type de porte, nous possédons une formule qui détermine les contrôlabilités de sa sortie en fonction de celles de ses entrées. Les observabilités sont ensuite calculées en allant vers les entrées primaires à partir des sorties primaires. De même, pour chaque type de porte, il existe une formule, qui pour chacune des entrées, donne son observabilité en fonction de l'observabilité de sa sortie et des contrôlabilités des autres entrées (de la même porte). Les valeurs des contrôlabilités (observabilités) des entrées (sorties) primaires sont initialisées à 0.5 (1).

La précision des mesures de testabilité a fait l'objet de plusieurs publications. Ainsi, dans [10, 88], les auteurs ont montré le caractère approximatif des nombres SCOAP. De même pour les mesures probabilistes, des chercheurs [61, 117] ont montré que la dépendance est la source d'erreur, car la méthode suppose l'indépendance des entrées d'une même porte. Ainsi, en présence des sortances reconvergeantes, ces mesures peuvent complètement échapper certains sites de problèmes de test. Par contre, ces hypothèses s'avèrent très utiles pour garder la complexité du calcul des mesures COP linéaire en fonction du nombre de portes d'un circuit. Les mesures de testabilité que nous avons défini dans

notre travail sont basées sur une méthode approximative probabiliste. Elles s'inspirent de la méthode de COP pour définir des mesures de testabilité au niveau RTL sans connaître aucun détail de l'implantation au niveau portes de modules fonctionnels (additionneurs, comparateurs, etc.) D'où une réduction des erreurs dues au problème de reconvergence qui existe dans les réseaux de portes logiques. Cependant, le problème de reconvergence demeure au niveau RTL entre les interconnexions des modules fonctionnels. L'annexe I constitue notre contribution aux mesures de testabilité probabilistes développées au niveau RTL.

2.2.3 Conception orientée pour la testabilité

Les techniques de conception orientée pour la testabilité (Design-For-Testability - DFT) sont des techniques qui visent à tenir compte très tôt des problèmes de test durant le processus de conception. Avec l'avènement des outils de synthèse, plusieurs techniques pour faciliter le test du produit synthétisé ont vu le jour.

Les techniques de DFT qui existent dans la littérature peuvent être divisées en deux groupes:

1. Les techniques structurées, et
2. Les techniques ad hoc.

2.2.3.1 Les techniques de test structurées

Si nous faisons abstraction des techniques qui visent à pallier aux limites de précision des modèles de pannes, les techniques de test structurées développées récemment sont essentiellement destinées aux circuits séquentiels. En effet, l'état de l'art des techniques de test des circuits combinatoires est relativement mature [33]. Cependant, ceci n'est pas le cas avec les circuits séquentiels où le problème de test est relié à l'existence des élé-

ments de mémoire et des lignes de rétroaction [4]. En effet, pour tester un circuit séquentiel, nous devons:

1. Initialiser le circuit en contrôlant les éléments de mémoire; sans l'initialisation, la réponse du circuit ne peut être prédite avec certitude, et
2. Appliquer une séquence de plusieurs vecteurs de test.

Balayage complet. Il y a plus de deux décennies, plusieurs chercheurs [13, 43, 47] ont proposé de convertir le problème de test des circuits séquentiels en un problème de test des circuits combinatoires. L'idée principale sous-jacente à ces techniques est de relier les éléments de mémoires en forme de registre à décalage (appelée la chaîne de balayage) durant le mode test. De cette manière, tous les éléments de mémoire deviennent directement contrôlables et observables, à travers la chaîne de balayage.

Ces techniques, connues sous le nom de balayage complet (full scan), sont très répandues. Actuellement, l'usage de la technique de la chaîne de balayage complet est devenue la norme dans l'industrie des dispositifs à semi-conducteurs.

Balayage partiel. Avec ces techniques, seule une partie des éléments de mémoire est sélectionnée pour former une chaîne de balayage. Ainsi, la circuiterie ajoutée est réduite, la dégradation des performances peut être améliorée en évitant les chemins critiques, et finalement le temps d'application de chaque vecteur de test peut être réduit.

En réalité, une comparaison des techniques de balayage partiel avec celles du balayage complet n'est pas aussi simple. Plusieurs auteurs remettent en cause les bienfaits du balayage partiel. En effet, le problème de la génération algorithmique demeure dans ce cas, essentiellement séquentiel. Par conséquent, le gain au niveau de la circuiterie, ou au niveau du temps d'application, peut facilement être annulé ou même devenir une perte nette à cause de la complexité de la génération algorithmique et de la perte en couverture de pannes.

Un problème relié au balayage partiel consiste à choisir parmi un grand nombre d'éléments de mémoire d'un circuit, un sous-ensemble qui sera inclus dans une chaîne de balayage. Ceci a pour but de garantir un certain niveau de test (exprimé souvent en pourcentage de pannes détectées) tout en minimisant les inconvénients qui en résultent. Le balayage partiel a fait l'objet d'un très grand nombre de publications comme en témoigne la liste bibliographique ci-jointe [8, 9].

2.2.3.2 Insertion de points de test

Une approche Ad Hoc directe pour améliorer la testabilité est l'insertion de points de test. Historiquement, cette approche fut principalement développée pour des circuits combinatoires [73, 112, 119]. Un point de test consiste en une logique additionnelle ajoutée à un noeud du circuit permettant d'améliorer la contrôlabilité et l'observabilité de ce noeud.

Une formulation du problème d'insertion des points de test consiste à choisir un petit ensemble de points (noeuds du circuit) pour insérer des structures logiques additionnelles (voir la figure 2.1), tout en:

1. Minimisant le nombre de ports d'entrée/sortie additionnels;
2. Minimisant l'augmentation de la surface globale occupée par la circuiterie ajoutée;
3. Limitant les dégradations de la performance;
4. Facilitant le développement du test.

Notons que les quatre points mentionnés ci-haut ne sont pas nécessairement de même priorité. En effet, l'objectif principal de toutes les techniques d'insertion de points de test est de garantir avant tout un bon niveau de testabilité (exprimé généralement en la couverture de pannes). Evidemment, ceci doit s'effectuer tout en respectant le budget alloué en termes de la surface additionnelle permise, de nombre de ports supplémentaires possibles et de la dégradation de performance tolérée.

Nous pouvons distinguer deux catégories de points de test, et par conséquent deux sortes de circuits logiques additionnels pour résoudre deux sortes de problèmes de test différents. Les deux catégories sont:

1. Les points de contrôle qui sont destinés à améliorer la contrôlabilité des noeuds internes du circuit et ainsi résoudre ce qu'on appelle le problème de contrôle;
2. Les points d'observation dont le rôle est d'améliorer l'observabilité des noeuds internes du circuit et ainsi régler le problème connu sous le nom de problème d'observabilité.

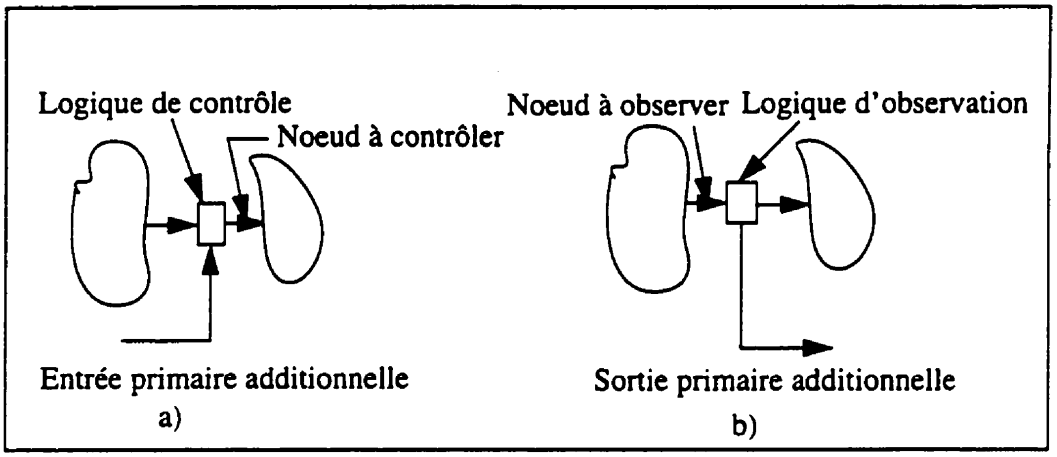

Figure 2.1 Les deux catégories de points de test a) point de contrôle b) point d'observation.

Dans les travaux reliés à l'insertion des points de test, nous pouvons distinguer quatre notions fondamentales:

1. La structure des points de test, autrement dit, la forme de la logique utilisée pour l'introduction du point de test;
2. L'emplacement de ces points dans le réseau des portes d'un circuit donné.

Structure des points de test. L'importance des structures de points de test découle évidemment du fait de vouloir réduire les pénalités encourues par l'insertion, tout en garantissant un bon niveau de testabilité. En effet, à chaque insertion de point de test peut

correspondre une "circuiterie ajoutée" et une dégradation des performances. Parmi les structures proposées dans la littérature, nous citons sans être exhaustif:

1. Les points de test classiques: Une porte ET est utilisée pour contrôler un noeud à 0, une porte OU pour le contrôler à 1 et une ligne vers l'extérieur pour l'observer (voir la figure 2.2);
2. Les portes XOR;

Les points de test classiques sont largement utilisés. Cependant, ce genre de points est généralement associé à une augmentation de la circuiterie ajoutée et à une dégradation non négligeables de la vitesse du circuit. De plus, l'insertion d'un point classique coupe une ligne en deux parties, la contrôlabilité de la partie reliée à la sortie de la porte insérée est améliorée, par contre, celle de la partie reliée à l'entrée de la porte demeure aussi mauvaise qu'avant l'insertion.

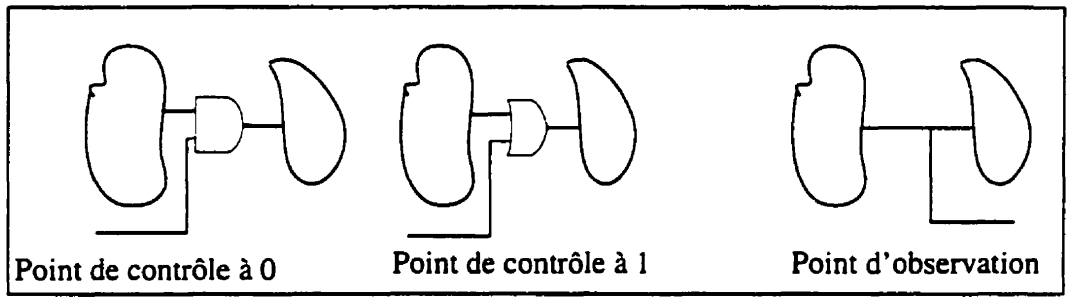

Figure 2.2 Les points de test classiques

Les portes XOR possèdent des caractéristiques très appréciées comme points de contrôle. En effet, les portes XOR sont de bons régulateurs de contrôlabilité: quelque soit la contrôlabilité d'une ligne, l'insertion d'une porte XOR améliore de façon équilibrée ses contrôlabilités à 0 et à 1. Plus encore, il n'y a pas généralement de dégradation de l'observabilité du cône d'entrée de cette ligne. Par exemple, selon la mesure COP, l'observabilité d'une entrée d'une porte XOR est égale à l'observabilité de sa sortie.

Sélection des points de test. Le problème de la sélection des points de test est reconnu comme étant un problème NP-complet (pour les circuits avec reconvergences) [63]. Dans la littérature, nous pouvons énumérer trois catégories de techniques de sélection:

1. Les techniques de sélection pour simplifier la génération algorithmique [56, 60];
2. L'insertion de points de test pour "segmenter" ou "partitionner" un circuit, dans le but de faciliter son test pseudo-exhaustif;
3. L'insertion de points de test pour améliorer le test aléatoire [25, 66, 134].

Dans la première catégorie, le but de l'insertion est de faciliter la génération algorithmique. Durant la dernière décennie, plusieurs travaux de cette catégorie furent proposés. Nous citons ici le travail de Gundlach [56], où l'insertion consiste à trouver un petit ensemble de points pour couper les rétroactions et réduire les reconvergences. Ainsi, la génération algorithmique est améliorée du moment où les deux facteurs qui influencent sa complexité (rétroactions et reconvergences) sont éliminés.

La deuxième catégorie englobe toutes les techniques destinées à rendre pratique le test exhaustif.

La dernière catégorie est destinée à améliorer le test aléatoire qui trouve évidemment une application directe dans les techniques du test intégré (BIST). Dans cette catégorie, nous pouvons distinguer deux grandes sous-catégories:

1. Les techniques basées sur la simulation de pannes [25, 63];
2. Les techniques basées sur les mesures de testabilité [119, 132, 145];

Dans la première sous-catégorie, nous pouvons citer le travail de Briers et c^{ie} [25]. Dans leur article, ils ont proposé une heuristique de placement pour éliminer les pannes résistantes au test aléatoire. Un second travail dans cette catégorie fut proposé par Iyengar et Brand [63]. Avec leur heuristique, une simulation de panne est invoquée pour collecter des informations sur les problèmes de propagation que rencontrent les pannes résistantes

au test aléatoire dans un réseau de portes. Les points de contrôle sont choisis selon leur contribution à améliorer la propagation de l'effet de ces pannes. Une deuxième simulation est ensuite invoquée pour déterminer l'ensemble des points d'observation.

Evidemment, ces techniques sont relativement coûteuses, car elles se basent sur la simulation de pannes. De plus, la simulation de pannes est étroitement reliée à la séquence de vecteurs de test utilisée. Bien entendu, ceci limite la précision de ce genre de techniques.

Pour éliminer le recours à la simulation de pannes, plusieurs auteurs ont proposé des méthodes heuristiques basées sur les mesures de testabilité. Cependant, malgré cette imprécision, ces techniques ont prouvé leur efficacité et elles sont actuellement largement utilisées, particulièrement dans le test des circuits combinatoires. Par exemple, Youssef et al. [145] ont proposé des heuristiques de placement qu'ils ont implantées dans un outil de conception pour le test aléatoire. Leurs heuristiques sont basées sur les mesures COP, les notions de secteurs de pannes ainsi que la corrélation entre les différents points de test. Un ensemble de pannes résistantes au test aléatoire est déterminé en premier lieu par le biais des mesures COP. Ces pannes sont ensuite regroupés dans des secteurs sous forme de cônes, dits secteurs de pannes. L'insertion s'effectue au niveau des sommets de ces secteurs seulement. Seiss et al. [119] ont proposé de leur coté une technique similaire basée sur une fonction de coût qui reflète la testabilité aléatoire des circuits combinatoires. Récemment, Tamarapalli et Rajski [132] ont publié un article sur une technique d'insertion de points de test dite multiphase. Dans chaque phase, un sous-ensemble de pannes est considéré. Les points de contrôle de chaque phase sont activés par des valeurs fixes.

2.2.4 Conception pour test intégré "Built-In Self-Test"

Avec la conception pour le test intégré, un circuit, une puce, une plaque ou un système peut se tester (lui-même) sans aucun apport de l'extérieur. Ceci a pour objectif:

1. Réduire le besoin d'avoir des équipements de test coûteux (un million de $ pour un testeur est un prix courant);
2. De réduire le volume des données (vecteurs de test) qui doit être manipulé par les équipements de test;
3. De réduire la complexité de la génération et de l'application des ensembles de vecteurs de test;
4. La réutilisation d'un test (test reuse) développé à un ou des niveaux d'intégration supérieurs du système (système, carte, circuit, etc.).

En général, les approches BIST peuvent être caractérisées par la nature de la génération utilisée et par la forme du test appliqué. Ainsi, on parle de BIST algorithmique versus aléatoire et de BIST exhaustif versus non exhaustif.

Le test exhaustif garantit toujours 100% de couverture des pannes "collé-à" simples. Les techniques du test exhaustif n'ont besoin ni d'un modèle de pannes, ni d'une simulation de pannes pour valider les résultats. Cependant, le temps requis pour générer tous les vecteurs de test rend cette technique impraticable pour des circuits de grande taille.

Dans le but de réduire la complexité des techniques de test exhaustif, de même que d'en préserver les avantages, plus spécifiquement 100% (ou presque) de couverture de pannes, plusieurs auteurs ont proposé des approches hybrides dites pseudo-exhaustives. Ces techniques consistent généralement à partitionner un circuit donné sous forme de sous-blocs, de manière à réduire le nombre des entrées de cônes d'entrée des sorties primaires.

Avec le test aléatoire, un ensemble de vecteurs aléatoirement générés est utilisé comme ensemble de vecteurs de test. À l'opposé des techniques exhaustives, le test aléa-

toire a l'avantage d'être réalisable pour des circuits combinatoires ou séquentiels pratiques. Souvent, des circuits très simples (LFSR[1], CA[2]) sont utilisés comme générateurs aléatoires. Malheureusement, l'estimation de la couverture de pannes et la qualité des tests demeurent encore problématiques avec les techniques aléatoires. En effet, les séquences de vecteurs de test aléatoires sont généralement plus longs que ceux obtenus par une génération algorithmique. Par conséquent, estimer la couverture de pannes par le biais de la simulation de pannes, comme nous le faisons habituellement, peut être coûteux. Dans le but de surmonter ce problème, des techniques analytiques furent proposées dans la littérature. Ces techniques ne sont que des approximations et aucune d'entre elles n'est en mesure de donner une valeur exacte de la couverture de pannes. Le deuxième problème qui est probablement le plus important, est d'amener la couverture de pannes à un niveau acceptable. En effet, des études ont montré que la couverture de pannes atteinte n'est souvent pas acceptable. Les chercheurs ont observé dans plusieurs circuits des pannes difficiles à détecter avec des vecteurs aléatoires. Ce genre de pannes est connu sous le nom de pannes résistantes au test aléatoire. Un exemple simple de ce type de pannes est la sortie collée à 0 d'une porte ET avec "n" entrées.

Plusieurs approches furent proposées dans la littérature pour faire face au problème des pannes résistantes. Ces approches peuvent être classifiées en trois groupes:

1. Les techniques basées sur la génération algorithmique;
2. Les techniques du test pondéré;
3. Les techniques d'insertion de points de test.

Avec les techniques basées sur la génération algorithmique, un générateur algorithmique est utilisé afin de produire un ensemble de vecteurs de test pour les pannes résistantes

1. LFSR: de l'anglais Linear Feedback Shift Register
2. CA: de l'anglais Cellular Automata

au test aléatoire. Cet ensemble est ensuite stocké dans une mémoire ROM ou généré par une circuiterie appropriée [6]. Durant la phase du test, les vecteurs de test de cet ensemble sont appliqués en premier lieu, suivis d'autres vecteurs générés aléatoirement. Dans d'autres techniques de la même catégorie, le générateur algorithmique est utilisé pour générer un test complet, et par la suite, un générateur pseudo-aléatoire est utilisé pour produire un ensemble contenant le test algorithmique déjà généré [21, 96].

Avec les techniques de la deuxième catégorie, l'idée est de générer des vecteurs pondérés pour corriger la faiblesse de l'approche aléatoire. La génération des vecteurs pondérés peut se faire en modifiant un LFSR de telle sorte qu'il tient compte du poids à accorder à chaque entrée du circuit sous test. Il subsiste cependant certains problèmes lorsqu'un seul ensemble de poids uniforme est appliqué. Plusieurs pannes peuvent encore résister lorsqu'un noeud conduisant à plusieurs portes nécessite des poids différents (non uniformes) pour ses sortances. D'où la nécessité de générer des vecteurs pondérés de façon non uniforme [77, 92, 141, 90]. Malheureusement, certaines structures peuvent résister même aux tests pondérés [143]. De plus, il faut générer plusieurs ensembles de poids. Le principe sous-jacent aux techniques utilisant des ensembles de poids multiples est principalement basé sur le partitionnement de l'ensemble des pannes d'un circuit en plusieurs sous-ensembles. Chaque sous-ensemble peut être couvert par un seul ensemble de poids. D'autres auteurs ont proposé une méthode de partitionnement indirect, où l'ensemble des vecteurs de test algorithmiquement généré est partitionné, à la place de l'ensemble des pannes. Enfin, les techniques de la troisième catégorie consistent à insérer un ensemble de points de test afin d'améliorer la testabilité aléatoire du circuit [63, 95, 136, 146], ce que nous utilisons dans notre méthode. Dans la prochaine section, nous présentons quelques techniques de base proposées dans la littérature concernant l'analyse de testabilité et l'insertion de points de test à haut niveau d'abstraction.

2.3 Analyse de testabilité à haut niveau d'abstraction

Un certain nombre de méthodes ont été proposées considérant l'analyse de testabilité et l'insertion de points de test à un haut niveau d'abstraction en l'occurrence le niveau comportemental et RTL. L'objectif de ces méthodes est de générer des circuits facilement testables par les outils ATPG ou dans un environnement de BIST. Dans ce qui suit, nous décrivons quelques méthodes proposées dans la littérature considérant l'analyse de testabilité et l'insertion de points de test au niveau comportemental et au niveau RTL.

2.3.1 Synthèse de haut niveau et l'analyse de testabilité

Dans cette section nous décrivons quelques méthodes qui ont abordé le problème d'analyse de testabilité et d'insertion de points de test durant la synthèse de haut niveau (SHN). Tout d'abord, nous décrivons brièvement les étapes principales de la SHN.

2.3.1.1 Étapes de la synthèse de haut niveau (SHN)

La SHN permet de transformer une description comportementale en une description au niveau RTL en procédant selon les étapes suivantes [129]:

1. **Description de comportement**: cette étape consiste à décrire le comportement du circuit dans un langage de haut niveau tel que un langage de programmation ADA, un langage de description de matériel VHDL, etc.
2. **Génération de la représentation interne**: Un système de synthèse ne peut pas travailler directement sur une description textuelle. Tous les systèmes de synthèse passent par une représentation interne où les relations de dépendance et d'exclusion entre les différentes opérations et données sont dégagées. Généralement, un comportement est décrit par deux graphes. Le premier est un graphe de flot de données (Data Flow Graph - DFG) qui représente les différentes opérations et les dépendances de données entre elles. Le deuxième est un graphe de flot de contrôle (Control Flow Graph - CFG) qui montre les séquencement des opérations tel que spécifié dans la description comportementale du circuit. Ces deux graphes peuvent êtres combinés en un seul graphe pour donner un graphe de flot de données et de contrôle (Control Data Flow Graph - CDFG).

3. **Ordonnancement et allocation**: L'ordonnancement et l'allocation sont deux étapes importante dans la SHN. L'ordonnancement consiste à trouver le meilleur ordre d'exécution des opérations, c'est à dire à assigner les opérations à des étapes de contrôle (cycle de la machine) pour exécuter cette séquence en un temps minimal, sans violer les dépendances en données des opérations. L'étape d'allocation consiste généralement à déterminer trois assignations. La première étape est l'assignation des opérations aux modules fonctionnels, en tenant compte du compromis entre la quantité de matériel et la vitesse d'exécution. La deuxième étape est l'assignation des variables aux registres, en minimisant le nombre de registres alloué. Enfin, la troisième étape est l'assignation des chemins de données à des interconnexions physiques (multiplexeurs ou bus), en minimisant le nombre d'interconnexions.

2.3.1.2 Synthèse des circuits facilement testables par les outils ATPG

Quand l'analyse de testabilité n'est pas considérée durant les étapes de la SHN, le chemin de données (datapath) généré contient souvent plusieurs boucles (lignes de rétroaction). La source de ces boucles est due principalement aux boucles qui se trouvent dans le CDFG et à celles générées durant l'étape d'allocation de ressources. Les boucles contribuent d'une manière significative à la difficulté de la génération séquentielle des vecteurs de test par les outils ATPG [32, 105]. Il a été démontré dans [32] par des résultats expérimentaux que l'utilisation de la technique de balayage partiel permet de minimiser le nombre de boucles et ainsi rendre le circuit facilement testable par les outils ATPG.

Certaines méthodes ont été proposées dans la littérature permettant de générer des chemins de données facilement testables [74, 76]. Ces méthodes utilisent les étapes d'ordonnancement et d'allocation dans le but de minimiser le nombre de boucle générées dans le chemins de données.

2.3.1.3 Amélioration de la contrôlabilité et de l'observabilité des registres

Les techniques traditionnelles d'allocation de registres durant la SHN ont pour objectif de minimiser le nombre de registres alloués pour stocker toutes les variables du CDFG.

Ces registres sont facilement contrôlables (observables) s'ils sont assignés aux entrées (sorties) primaires du circuit généré.

Dans [103], une méthode a été proposée utilisant l'étape d'allocation des variables du CDFG dans le but de maximiser le nombre de registres assignés aux entrées/sorties du chemin de données. Malgré que cette technique essaie d'allouer le minimum de registres et en même temps améliorer la testabilité du chemin de données (améliorer la contrôlabilité et l'observabilité des registres), le nombre de registres peut être excessif. D'où, une augmentation de la surface résultante du chemin de données. Une autre méthode proposée dans [106] a utilisé l'étape d'ordonnancement dans le but de maximiser le nombre de registres assignés aux entrées/sorties primaires du chemin de données. Cette méthode permet d'obtenir une meilleur optimisation du nombre de registres en la comparant à la méthode proposée dans [103].

2.3.1.4 Modification de la description comportementale pour la testabilité

Une description comportementale d'un circuit peut être modifiée dans le but de générer un circuit facilement testable. Dans [41], la description comportementale est convertie en un CFG. Ce dernier est ensuite analysé pour identifier les variables du CFG qui sont difficiles à contrôler et/ou à observer. Les auteurs de cette méthode ont classé les variables du CFG en quatre groupes distincts: variables contrôlables, partiellement contrôlables, observables et partiellement observables. En se basant sur cette classification, de nouvelles instructions de test sont ajoutées dans la description originale du circuit afin d'améliorer la contrôlabilité et l'observabilité des variables identifiées comme non contrôlables et non observables. Il a été démontré dans cette méthode, que la modification de la description originale peut générer des circuits facilement testables au coût d'une surface de sili-

cium additionnelle. Une autre méthode permettant de modifier la description comportementale avec le même objectif de générer un circuit facilement testable a été proposée dans [23]. Dans cette méthode, la description comportementale est convertie en un CDFG. Ce dernier est modifié en ajoutant de nouvelles opérations tout en préservant la spécification originale du circuit. Des opérations comme ajouter la valeur zéro à une variable du CDFG ou multiplier une variable par la valeur unité, sont ajoutées entres les différentes opérations du CDFG. L'objectif de cette modification est de minimiser le nombre de boucles générées dans le chemin de données.

Une autre technique a été proposée dans [51] permettant de contrôler et d'observer localement les entrées/sorties des modules fonctionnels d'un circuit composé de plusieurs modules hiérarchiques. Des modes de test globaux et des contraintes de test sont propagés à partir du module principal à travers les différents modules fonctionnels dans le but de contrôler et d'observer leurs entrées/sorties. Dans le cas où les contraintes de propagation ne sont pas satisfaites, deux possibilités sont envisageables dans cette technique: la première solution est la modification de la description comportementale du module principal. La deuxième solution est la modification des modules locaux dans le but de satisfaire les contraintes de propagation des modes de test [29]. Il a été démontré dans cette technique que la modification du comportement du circuit peut générer des circuits ayant une bonne qualité de test au coût d'une augmentation modeste de la surface.

Enfin, un système de haut niveau appelé "Genesis" a été proposé dans [87]. Le but de ce système est de générer un test d'une manière hiérarchique au niveau des modules fonctionnels d'un circuit donné. Ce système permet de générer un circuit au niveau RTL (partie contrôle et chemin de données) où les entrées (sorties) de chaque module fonctionnel sont contrôlables (observables) à partir des entrées (sorties) du module principal. Dans

cette méthode, on associe à chaque opération un environnement de test afin de contrôler (observer) ses entrées (sorties) à partir des entrées (sorties) du module principal. Considérons le DFG de la figure 2.3 où on veut trouver un environnement de test de l'opération (*2). Pour cela, nous avons besoin de contrôler les entrées p et b et observer sa sortie q. Soit v1, le vecteur de test désiré qui est appliqué à l'entrée p et le vecteur v2 à l'entrée b, générant à la sortie q le vecteur v3. Si on assigne aux entrées primaires a = (v1 - v2), b= v2 et c = 0, le vecteur de test désiré v1 est justifiable aux entrées de l'opérateur (*2) et la réponse v3 est observable à la sortie f du module principal. Donc, l'environnement du test pour l'opération (*2) est déterminé par les propriétés suivantes: a = (v1 - v2), b = v2 et c=0.

Cependant, cette méthode reste limitée en utilisant des propriétés de transparence des opérations (ajouter la valeur 0 à une variable ou multiplier une variable par la valeur 1) dans le DFG. En effet, il n'est pas toujours possible de trouver un environnement de test pour une opération dans un DFG. Une amélioration de cette méthode a été proposée dans [89]. Cette méthode utilise la notion d'environnement de test développé dans [87] dans le but d'identifier les signaux internes de la description RTL qui sont difficilement contrôlables ou observables. Des multiplexeurs sont insérés au niveau RTL afin de faciliter la justification du test d'une opération donnée.

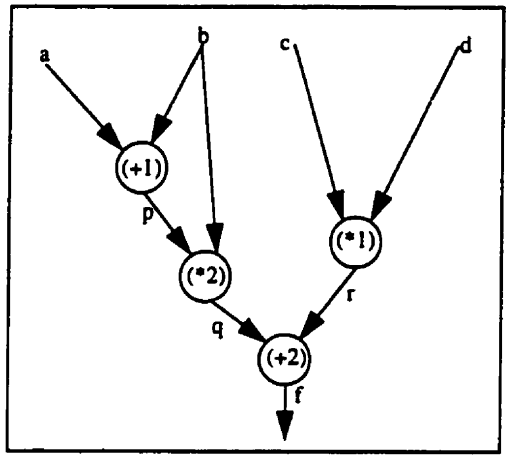

Figure 2.3 Exemple d'environnement de test d'une opération dans le DFG.

2.3.2 Synthèse pour le test intégré (BIST)

Certaines méthodes ont été proposées dans la littérature pour générer durant les étapes de la SHN, des circuits facilement testables dans un environnement de BIST et de test aléatoire [79, 123]. Ces méthodes utilisent l'étape d'allocation des unités fonctionnelles et des registres dans le but d'orienter la synthèse de générer un chemin de données facilement testable dans un environnement de BIST. Basées sur des techniques de graphes de conflit et de calcul de fonction de coût, ces méthodes se concentrent aux problèmes de contrôlabilité et d'observabilité des registres. L'évaluation de la testabilité des unités fonctionnels n'est point évoquée.

Une autre méthode permettant de générer à partir d'une description comportementale, un circuit facilement testable par la méthode BIST, a été proposée dans [50, 80]. Des mesures de testabilité ont été développées pour estimer la contrôlabilité et l'observabilité des signaux internes de la description comportementale dans un contexte de test aléatoire. En se basant sur ces mesures de testabilité, des points de test sont ensuite insérés dans la

description originale du circuit afin d'améliorer la contrôlabilité et l'observabilité des signaux internes. Cependant, chaque point de test nécessite un registre supplémentaire qui sera configuré en un générateur de vecteurs de test ou un analyseur des réponses. D'où l'inconvénient majeur de cette méthode où la surface additionnelle peut être trop significative pour des circuits complexes. De plus, l'insertion de points de test au niveau comportemental est considérée seulement pour le chemin de données. Dans la partie de la logique de contrôle et l'interface entre le chemin de données et la logique de contrôle, l'insertion de points de test est purement structurelle, c'est à dire considérant la description du circuit après les étapes la SHN.

2.3.3 Analyse de testabilité au niveau Transfert de Registres (RTL)

Un ensemble de méthodes a été proposée considérant l'analyse de testabilité au niveau RTL, c'est-à-dire avant la synthèse logique. L'objectif principal de la plupart de ces méthodes est de générer un circuit facilement testable par les outils ATPG (séquentiel ou combinatoire).

Une méthode a été proposée dans [17] considérant l'analyse de testabilité et l'insertion de points de test au niveau RTL pour des circuits spécifiés en langage VHDL. Cette méthode utilise la méthode de test de balayage partiel. La description VHDL du circuit est convertie en une représentation intermédiaire appelée, réseaux de Petri temporel étendus (ETPN: Extended Time Petri Net) [124]. Chaque noeud du ETPN représente un registre ou une unité fonctionnelle. Des mesures de testabilité (contrôlabilité et observabilité) ont été définies et utilisées pour identifier les signaux qui sont difficilement contrôlables et observables. Après quoi, des registres sont insérés comme points de test. En effet, lorsqu'un bit d'un signal est identifié comme non contrôlable ou non observable, des registres sont insérés dans tous les bits de ce signal indépendamment de la valeur de

contrôlabilité et d'observabilité des autres bits de ce même signal. Par conséquent, une augmentation de la surface du circuit en résulte.

Une autre méthode appelé "RT-SCAN", a été proposée dans [86]. L'idée de base de cette méthode est de transformer le circuit en un circuit combinatoire sans utiliser aucune méthode de balayage (complet ou partiel). Dans cette méthode, les auteurs essaient d'utiliser tous les chemins de données possibles entre les paires de registres du circuit au niveau RTL. Des multiplexeurs sont utilisés pour créer ces chemins et ainsi contrôler et observer le contenu des registres. Cependant, de nouveaux chemins sont nécessaires à ajouter dans le circuit afin de propager le contenu des registres ce qui résulte en une augmentation de la surface du circuit.

Enfin, une méthode d'analyse de testabilité au niveau RTL a été proposée dans [16]. Les auteurs ont utilisé le langage de description Verilog au niveau RTL pour spécifier le circuit et des vecteurs de test fonctionnels pour évaluer la couverture de pannes résultante. Entre autres, cette méthode permet aussi d'identifier les signaux internes difficile à contrôler ou à observer. Pour améliorer la couverture de pannes, des vecteurs additionnels sont ajoutés dans la première étape. Ensuite, une modification du code Verilog est suggérée par les auteurs pour améliorer la couverture de pannes. Cependant, cette modification du code est laissée à la responsabilité du concepteur.

2.3.4 Conclusion

Selon les travaux cités jusque là, nous remarquons une diversité des stratégies adoptées par les méthodes proposées dans la littérature pour résoudre le problème d'analyse de testabilité et d'insertion de points de test à un haut niveau d'abstraction (comportemental et RTL). Ces méthodes diffèrent l'une de l'autre par l'objectif de méthodologie du test à

atteindre à savoir: synthèse des circuits facilement testables par les outils ATPG (séquentiel et combinatoire) ou par la méthode BIST. Au niveau comportemental, certaines méthodes utilisent l'étape d'allocation de ressources matérielles et celle de l'ordonnancement des opérations dans le CDFG dans le but de générer des circuits facilement testables par les outils ATPG ou la méthode BIST. D'autres méthodes essaient de modifier la description comportementale du circuit avant le processus de SHN. Cependant, comme nous l'avons mentionné, la plupart de ces méthodes proposées se sont concentrées à améliorer seulement la testabilité du chemin de données. Il est supposé dans ces méthodes, que la logique de contrôle peut être testée indépendamment du chemin de données. En effet, même si le chemin de données et la logique de contrôle sont testés individuellement, leurs interconnexion en une seule unité n'est pas forcément facilement testable [125]. En plus, ces méthodes restent incompatibles avec les outils de synthèse existant dans le marché actuel tels que Synopsys, Mentor, Cadence, etc. Au niveau RTL, la plupart des méthodes se sont concentrés à synthétiser des circuits facilement testables par les outils ATPG séquentiel ou combinatoire. Leur objectif est d'obtenir un faible temps CPU, un faible nombre de vecteurs de test et une couverture de pannes maximale au détriment de surface et de délai additionnels. Jusqu'à présent, aucune méthode n'a abordé d'une manière systématique le problème d'insertion de points de test au niveau RTL pour les circuits spécifiés en langage VHDL ou en Verilog. Une méthode qui a évoqué ce problème a été proposée dans [16]. Dans cette méthode, les auteurs identifient les signaux difficiles à contrôler et/ ou à observer et laissent les modification du circuit à la responsabilité du concepteur. Aussi, le problème de contrôlabilité et d'observabilité des signaux internes des modules fonctionnels reste un problème majeur pour toutes les méthodes proposées. Contrairement à ces méthodes, dans notre méthode, nous analysons au niveau RTL les signaux et les variables de la spécification VHDL ainsi que certains signaux internes difficiles à contrô-

ler ou à observer des modules fonctionnels (additionneurs, comparateurs, etc). Ensuite, une modification du code VHDL est effectuée au niveau de ces signaux et variables. Nous considérons aussi chaque bit de chaque signal et variable pour l'analyse de testabilité et l'insertion de points de test indépendamment des autres bits, ce qui n'est pas le cas pour les autres méthodes. En effet, cette insertion au niveau bit permet de réduire la surface et rend la méthode plus proche de l'insertion structurelle. Enfin, un ensemble de points de test est inclus dans la spécification originale VHDL du circuit. De cette manière, l'outil de synthèse utilisé permet d'optimiser simultanément la logique fonctionnelle du circuit original et la logique des points de test insérés en tenant compte de toutes les contraintes de conception à savoir, la surface, la vitesse du circuit, la testabilité, etc. Enfin, notre méthode est compatible avec les outils de synthèse existant tels que Synopsys, Mentor Graphics, Cadence, etc.

CHAPITRE 3

Analyse de testabilité et d'insertion de points de test au niveau transfert de registres

3.1 Introduction

Dans le présent chapitre, qui a fait l'objet d'une publication soumise à "IEEE Transactions on Computer-Aided-Design", on propose une nouvelle méthode d'analyse de testabilité et d'insertion de points de test dans les circuits intégrés décrits au niveau RTL et spécifiés en langage VHDL. Notre méthode utilise les mesures de testabilité développées et présentées en l'annexe 1. La méthode de calcul des mesures de testabilité s'inspire de la méthode de COP développée au niveau portes logiques et adapté au niveau RTL. En effet, l'estimation des mesures de testabilité au niveau RTL permet de réduire les erreurs dues aux problèmes de reconvergence dans les réseaux de portes logiques. Aussi, la méthode de calcul de COP devient de plus en plus complexe au niveau portes logiques à cause de la complexité croissante des circuits intégrés.

Dans ce chapitre, nous montrons comment utiliser ces mesures de testabilité pour guider le processus d'insertion de points de test au niveau RTL pour les circuits spécifiés en langage VHDL. Tout d'abord, la spécification VHDL est convertie en un Graphe Acyclique Dirigé (Directed Acyclic Graph - DAG). Chaque noeud du DAG représente une opération VHDL (addition, multiplication, etc.) et les arcs représentent les signaux et

variables de la spécification VHDL. Les étapes de construction du DAG sont présentées en l'annexe 2. Ces mesures de testabilité consistent à calculer la contrôlabilité à 0 et à 1 et l'observabilité de chaque bit de chaque signal/variable ainsi que certains signaux internes des modules fonctionnels (additionneurs, comparateurs, etc.) dans la spécification VHDL. Ces signaux internes sont obtenus en décomposant les modules fonctionnels en sous-modules fonctionnels élémentaires. En utilisant les résultats du calcul des mesures de testabilité, on identifie le(s) bit(s) de chaque signal/variable qui sont difficiles à contrôler et/ou à observer. Des points de test sont ensuite insérés au niveau VHDL. Chaque point de test consiste en un point de contrôle ou un point d'observation où il est défini par une fonction (pour le point de contrôle) ou par une procédure (pour le point d'observation). À cet effet, un package a été défini incluant toutes les définitions des fonctions et procédures. Des circuits-test décrits au niveau RTL et spécifiés en langage VHDL synthétisable sont utilisés pour montrer l'efficacité de notre méthode en termes de surface, vitesse du circuit et de testabilité. D'autres résultats expérimentaux sont aussi présentés dans le chapitre 4.

3.2 Testability Analysis and Test-Point Insertion in RTL VHDL Specifications For Scan-Based BIST

Samir Boubezari[1], Eduard Cerny[2], Bozena Kaminska[1] and Benoit Nadeau-Dostie[3]

Electrical Engineering Department [1]
École Polytechnique de Montreal,
P.O. Box 6079, Station Centre Ville,
PQ, H3C 3A7 Canada
samir@grm94.polymtl.ca
bozena@opmaxx.com

Département d'informatque[2]
et de recherche opérationnelle,
Université de Montréal, C.P. 6128,
succ. Centre Ville, Montréal, P.Q,
Canada, H3C 3J7,
cerny@iro.umontreal.ca

LogicVision[3]
1525 Carling Avenue,
Suite 404, Ottawa, Ontario,
Canada, K1Z 8R3
benoit@lvision.com

ABSTRACT

This paper proposes a new testability analysis and test-point insertion method at the Register Transfer Level (RTL), assuming a full scan and a pseudorandom BIST design environment. The method is based on analyzing the RTL synchronous specification in synthesizable VHDL. A VHDL Intermediate Form representation is first obtained from the VHDL specification and then converted to a Directed Acyclic Graph that represents all data dependencies and flow of control in the VHDL specification. Testability Measures are computed on this graph. The considered TMs are Controllability and Observability for each bit of each signal/variable that is declared or may be implied in the VHDL specification. The calculation is carried out at a functional level rather than the gate level, to reduce or eliminate errors introduced by ignoring reconvergent fanout in the gate network, and to reduce the complexity of the DAG construction. Internal signals of Functional Modules (implementation of VHDL operations), are also analyzed to compute their Controllability and Observability values. Based on the Controllability/ Observability values, test-point insertion is performed to improve the testability for each bit of each signal/variable. This insertion is carried out in the original VHDL specification and thus becomes a part of it unlike in other existing methods. This allows full use

of RTL synthesis optimization on both functional and test logic concurrently within the designer constraints such as area and delay. A number of benchmark circuits were used to show the applicability and the effectiveness of our method in terms of the resulting testability, area, and delay.

3.2.1 Introduction

VLSI circuit complexity has made testing difficult and more expensive. Increasing the testability of a design is one of the important issues in the design cycle. In the past, we could add Design-For-Testability (DFT) circuits manually, after logic synthesis. But today's need for a shorter time to market makes this an unaffordable bottleneck. Ignoring DFT during the design cycle affects product quality and introduces schedule delays. Most industrial digital designs use automated RTL synthesis and we can thus achieve DFT by incorporating test and synthesis into a single methodology that is automated as possible. Indeed, considering testability during design synthesis can reduce the overall design and manufacturing time. Even more important, the testability enhancement at the entry level to a synthesis tool makes it independent of the tool and the implementation technology. It becomes part of the design specification and may be optimized with the other synthesis tasks in terms of area and delay.

The main objective of our method is thus to raise the level of abstraction at which testability analysis and test-point insertion is performed. We propose a new testability analysis and test-point insertion method at the RTL, assuming full scan and pseudorandom BIST design environment. Full scan in combination with pseudorandom patterns is widely adopted in the industry due to its ease of implementation and fault diagnostic. Unfortunately, the presence of random pattern resistant faults in many practical circuits poses a serious limitation to its success. The solutions to tackle this limitation can be broadly clas-

sified as those that modify the input patterns or those that modify the circuit-under test. In this paper, we are interested in the second class of solutions, circuit modifications, that introduce test points to improve the random pattern testability of a circuit. Our goal is to analyze and modify the VHDL RTL description of the circuit, in order to generate an easily testable gate-level circuit by a pseudorandom sequence under the BIST environment. This is the main advantage and motivation of this work. That is, to apply synthesis compilation and optimization technology directly to a testable VHDL description, optimizing functional and inserted test logic concurrently, rather than introducing testability after the VHDL has been compiled to gate-level. The resulting gate-level circuit will be optimized for delay, area and testability.

The proposed method uses as the starting point a VHDL specification given at the synthesizable RTL. It is analyzed to produce an intermediate representation, called the VHDL Intermediate Format (VIF), and transformed into a Directed Acyclic Graph (DAG) on which testability analysis is performed by computing and propagating Testability Measures (TMs) forward and backward through the VHDL statements. The TMs are the Controllability and the Observability for each bit of each signal/variable. Some internal signals of Functional Modules (FMs) such as adders, comparators and multiplexers are also analyzed to determine their Controllability and the Observability values. The internal signals are obtained by decomposing the FMs into sub-FM blocks.

These measures are then used to identify bits of signals/variables having too low Controllability or Observability. Test-point insertion is performed to improve Controllability and Observability, again at the RTL.

Test points at the VHDL RTL are described by a set of synthesizable VHDL functions (procedures) which are used to insert Control (Observation) points on bits of signals/vari-

ables. The corresponding functions/procedures are defined in a package which is included in the original specification. The definition of these functions/procedures includes the normal and the test mode of the specification such that after technology mapping, the resulting gate-level circuit can be executed in both modes.

The computation of TMs method is purely functional, that is, it does not assume the knowledge of a gate-level implementation of the circuit being analyzed. Therefore, it allows us to compute testability estimations with a high degree of accuracy for circuits on which existing tools fail due the enormous amount of information contained in a gate-level implementation of the circuit. Some benchmark circuits which are random pattern resistant are used to show the effectiveness and the viability of the proposed method in terms of the resulting testability, area, and delay.

The paper is organized as follows: Section 3.2.2 gives a summary of previous work in the literature. The overall approach is summarized in Section 3.2.3. Section 3.2.4 describes the DAG construction, while Section 3.2.5 presents the main formulas of the TMs calculations. The test-point insertion method is discussed in Section 3.2.6. Section 3.2.7 presents the experimental results, and Section 3.2.8 concludes the paper.

3.2.2 Previous work

Recently, several RTL and behavioral level design and synthesis-for-testability approaches were proposed to generate easily testable circuits for partial scan, sequential ATPG, and BIST testing methodology [62]. The proposed approaches include RTL scan selection [69, 17], modifications to the behavioral description of a design to improve the testability of the synthesized circuit [41, 70], and considering testability during the behavioral synthesis process [74, 76, 78, 79]. The high-level techniques concentrate on improv-

ing the testability of datapaths, assuming that the controller can be tested independently and that its outgoing control signals to the datapath are fully controllable in the test mode. However, even when both the controller and the datapath are individually testable, the composite circuit may not be. A method based on testability analysis at the behavioral level was presented in [50, 80]. The authors perform the test-point insertion in the datapath at the behavioral level. However, for the controller and the interface between the datapath and the controller, test-point insertion is performed at the structural level. For hierarchical designs, a technique has been developed in [70, 29, 51] to generate top test modes and constraints required to realize a module's local test modes. The process of generating global test modes may reveal that some constraints cannot be satisfied, in which case, either the top level description of an individual module, must be modified to satisfy the constraints. It has been shown that behavioral modification can yield implementation with higher test efficiency than the original design with a modest increase in area. A high level synthesis system called "Genesis" was proposed in [87] which targets hierarchical testability. It synthesizes RTL controller/datapath circuits from a behavioral description in such way that the precomputed test sets of all the modules and registers are justifiable at the system level. In this technique, limited transparency properties of operations (0 for addition, 1 for multiplication) are used to check the existence of a justification and propagation path for the operation in the Data-Flow-Graph. An improvement of this method was proposed in [89]. It uses hierarchical testability analysis developed in [87] to identify signals in the given RTL circuit that are bottlenecks from the hierarchical Controllability/ Observability point of view. It then selectively inserts test multiplexers to render the circuit hierarchically testable. The Testability of signals is simply two-valued (testable or not testable).

Some RTL testability analysis methods have been proposed to generate easily testable circuits for sequential ATPG [81, 82, 86]. The main objective of these methods is to reduce the ATPG CPU time at the expense of area overhead. In [81], an RTL method was proposed which enables circuit testing using combinational test patterns. The methodology uses existing paths between registers, going through multiplexers, to load a combinational test pattern into the circuit FFs without having to use scan FFs. A technique was proposed later in [82] that can also use existing paths through functional units. However, appropriate constants (identity elements) need to be added to the side inputs of the units to create I-paths[83]. An improvements of this method was proposed recently in [86]. Finally, an RTL testability analysis method was proposed in [16]. The authors use Verilog RTL models and functional verification patterns to improve the fault coverage of the resulting circuit at the gate level. Also, this method provides information about the areas in the RTL models with a potential testability improvement by architectural changes or an RTL fault coverage improvement by adding more test patterns. However, test-point insertion at the RTL was not addressed in this method and it was left as the designer's responsibility.

3.2.3 The proposed method

Figure 3.1 depicts the overall structure of our testability analysis environment which can operate as a front-end to an RTL synthesis tool such as Synopsys Design Compiler. In the first step, a VHDL analyzer from LEDA [48] is used to produce the VIF representation, and to identify all registers (full scan is assumed) and sequential VHDL statements[1]. A Directed Acyclic Graph (DAG) is used to store this information by linking the present

1. Note that concurrent statements will be translated to their equivalent processes containing sequential statements.

states of registers with the next states through the VHDL statements. All integers and enumerated types are converted to bits or bit vector and VHDL operations are modeled by their Boolean functional models. The TMs are the Controllability and the Observability of each bit of each signal/variable. TMs are then computed using this DAG by initializing the Controllability of primary and pseudo-primary inputs to 0.5 (both 0 and 1), the Observability of primary and pseudo-primary outputs to 1, and by propagating them forward and backward through the VHDL statements.

A method for propagating TMs through VHDL operators was developed. It allows to identify hard-to-detect bits of signals/variables of the VHDL specification including internal signals of FMs. This information is used to insert test points, again in the specification at the RTL by locally converting the affected signal/variable to the bit level and back. Each test points is described by a synthesizable VHDL function/procedure which is defined in a package. The Function (procedure) is used to insert a Control (Observation) point on a given bit of a signal/variable in the VHDL specification. As a result of the test-point insertion, our method again produces a synthesizable RTL VHDL specification which can be input to a synthesis tool. This allows designers to optimize their designs for different design constraints (e.g., area and delay) including testability. This is the main advantage of our method to include testability at the VHDL level before RTL synthesis unlike in the existing approaches. The algorithm in Figure 3.1 was implemented using the C language in about 14,000 code lines. In the following sections, we describe each step in Figure 3.1.

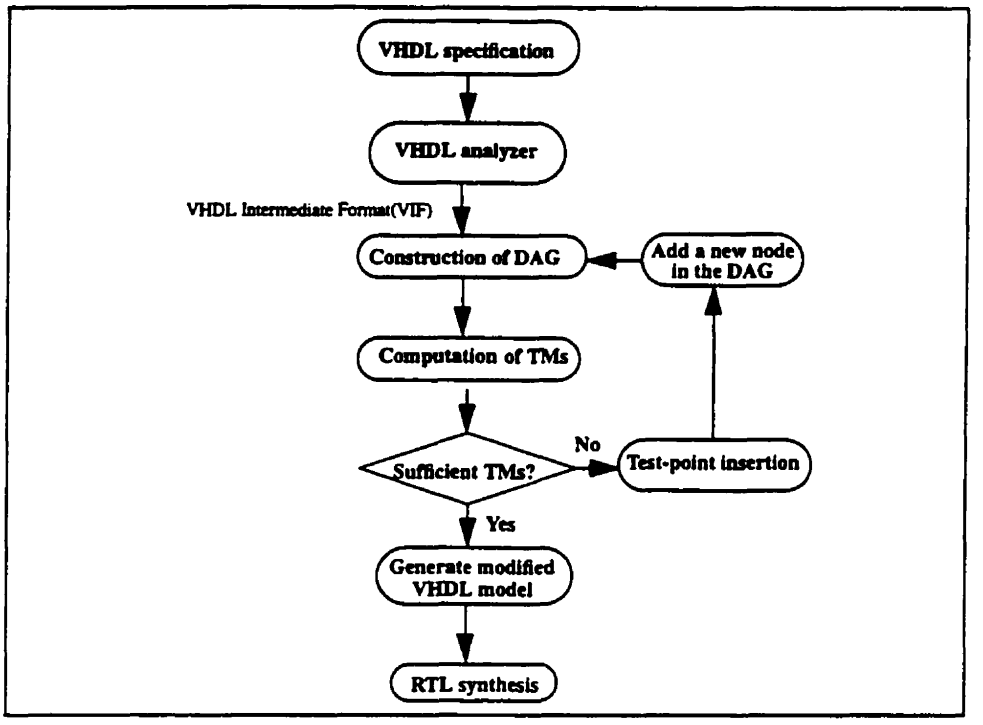

Figure 3.1 Hardware synthesis with incorporated testability analysis and test-point insertion.

3.2.4 Construction of the Directed Acyclic Graph

As shown in Figure 3.1, a VHDL specification is compiled into its VIF representation, and then a DAG is constructed that represents the flow of information and data dependencies. Each internal node of the DAG corresponds to an operation of the VHDL specification such as arithmetic, relational, data transfer and logical operations. The source (sink) nodes of the DAG represent the present (next) state and primary inputs (outputs). The present and the next states are given by the registers that could be synthesized from the VHDL specification. Edges represent signals/variables declared by the designer in the specification and intermediate signals/variables as defined in Definition 1.

Note that no hardware sharing is performed during the VHDL translation in the DAG. That means, each VHDL operation corresponds to a new node in the DAG. Note also that the entire VHDL language is supported by the analyzer, however, only the synchronous synthesizable constructs as accepted by commercial RTL synthesis tools (Synopsys, Mentor,...etc.) are supported by our method.

Definition 1:

An intermediate signal/variable is an unnamed signal/variable formed by an expression which is not a simple signal/variable name. For example, in the VHDL specification shown in Figure 3.2(a), two intermediate signals (S_int1 and S_int2) and two intermediate variables (V_int1 and V_int2) are implied as shown in Figure 3.2(b).

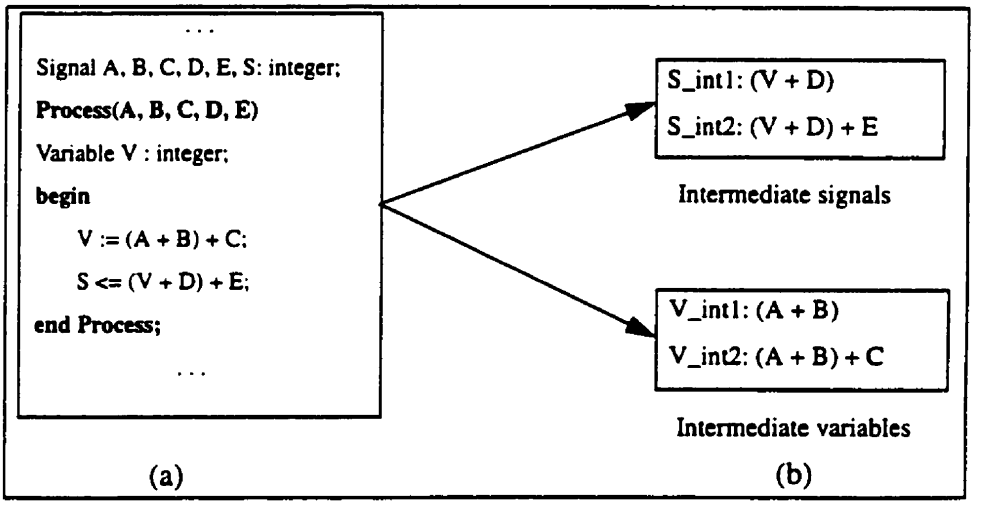

Figure 3.2 Definition of intermediate signals/variables.

In the rest of this paper, whenever a signal/variable is mentioned, it refers to a signal/variable that is declared by the designer in the VHDL specification or to an intermediate

signal/variable as stated in Definition 1. Below, we summarize the main steps in constructing the DAG:

1. Generate Control and Data Flow Graph (CDFG) for each process of the VHDL specification.
2. Unroll all for...loops and expand procedures/functions by adding new nodes to the CDFG.
3. Convert data types to bits.
4. Translate the resulting CDFG into a DAG.
5. Connect DAG graphs of individual process to produce the global DAG.

3.2.4.1 Generation of CDFG

Each VHDL process can be transformed into a Control Flow Graph (CFG) to represent the control flow of operations. The CFG is a directed graph defined as:

$$CFG = (V, E) \text{ , where}$$

V is a set of nodes corresponding to the different VHDL sequential statements:

$$V = V_w \cup V_b \cup V_l \cup V_e \text{, where}$$

V_w is the set of synchronization nodes (wait statements),

V_b is the set of conditional statement nodes (if, case),

V_l is the set of for...loop statement nodes,

V_e is the set of other nodes (signal/variable assignments, procedure calls,...).

and E is the set of edges representing the flow of control.

Each node of a CFG is associated with a Data Flow Graph (DFG). Each node of DFG represents one operation in the VHDL specification, and an DFG edge represent signals

and variables connecting the nodes. There is an edge from operation o_i to operation o_j if the result of operation o_i is input to operation o_j. A CFG in which each node is a DFG is called a CDFG.

3.2.4.2 Loop unrolling and expansion of procedures/functions

For...loop statements are used to repeat a sequence of operations for a constant number of times. The result of synthesis would be a replication of the hardware corresponding to the statements inside the loop, one for each iteration. Each procedure call is expanded in-line. The contents of the procedure are first copied into the process in place of the call. Then, the actual parameters are substituted for the formal parameters of the procedure. Functions are expanded in a similar fashion except that a function is expanded immediately before the expression that calls it. As a result of loop unrolling and expansion of procedures/functions, the CDFG is augmented by new nodes.

3.2.4.3 Data type conversion

If already not declared as such, all VHDL data types are converted into bits. The main issue with enumerated types is the encoding of the possible values. By default we can convert them into bit-vectors whose length is determined by the minimum number of bits required to code the number of enumerated values, and the actual code assigned to each value corresponds to the binary representation of the position in the type declaration (starting from zero).

Integer subtypes are defined using subranges that impose bounds on the possible values. They are encoded using bit-vectors whose length is the minimum necessary number of bits to hold the defined range. If the range includes negative numbers, it is encoded as a 2's-complement bit-vector. In general, bits of an integer type value are not directly acces-

sible and depends on the synthesis tool we use to synthesize the circuit. No assumption is made on the location of these bits. Functions for translating integers to bit vectors and back are used from the packages provided by the synthesis tool. With Synopsys, we use the standard packages of IEEE in which functions for type conversions are defined.

3.2.4.4 Translation of the CDFG into DAG

The next step is to translate the resulting CDFG into a DAG. Synchronous registers are inferred on signals and some variables assigned in a clocked process. However, once we separate the present and the next states of registers into separate nodes, we obtain a DAG. The source nodes are the primary inputs and present state values of registers and the sink nodes are the primary outputs and the next-state values. Pseudo-primary inputs (outputs) correspond to synthesized register outputs (inputs). The algorithm to identify the synthesized registers in the VHDL specification is the same as used by most synthesis tools.

Identification of multiplexers

In this section we give a translation for the VHDL conditional statements ("if" and "case"). A new node type is added to the DAG representing the multiplexer operation. In fact, each conditional statement ("if" or "case") is translated into a set of multiplexers with one output for each signal/variable assigned within the conditional statement. The condition expression translates into a set of nodes which feed the control input of each multiplexer. The data inputs to each multiplexer are fed from the nodes of the corresponding expression being assigned. Thus, to translate each conditional statement to a multiplexer operations, we have to find its scope, i.e., its corresponding end-statement.

3.2.4.5 Connecting of processes

A VHDL specification consists of a set of interacting processes (clocked and unclocked). Entity ports and internal signals (declared in the VHDL architecture) are used to communicate between the processes. The connection between processes is represented as a Directed Graph. However, we have seen that each VHDL process can be transformed into a CDFG which is represented as a DAG. Hence, the global VHDL specification is also represented as a DAG. We illustrate the overall construction on the following example.

Example 1:

Consider the VHDL specification as shown in Figure 3.3, it represents a Moore finite state machine with 4 states (S0, S1, S2, S3) and one output Z. The specification consists of two processes (a clocked one and a combinational one). Its DAG is shown in Figure 3.4, the primary and pseudo-primary inputs/outputs representing the present and the next registers are also indicated there. The signal CURENT_STATE is synthesized as a register and thus becomes a pseudo-primary two bit-wide input/output, since it is declared as an enumerated data type of 4 possible values. The constants S0, S1, S2, and S3 are encoded as 00, 01, 10, and 11, respectively. Each multiplexer operation corresponds to a conditional statement.

3.2.5 Testability computation

Most previous testability analysis methods are restricted to circuits consisting of logic gates only, i.e., complex functional modules must be expanded to a gate-level equivalent. In contrast, our method handles most VHDL operations at the functional level in addition to logical operations. The following operators are supported by our method: n-bit adders, n-bit comparators (<, <=, ... etc.), n-bit multipliers, n-bit subtractors, and general multi-

```
entity MOORE is          -- Moore machine
 port(X, CLOCK: in BIT;
    Z: out BIT);
end;
architecture BEHAV of MOORE is
 type STATE_TYPE is (S0, S1, S2, S3);
 signal CURRENT_STATE, NEXT_STATE: STATE_TYPE;
begin
 -- Process to hold combinational logic
 COMBIN: process(CURRENT_STATE, X)
 begin
  case CURRENT_STATE is
   when S0 =>
    Z <= '0';
    if X = '0' then
     NEXT_STATE <= S0;
    else
     NEXT_STATE <= S2;
    end if;
   when S1 =>
    Z <= '1';
    if X = '0' then
     NEXT_STATE <= S0;
    else
     NEXT_STATE <= S2;
    end if;
                    ./..
```

```
   when S2 =>
      Z <= '1';
      if X = '0' then
       NEXT_STATE <= S2;
      else
       NEXT_STATE <= S3;
      end if;
     when S3 =>
      Z <= '0';
      if X = '0' then
       NEXT_STATE <= S3;
      else
       NEXT_STATE <= S1;
      end if;
   end case;
end process COMBIN;
-- Process to hold synchronous elements (flip-flops)
 SYNCH: process
 begin
  wait until CLOCK = '1';
  CURRENT_STATE <= NEXT_STATE;
 end process SYNCH;
end BEHAV;
```

Figure 3.3 VHDL specification of a Moore machine.

plexers which are inferred by the conditional statements (if, case). All of these are represented functionally, meaning that the Controllability/Observability propagation through them is computed with a high degree of accuracy. This is not the case with gate-level models, because reconvergent fanout in such models may introduce errors in the calculations. In our method we compute a Controllability of zero (C_0) and of one (C_1), and Observability (O) values on each bit of each signal/variable and internal signals of FMs. We show next the propagation of C_0, C_1 and O through typical VHDL operators.

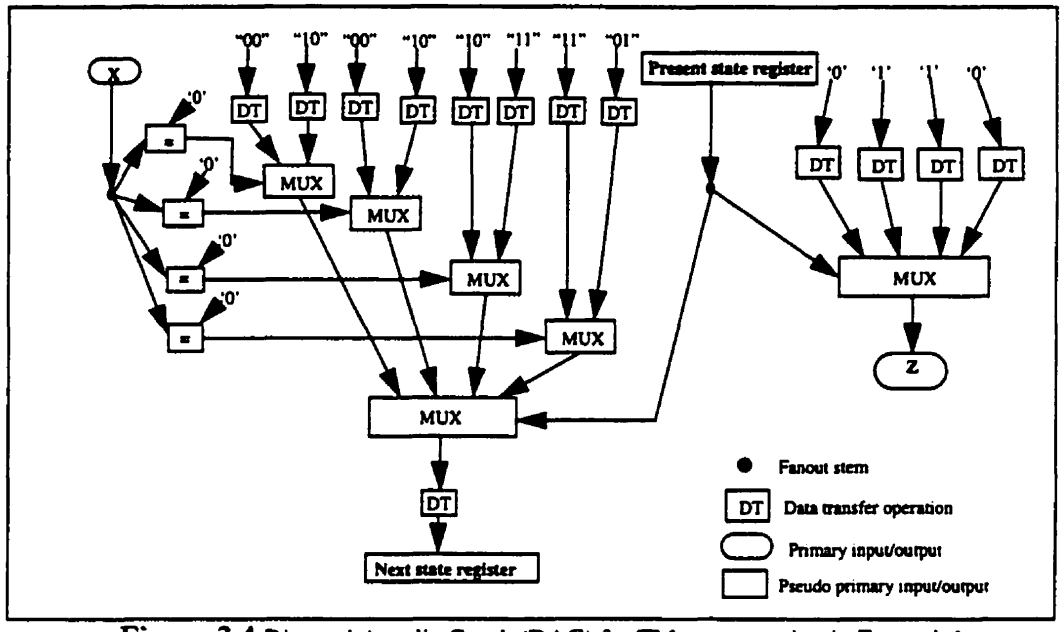

Figure 3.4 Directed Acyclic Graph (DAG) for TMs computation in Example1.

Definition 2: Combinational Controllability [14] is the probability that a signal s has a specific value. We have two measures, 1-Controllability ($C_1(s)$) and 0-Controllability ($C_0(s)$) such that $C_0(s)= 1 - C_1(s)$.

Definition 3: Combinational Observability $O(l, s)$ of a line l is defined as the probability that a signal change on l will result in a signal change on an output s. For multiple output modules, the Observability of a line must be computed relative to each output and the overall Observability $O(l)$ of l is given by $max_s [O(l, s)]$

3.2.5.1 Controllability calculations

In this section, we give the formulas for determining the Controllability of an output of some VHDL operators, given the Controllability of the inputs. We show next the Controllability formula for an n-bit adder, the other formulas are included in Appendix 1.

We wish to compute the Controllability of the outputs of an n-bit adder, given the Controllability of its inputs, assuming that the inputs are independent. When two n-bit binary numbers a and b are added, each bit of the sum s is a function of the corresponding bits of a and b and of the carry from the next less significant bits.

The 1-Controllability measures $C_1(s_i)$ and $C_1(c_{i+1})$ can be computed by considering the minterms leading to 1 on the respective output:

$$C_1(s_i) = \alpha + C_1(c_i) - 2 \times (\alpha \times C_1(c_i)) \tag{3.1}$$

$$C_1(c_{i+1}) = \alpha \times C_1(c_i) + C_1(a_i) \times C_1(b_i) \tag{3.2}$$

$$\text{where } \alpha = C_1(a_i) + C_1(b_i) - 2 \times C_1(a_i) \times C_1(b_i) \tag{3.3}$$

Note that α is the probability that $(a_i \oplus b_i) = 1$ and, consequently, $C_1(s_i)$ is the probability that $(a_i \oplus b_i \oplus c_i) = 1$.

To compute the Controllability of each output of an n-bit adder, we can use a cascade of n full adders configured as a ripple-carry adder as shown in Figure 3.5. There is no reconvergent fanout in this circuit and all inputs are independent, hence the Controllability computed on this tree structure is exact. To find the Controllability measure $C_1(s_i)$ at the output s_i, Equations (3.1), (3.2) and (3.3) can be used for each 1-bit adder, and bit i can be evaluated when bits 0, 1, ..., $i-1$ have been computed. The calculation can be made in linear time in terms of the number of inputs. The same structure can be used to compute the Controllability of a subtractor using 2's-complement representation of negation.

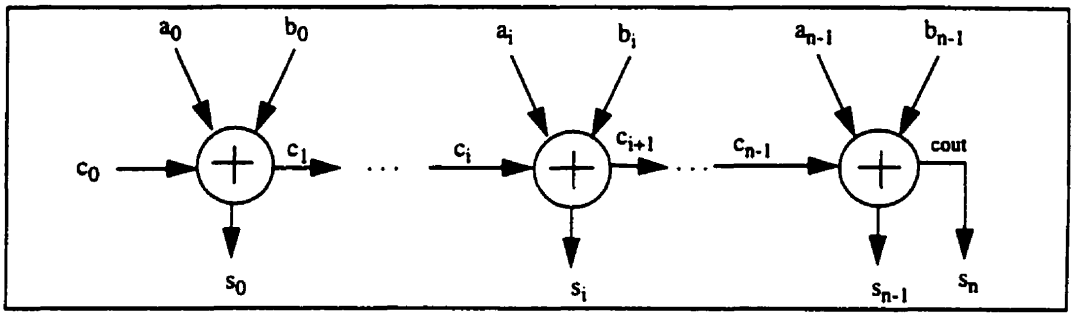

Figure 3.5 A ripple-carry adder composed of n full adders

3.2.5.2 Observability calculations

In this section, we give the Observability formula for an n-bit adder. The other formulas can be found in Appendix 1.

Consider again the n-bit ripple-carry adder shown in Figure 3.5, and let us compute the Observability of each input. According to the boolean function of a 1-bit adder, the change on any input a_i, b_i or c_i is always observable at s_i. It follows that:

$$O(a_i, s_i) = O(b_i, s_i) = O(c_i, s_i) = O(s_i), \quad i = 0, \ldots, n-1 \tag{3.4}$$

The Observability of an input at level i at the other outputs k, $k > i$, depends on the propagation of the carry from stage i to these outputs. For instance, to observe a_i at s_k such that $k = i+1, \ldots, n$, we have to set $(b_i \oplus c_i) = 1$ and $(a_j \oplus b_j) = 1$, for all j such that $j = i+1, \ldots, k-1$. Equation (3.5) gives the general formula to compute the Observability of each input a_i at outputs k, such that $k = i+1, \ldots, n$.

$$O(a_i, s_k) = \left[C_1(c_i \oplus b_i) \times \prod_{j=i+1}^{k-1} C_1(a_j \oplus b_j) \right] \times O(s_k), \; i = 0, \ldots n-1, \tag{3.5}$$

The resulting Observability value of input i is the maximum value of Equations (3.4) and (3.5), i.e., $O(a_i) = max_{i \le k \le n}[O(a_i, s_k)]$.

Similar formulas can be used to compute the $O(b_i, s_k)$ and $O(c_i, s_k)$ (in particular for c_0).

3.2.6 Test-point insertion

In this section, we derive a modified VHDL specification from the original one which includes a set of test points expressed using synthesizable VHDL functions and procedures. Functions (procedures) are used to improve the Controllability (Observability) on bits of some signals/variables and the internal signals of FMs. The modified VHDL specification describes both the normal and the test modes of the given circuit. An extra pin input is added to the circuit's inputs to determine which of the two modes is active.

Each test point to be inserted corresponds to a new node in the DAG. In turn, this node corresponds to a function/procedure to be added to the original VHDL specification. Therefore, we have to establish the relationship between the DAG representation and the VHDL specification. Each signal/variable candidate for test-point insertion is identified by a label, that is the name of the hierarchical path and the line number in the modified VHDL specification. The internal signal is identified by the line number in which the corresponding FM should have a test point inserted.

At the VHDL level, control points consist of the logical "OR" or "AND" operations which combine as inputs a given bit of a given signal/variable and some extra control inputs which are added to the circuit inputs. The "OR" ("AND") operation is used to increase the 1-Controllability (0-Controllability) value on the given bit of the given signal/

variable. An Observation point consists of a register which is assigned to the corresponding bit of the given signal/variable.

In the following, we show first how to insert test points on signals/variables in the VHDL specification and next on the internal signals of FMs. The algorithm used for selecting test points is presented after this section.

3.2.6.1 Test points insertion on signals/variables

In this section, we show how to insert Control and Observation points on bits of signals/variables in the VHDL specification. Functions (procedures) are used to insert control (observation) points. The functions/procedures are overloaded for different signal types and new parameters. The number of parameters depends on the data type of the corresponding signal/variable which can be either an integer, array of bits or a single bit.

Control-point insertion

Figure 3.6 shows an example of a VHDL specification. It is composed of one unclocked process with three sequential signal assignments. Suppose that we want to insert three control points on signals Z1, Z2 and Z3 which are declared as three different data types. Assume that we want to increase the 0-Controllability on single bit Z1, to increase the 1-Controllability on bit position 2 of signal Z2 and to increase the 1-Controllability on bit position 0 of signal Z3. The modified VHDL specification including the required control points is shown in Figure 3.7. To insert a control point, we use the same name of the function which is called **Insert_Control_Point(...)** and is defined in Figure 3.7. For signal Z1, which is declared as a single bit (std_logic), we need as function parameters the expression assigned to the signal, the type of the control point to insert

("AND" operation), an extra control input (TEST_IN_1)[1] and the signal Test_Mode which is used to switch from the normal mode to the test mode and vice-versa. The "AND" operation combines the output bit of the expression which is assigned to signal Z1 and the control input TEST_IN_1 and the signal Test_Mode. Although, the function is defined in the architecture, but usually it would be placed in a package. The constant signal CONT_TYPE declared as an enumerated type is used to select between the control point type.

The same overloaded function name is used to insert a control point on signals Z2 and Z3. Signal Z2 is declared as an array of bits for which we want to increase the 1-Controllability on bit position 2. In this case, we add another parameter, POSITION, that gives the position of the bit in the array whose Controllability is to be modified. The corresponding function definition is given in Figure 3.7. Now, we want to increase the 1-Controllability on bit position 0 of Signal Z3. The corresponding function converts the corresponding signal to a bit vector, inserts a control point on bit 0 and then converts the bit vector back to the integer type. In this case, we add a new parameter to the function, SIZE, which is the number of bits required to encode signal Z3. Note here, we need to translate signal Z3 into a bit vector and back by using the functions conv_std_logic_vector(...) and conv_integer(...) which are defined in the standard packages of IEEE Standard. Figure 3.8 shows the resulting gate-level circuit obtained by Design Compiler. The required Control points are included in the gate-level circuit obtained after synthesis.

Observation-point insertion

1. This signal can come from a register which will be included in the scan chain.

Figure 3.9 shows another VHDL example. It consists of one unclocked process with two sequential assignment statements. Assume now that we want to increase the Observability value on bit position 2 of the assigned variable V. An Observation point is thus required on this bit position. An Observation point consists of assigning a register to this specific bit position. The modified VHDL specification is shown in Figure 3.10. A procedure called **Insert_Observation_Point(...)** is used to achieve that at the VHDL level. Bit 2 of V is first transferred to an internal signal S. Next, signal S is transferred to a signal SCAN_OUT which is assigned in a clocked process in order to infer a register. Note that the same procedure name is again overloaded to observe any bit of any data type signal/ variable. Figure 3.11 shows the resulting gate-level circuit obtained after synthesis by Design Compiler. It is shown that bit 2 of variable V is assigned to a register (D-FF).

```
                                    ... /...
entity EXAMPLE is
 port(A, B       : in integer range 0 to 7;
    C, D         : in std_logic;
    E, F         : in std_logic_vector(3 downto 0);
    Z1           : out std_logic;
    Z2           : out std_logic_vector(3 downto 0);
    Z3           : out integer range 0 to 7
    );
end EXAMPLE;

architecture RTL of EXAMPLE is
begin
 process(A, B, C, D, E, F)
 begin
  -- Increase the 0-Controllability on signal Z1
  Z1 <= C xor D;
  -- Increase the 1-Controllability on bit position 2 of signal Z2
  Z2 <= E and F;
  -- Increase the 1-Controllability on bit position 0 of signal Z3
  Z3 <= A + B;
 end process;
end RTL;
```

Figure 3.6 VHDL Example for Controllability test-point insertion.

```
                    ...
Type CONTROL_POINT is (OR_POINT, AND_POINT);

entity EXAMPLE is
 port(A, B   : in integer range 0 to 7;
    C, D     : in std_logic;
    E, F     : in std_logic_vector(3 downto 0);
    -- The extra control points
    TEST_IN_1, TEST_IN_2, TEST_IN_3: in std_logic;
    -- Signal to switch between the test and the normal mode
    Test_Mode : in STD_LOGIC;
    Z1        : out std_logic;
    Z2        : out std_logic_vector(3 downto 0);
    Z3        : out  integer range 0 to 7);
end EXAMPLE;

architecture RTL of EXAMPLE is
-- Control point  insertion of a signal/variable
--  declared as STD_LOGIC_VECTOR
 function Insert_Control_Point(
   SIG_VAR          :STD_LOGIC_VECTOR;
   CONT_TYPE        :CONTROL_POINT;
   POSITION         :INTEGER;
   TEST_IN          :STD_LOGIC;
   TM          :STD_LOGIC) return STD_LOGIC_VECTOR  is

variable V: std_logic_vector(SIG_VAR'left downto SIG_VAR'right);
 begin
  V := SIG_VAR;
 if (CONT_TYPE = OR_POINT) then
       V(POSITION) := (TM and TEST_IN) or V(POSITION);
 else
       V(POSITION) := (not TM or TEST_IN) and V(POSITION);
 end if;
 return (V);
 end Insert_Control_Point;

-- Control point  insertion of a signal/variable
--  declared as INTEGER
 function Insert_Control_Point(
   SIG_VAR    :INTEGER;
   SIZE       :INTEGER;
   CONT_TYPE  :CONTROL_POINT;
   POSITION   :INTEGER;
   TEST_IN    :STD_LOGIC;
   TM         :STD_LOGIC)  return INTEGER is
  variable V: std_logic_vector(SIZE-1 downto 0);
  variable V_INT: integer;
                        .../...
 begin
  V:= conv_std_logic_vector(SIG_VAR,  SIZE);
  if (CONT_TYPE = OR_POINT) then
     V(POSITION) := (TM and TEST_IN) or V(position);
  else
     V(POSITION) := (not TM or TEST_IN) and V(POSITION);
  end if;
  V_INT := CONV_integer(V);
  return (V_INT);
 end Insert_Control_Point;

-- Control point  insertion of a signal/variable
--  declared as STD_LOGIC
function Insert_Control_Point(
   SIG_VAR    :STD_LOGIC;
   CONT_TYPE  :CONTROL_POINT;
   TEST_IN    :STD_LOGIC;
   TM         :STD_LOGIC) return STD_LOGIC is
  variable V     : STD_LOGIC;
 begin
  V := SIG_VAR;
  if (CONT_TYPE = OR_POINT) then
    V := (TM and TEST_IN) or V;
  else
    V := (not TM or TEST_IN) and V;
  end if;
  return (V);
 end Insert_Control_Point;
-- End of function definition
begin

process(A, B, C, D, E, F, TEST_IN_1, TEST_IN_2,
                                    TEST_IN_3, Test_Mode)
 begin
  Z1  <= Insert_Control_Point((C xor D), AND_POINT,
                                    TEST_IN_1, Test_Mode);

  Z2  <= Insert_Control_Point((E and F), OR_POINT,
                                    2, TEST_IN_2, Test_mode);

  Z3  <= Insert_Control_Point((A + B), 3, OR_POINT,
                                    0, TEST_IN_3, Test_Mode);
 end process;
end RTL;
```

Figure 3.7 Modified VHDL specification including three control points.

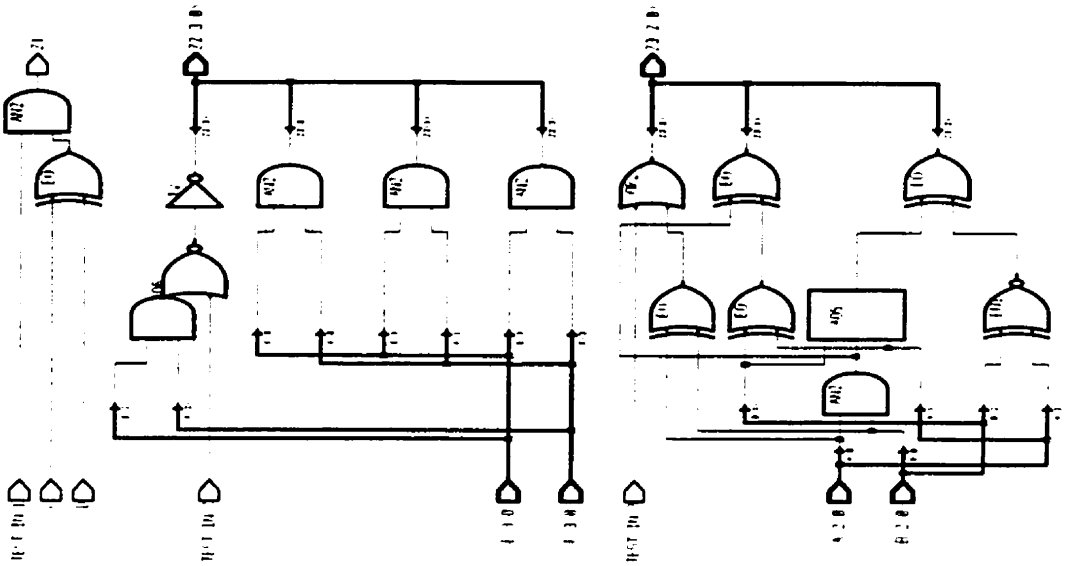

Figure 3.8 The resulting gate-level circuit including the three Control points.

```
                                    ... / ...
entity EXAMPLE is
 port (A, B, C          : in std_logic_vector(3 downto 0);
       Z                : out std_logic_vector(3 downto 0));
end EXAMPLE;

architecture RTL of EXAMPLE is
begin
 process(A,B,C)
 variable V     : std_logic_vector(3 downto 0);
 begin
   -- Insert an Observation point on bit position 2 of assigned variable V
    V := A and B;
    Z <= V xor C;
 end process;
end RTL;
```

Figure 3.9 VHDL example for Observability test-point insertion.

```
                    . . .
entity EXAMPLE is
 port (A, B, C        : in std_logic_vector(3 downto 0);
       Z              : out std_logic_vector(3 downto 0);
       CLK            : in Bit;
    -- Signal used for observation
    SCAN_OUT          : out std_logic);
end EXAMPLE;

architecture RTL of EXAMPLE is
-- Observation point insertion of a signal/variable
-- declared as STD_LOGIC_VECTOR
  Procedure Insert_Observation_Point(
   SIG_VAR     :STD_LOGIC_VECTOR;
   POSITION    :INTEGER;
   signal SCAN :out STD_LOGIC ) is
 begin
  SCAN <= SIG_VAR(POSITION);
 end Insert_Observation_Point;

 signal S     : std_logic;
                . . /. .
```

```
begin
 process(A, B, C)
 variable V : std_logic_vector(3 downto 0);
 begin
   V := A + B;
  -- Insert an observation point at bit 2 of variable V
  Insert_Observation_Point(V, 2, S);
  Z <= V + C;
 end process;

-- This clocked process is used to infer registers
-- for observation points
 process
 begin
  wait until CLK = '1';
  SCAN_OUT <= S;
 end process;
end RTL;
```

Figure 3.10 Modified VHDL specification including one Observation point.

3.2.6.2 Test-point insertion on internal signals of FMs

In the previous section, we only analyze the testability of signals/variables that are explicitly declared or may be implied in the VHDL specification. These signals/variables are used to connect VHDL operators which are mapped to FMs (adders, comparators, etc.). As well known, large multi-bit FMs can be difficult to test by random patterns due to a low Controllability et/ou Observability on internal signals. These signals such as carry lines inside an adder do not have a direct correspondence in the VHDL specification for test-point insertion as we did in the previous examples. In the following, we show how to insert test points at the RTL on the internal signals of FMs.

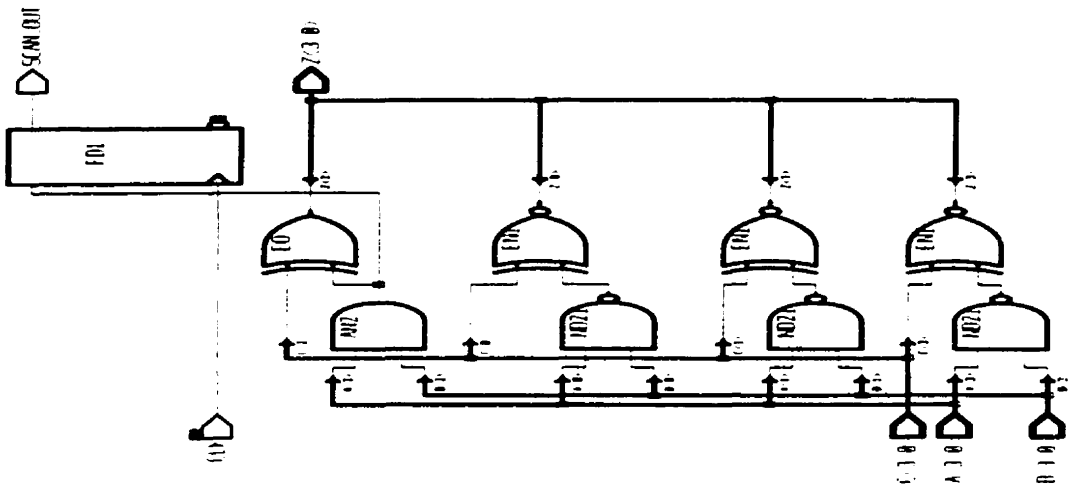

Figure 3.11 The resulting gate-level circuit including one observation point.

Comparators

Let's consider as an example, the equality comparator to illustrate how to insert test points on internal signals. An n-bit equality comparator can be decomposed functionally into n 1-bit equality comparators in which the corresponding outputs are ANDed as shown in Figure 3.12. It is known that the n-inputs AND operation is difficult to test with random testing when the number inputs increases. One solution, is to decompose the n-inputs AND operation into a cascaded of (n-1) two-input AND operations as shown in Figure 3.13. The signals shown as dotted lines are considered as internal signals. These signals are analyzed for testability and may require test points (Control or Observation points) on

them at the RTL. Here also, we defined a function (procedure) which inserts a given number of control (observation) points at some specified internal signals of an equality comparator. Its use will be illustrated next on an example.

Figure 3.14 shows a VHDL specification of a 16-bit counter in which a 16-bit equality comparator is used in the if expression. We want to insert two control points at two different internal signals of the comparator. In this example, we want to increase the 1-Controllability on both signals. Each internal signal is characterized by a position in the equality comparator structure. For example, the signal P_i as shown in Figure 3.13, corresponds to an i-bit comparator and is assigned the i^{th} position in the comparator structure (starting from zero). This information is used as a parameter in the function (procedure) used to insert a control (observation) point at a specific internal signal, P_i. The modified VHDL specification including the required control points at positions 5 and 10, is shown in Figure 3.15. The function called **Insert_Control_Equal(...)** is used to insert a given number of control points at some internal signals of the equality comparator. To insert control points, the corresponding function takes the following parameters: The two operands of the comparison L and R, the number of control points to insert N, the corresponding positions of the control points represented by the array P, the type of the control points (Increasing the 1-Controllability or the 0-Controllability), the required extra control inputs as an array Sig_array and the signal Test_Mode. Note that this functions uses the function **Insert_Control_Point(...)** defined previously. These functions are defined in the package **"Test_Points"** which is included in the specification as shown in Figure 3.15. In the same manner, we define a procedure called **Insert_Observation_Equal(...)** used to insert a given number of observation points on some internal signals of the equality comparator. We use the same VHDL example (Figure 3.14) and show how to insert two observation points at two internal signals at positions 5 and 10. The modified VHDL specification is

shown in Figure 3.16. Note that the function/procedure is again overloaded for comparators operating on other data types.

We can use a similar method as for the equality comparator to insert test points on internal signals of the other comparator types (<, >, <=, >=). For instance, the logic equation of an n-bit less than comparator is given in Equation (10) in Appendix 1. This comparator requires an n-inputs OR operation which can be decomposed into a cascaded of (n-1) two-input OR operations.

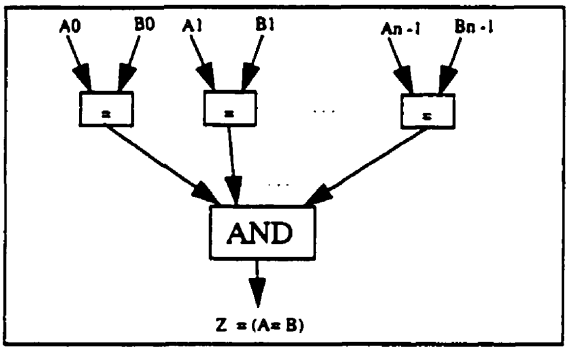

Figure 3.12 The structure of an n-bit equality comparator.

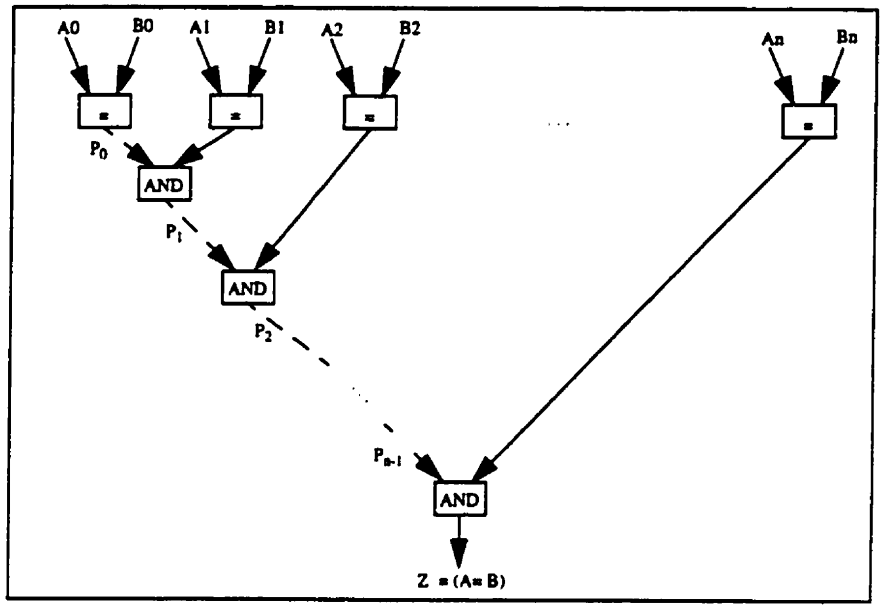

Figure 3.13 The internal signals considered for test-point insertion of an n-bit equality comparator.

```
entity COMP is
 port(IN_COUNT  : in STD_LOGIC_VECTOR(15 downto 0);
    CLOCK       : in Bit;
    OUT_COUNT : out std_logic_vector(15 downto 0)
    );
end COMP;

architecture RTL of COMP is
begin
 process
 begin
  wait until CLOCK = '1';
  if (IN_COUNT = "1111111111111111") then
       OUT_COUNT <= "0000000000000000";
  else
      OUT_COUNT <= IN_COUNT + "0000000000000001";
  end if;
 end Process;
end RTL;
```

Figure 3.14 A VHDL specification of a 16-bit counter.

```
                                    . . .
use WORK.Test_Points.all;

entity COMP is
 port(IN_COUNT                    : in STD_LOGIC_VECTOR(15 downto 0);
     CLOCK                        : in Bit;
     TEST_IN_1, TEST_IN_2         : in BOOLEAN;
     Test_Mode                    : in STD_LOGIC;
     OUT_COUNT                    : out std_logic_vector(15 downto 0)
     );
end COMP;

architecture RTL of COMP is
type Position_array is array(INTEGER range <>) of INTEGER;
type Cont_type_array is array(INTEGER range <>) of CONTROL_POINT;
type Sig_array is array(INTEGER range <>) of BOOLEAN;

function unsigned_Insert_Control_Equal(L,R :UNSIGNED; N:INTEGER; P:POSITION_ARRAY;
CONT_TYPE:Cont_TYPE_ARRAY; TEST_IN:SIG_ARRAY;TM : STD_LOGIC ) return BOOLEAN is
 variable V:BOOLEAN;
begin
 V := (L(L'left downto (P(0)+1+L'right)) = (R'left downto (P(0)+1+R'right)));
For i in (N-1) downto 0 loop
  if (i = N-1) then
    V := V and (Insert_Control_Point((L((P(i) + L'right) downto L'right) =
                                    R((P(i) + R'right) downto R'right)), CONT_TYPE(i), TEST_IN(i), TM));
  else
    V := V and (Insert_Control_Point((L((P(i) + L'right) downto (P(i+1) + 1 +L'right)) =
                                    R((P(i) + R'right) downto (P(i+1) + R'right + 1))),CONT_TYPE(i), TEST_IN(i), TM));
 end if;
 end loop;
 return V;
end unsigned_Insert_Control_Equal;

function Insert_Control_Equal(L,R: STD_LOGIC_VECTOR; N: INTEGER; P: POSITION_ARRAY;
CONT_TYPE: Cont_TYPE_ARRAY; TEST_IN: SIG_ARRAY;TM: STD_LOGIC) return BOOLEAN is
begin
 return unsigned_Insert_Control_Equal(UNSIGNED(L), UNSIGNED(R), N, P, CONT_TYPE, TEST_IN, TM);
end Insert_Control_Equal;

begin
 process
 begin
  wait until CLOCK = '1';
 if (Insert_Control_Equal IN_COUNT, "1111111111111111", 2, (10, 5),(OR_POINT, OR_POINT),
                          (TEST_IN_1,TEST_IN_2), Test_Mode)) then
   OUT_COUNT <= "0000000000000000";
 else
   OUT_COUNT <= IN_COUNT + "0000000000000001";
  end if;
 end Process;
end RTL;
```

Figure 3.15 Modified VHDL specification of a 16-bit counter: Control point insertion on the internal signals of an equality comparator.

```
                                                    . . .
use WORK.Test_Points.all;

entity COMP  is
 port(IN_COUNT          : in STD_LOGIC_VECTOR(15 downto 0);
     CLOCK              : in Bit;
     OUT_COUNT          : out std_logic_vector(15 downto 0);
     SCAN               : out Sig_array(1 downto 0)
     );
end COMP;

architecture RTL of  COMP is
 Procedure Insert_Observation_Equal(L, R: STD_LOGIC_VECTOR; variable V: out BOOLEAN; N: INTEGER;
   P :Position_array; signal SCAN   :out Sig_array) is
 variable V0,V1:BOOLEAN;
 begin
 V0 := (L(L'left downto (P(0)+1+L'right)) = R(R'left downto (P(0)+1+R'right)));
 For i in 0 to (N-1) loop
    if (i = N-1) then
       V1 := (L((P(i)+L'right) downto L'right) = R((P(i)+R'right) downto R'right));
       SCAN(i) <= V1;
       V0 := V0 and V1;
    else
       V1 := (L((P(i) + L'right) downto (P(i+1) + 1 +L'right)) = R((P(i) + R'right) downto (P(i+1) + R'right + 1)));
       SCAN(i) <= V1;
      V0 := V0 and V1;
   end if;
end loop;
   V := V0;
 end Insert_Observation_Equal;

begin
process
   variable VAR  : BOOLEAN;
 begin
   wait until CLOCK = '1';
   VAR :=(IN_COUNT = "1111111111111111");
   Insert_Observation_Equal(IN_COUNT, "1111111111111111", VAR,2,(10, 5),SCAN);
   if (VAR ) then
    OUT_COUNT <= "0000000000000000";
   else
    OUT_COUNT <= IN_COUNT + "0000000000000001";
   end if;
 end Process;
end RTL;
```

Figure 3.16 Modified VHDL specification of a 16-bit counter: Observation point insertion on the internal signals of an equality comparator.

Adder

In an adder, the carry out lines (Figure 3.5) are the internal signals to consider for test-point insertion. Test-point insertion inside multiplier FMs is implemented based only on the shift-add structure. Other implementations of the multiplier are currently under investigation.

Multiplexers

Multiplexers are inferred by conditional statements ("if" and "case") in the VHDL specification. Figure 3.17(a) shows a VHDL specification containing a "case" statement. The corresponding DAG representation is given in Figure 3.17(b). A 4:1 multiplexer (mux) can be difficult to test when the number of inputs (i.e., the number of cases in the "case" statement) increases, as usually is the case in VHDL designs. This difficulty comes from the internal signals that are not visible at the VHDL level. Figure 3.18, shows one possible representation of a 4:1 mux by means of 2:1 muxes. The internal signals are shown as dotted lines. As shown in Figure 3.18, test points inserted on the internal signals do not have a direct correspondence in the VHDL specification. One solution is to decompose the large case statement into smaller nested case statements. However, this may be difficult and can dramatically change the original VHDL code. Another solution consists of using the inputs and the outputs of the 4:1 mux to improve Controllability and Observability values of the internal signals. This can be increased in different ways. We can increase the Observability of the mux output which is visible in the VHDL specification (Figure 3.18). We thus insert an Observation point on variable V which is the output of the corresponding 4:1 mux. We use the same VHDL procedure defined previously to insert an Observation point (Section 3.2.6). This corresponds to adding the procedure **Insert_Observation_Point(...)** just after the line corresponding to the end "case" statement in Figure 3.17(a). Note that the proposed algorithm identifies each "if" and "case"

statement and their corresponding end scope (i.e., "end-if" and "end- case" statements). The modified VHDL specification including the observation point is shown in Figure 3.19. A clocked process is used to infer a register on variable V. Note that we can increase the Controllability of the control inputs or we can increase the Controllability of data inputs by using the function **Insert_Control_Point(...)** defined previously (Section 3.2.6).

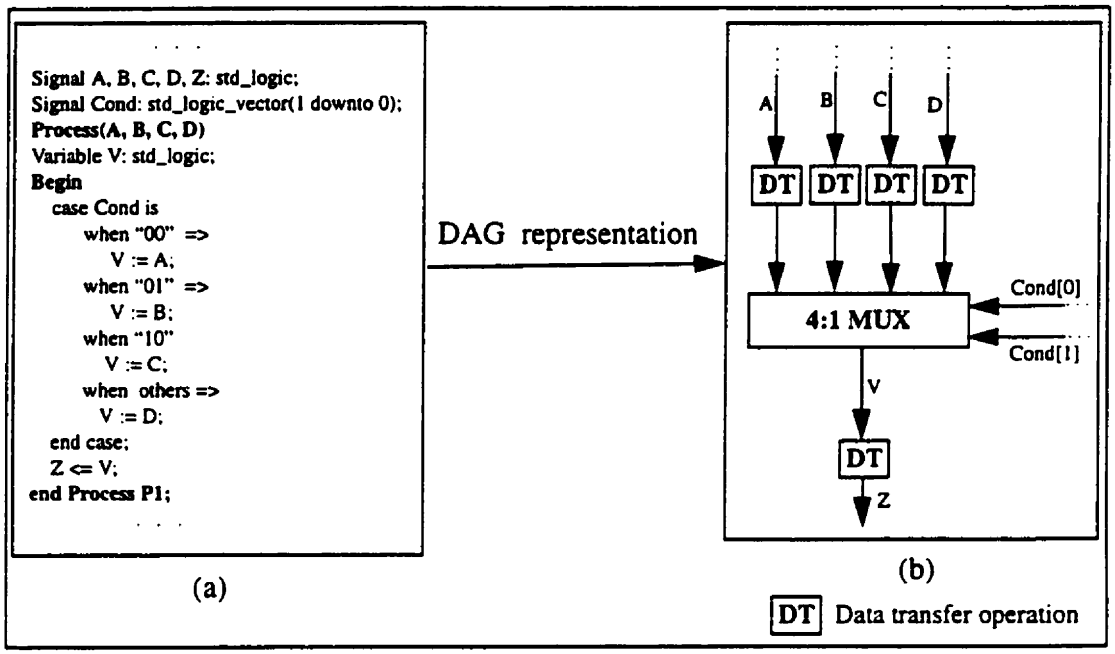

Figure 3.17 DAG representation for a "case" statement.

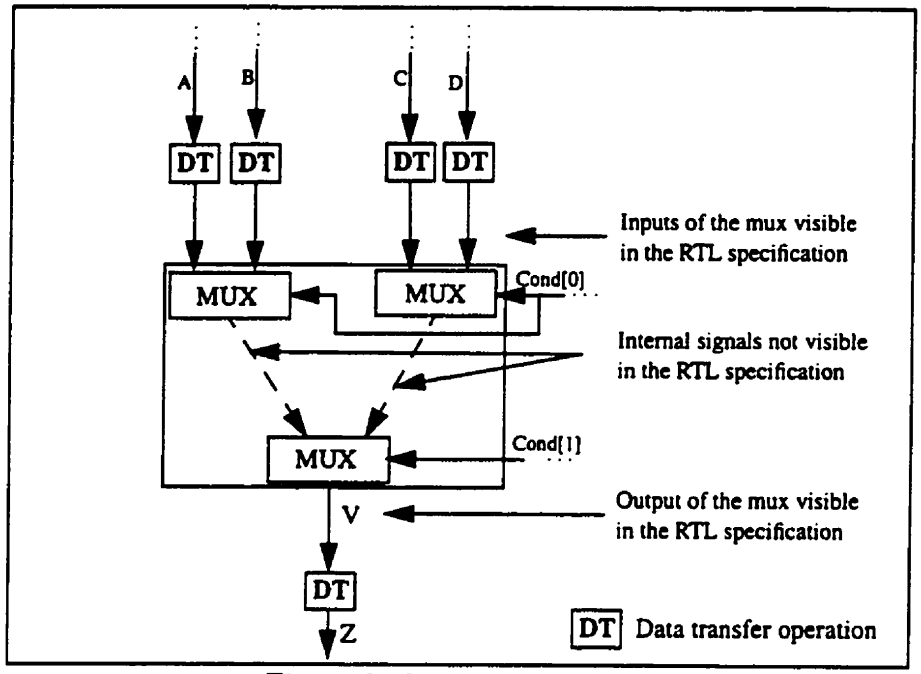

Figure 3.18 Internal signals of a 4:1 mux.

3.2.6.3 Test point selection algorithm

In this section, we present the greedy algorithm used for selecting test points (Control and Observation points) [147, 50]. However, any more efficient test-point insertion method can be used to select the best locations for insertion. Before we describe the algorithm, we give some definitions.

Definition [119]:

The Detectability of a fault in a single-bit signal S is defined as follows:

When S is stuck-at 1, then we have:

$$D_1(S) = (1 - C_1(S)) \times O(S) \tag{3.6}$$

Similarly, if S is stuck-at-0, then we have:

```
                . . .
Procedure Insert_Observation_Point(
    SIG_VAR     :STD_LOGIC;
    signal SCAN  :out STD_LOGIC ) is
 begin
   SCAN <= SIG_VAR;
 end Insert_Observation_Point;

Signal A, B, C, D, Z, SCAN: std_logic;
Signal Cond: std_logic_vector(1 downto 0);

Process(A, B, C, D)
Variable V: std_logic;
Begin
   case Cond is
       when "00" =>
          V := A;
       when "01" =>
          V := B;
       when "10"
          V := C;
       when  others =>
          V := D;
   end case;
    Insert_Observation_Point(V, SCAN);
   Z <= V;
end Process P1;

-- This clocked process is used to infer registers for observation points
 process
 begin
   wait until CLK = '1';
   SCAN_OUT <= SCAN;
 end process;
                . . .
```

Figure 3.19 Modified VHDL specification with one Observation point to enhance the Detectability of the internal signals of a mux.

$$D_0(S) = C_1(S) \times O(S) \tag{3.7}$$

Definition 5:

We define a VHDL specification as random testable if each bit of signal/variable and of the internal signals of FMs has a Detectability value above a given threshold Dth.

The value of Dth comes from experience and depends on the desired fault coverage, test length, and circuit complexity. Low detectability below the Dth value indicates poten-

tial Controllability or Observability problems that may negatively impact the length of the test sequence of the resulting circuit.

The test-point insertion process starts with the calculation of the Detectability value of each bit of each signal/variable in the VHDL specification and the internal signals of FMs. Those signals/variables with Detectability values below a Dth value, are candidates for test point insertion. Among them we first select those candidate signals/variables that have Controllability values below the Controllability threshold. Those nearest to the primary or the pseudo-primary inputs are selected; this is because control point insertion will affect the Controllability values not only of the insertion point, but also of all the signals/variables driven by it. Once the insertion is done, the Controllability values are recomputed, and the process is repeated until all Controllability values are above the threshold.

Next, the analysis considers Observability values. Decisions to improve Observability are deferred because a change in Controllability may affect Observability, but a change in Observability has no effect on Controllability. If control points are inserted, the Observability values are recomputed on all bits of all signals/variables and internal signals. Those with Observability value below the Observability threshold are candidates for observation point insertion; among them, the one closest to primary and pseudo-primary outputs are selected first. Once observation point insertion is done, the Observability values are recomputed, and the process is repeated until all Observability values are above the threshold. Note here, that constant signals/variables are not considered for test-point insertion candidates since they are optimized and propagated by the synthesis tool.

3.2.7 Experimental results

In this section, we present the experimental results we obtained on three benchmark circuits which are random pattern resistant. Circuits C1 and C2 (Table 3.1) are some design blocks in an I/O chip [94] that were designed at the RTL and synthesized by Synopsys tools to the gate-level. Circuit C3 is 32-bit counter. All circuits contain both control and data operations. The largest circuit is C2 which consists of about 1200 VHDL lines. Each circuit is synthesized to the gate-level before and after the test-point insertion. A random ATPG tool is used to evaluate the fault coverage of the gate level circuit with the full scan option before and after the test-point insertion. For each circuit we fix the Detectability threshold value depending on the circuit complexity and the test length, and then determine the required test points. For the three considered circuit, we fix the Detectability threshold value as 0.001, the Controllability threshold value as 0.01 and the Observability threshold value as 0.001. The circuits were synthesized under various area and delay constraints. The delay is the worst case propagation path in the resulting gate-level circuit.

Figure 3.20 shows the VHDL specification of circuit C1 which is used to illustrate the method. It consists of two processes (a clocked one and a combinational one). A function called BIT_CNT_INC(...) is used in the clocked process and defined in the architecture. This function is expanded in line and the for...loop statement used inside this function is unrolled. The testability analysis of this circuit yields 5 test points (4 Control points and one Observation point). The test required points increase the 1-Controllability value of the variable TOGGLE at the 7^{th} and the 13^{th} iteration of the for...loop, and the Observability value of the variable TOGGLE at the 6^{th} iteration, and the 1-Controllability value of two internal signals of the equality comparator. The latter is used in the "if" condition of the unclocked process. The modified VHDL specification is shown in Figure 3.21. We used

the previously defined function/procedure as presented in Section 3.2.6. They are defined in the package "Test_Points" which is included in Figure 3.21. Note that for this example, the variable TOGGLE is observed through the signal BIT_CNT_REG(16) which is assigned in a clocked process. This is because we cannot use the previously defined procedure for observation point insertion inside a function.

Tables 3.2 and Table 3.3 show the results obtained under two different area and delay constraints. They are identified as Optimization 1 (Table 3.2) and Optimization 2 (Table 3.3). In each case, we performed a fault simulation of 32K random patterns before and after the test-point insertion. In Tables 3.2 and 3.3 we show the following quantities: the resulting fault coverage after applying 32K random patterns, the number of test points (Control/Observation points) added to the VHDL specification, the total cell area, the delay of the critical path incurred by the test-point insertion, the corresponding percentage area, delay and the total number of faults over the number of redundant faults. Note that each unit of the total cell area corresponds approximately to a two-input-Nand gate.

Considering the results of Table 3.2 and Table 3.3, we can see that, first, the insertion of a small number of test points leads to an increased fault coverage after applying 32K random patterns. This indicates that our method provides good analysis of testability, even though it is carried out at the RTL, while the coverage analysis is done at the gate level. For example, in the case of C1 (Figure 3.20), inserting the 5 test points improves the fault coverage from 95.48% to 100% (86.98% to 99.76%) for the case of Optimization 1 (Optimization 2). Similarly, remarkable improvements can be observed for the other two circuits. The area overheads range from 3% to 10%. However, the delays actually improved after test-point insertion for C1 and C2 as shown in Table 3.2, and for C2 as shown in Table 3.3. This indicates that the insertion does not necessarily imply an increase in area

and delay if carried out at the VHDL source code. The synthesis tool optimizes concurrentely the inserted test points and the functional logic within the design constraints (delay and area). This is one of the main advantages of inserting testability at the RTL before synthesis. Designers have complete control of the overall optimization process and the test points remain in an integral part of the RTL specification.

Note that we can still obtain an improvement by modifying the optimization constraints. Circuit C2 is the highest random pattern resistant circuit for which we obtained very good results in terms of delay at the expense of a small increase in area (approximately 3%). The fault coverage increased approximately by 15% for this circuit. Note that the circuit required 10 test points inside the internal signals of some of the equality comparators used in the VHDL specification. Figure 3.22 shows a portion of the RTL description of this circuit before and after test-point insertion.

Another important criterion characterizing random pattern testability of a circuit is the test length required to achieve a certain fault coverage. For each of the circuits considered, Table 3.4 lists the test length which is necessary to obtain a certain fault coverage before and after test-point insertion. We consider only the constraints obtained by Optimization 1. It can be seen that the test length can be reduced by several orders of magnitude with very few test points. Since, the test length is proportional to the time required to perform the self-test, a similar reduction in the test time is achieved. We can notice that for C2 which is highly random test resistant, 95% fault coverage is achieved with 19 test points. The maximum fault coverage before test-point insertion is less than 80% for 10^7

random patterns, while we achieve 95% after test-point insertion with only 2×10^5 random patterns.

Table 3.1 Circuit information

Circuit	No. of inputs	No. of outputs	No. of Flip-Flops
C1	3	17	16
C2	21	50	31
C3	33	32	32

Table 3.2 Experimental results with Optimization 1

Circuit	Fault coverage [%] after 32 K random patterns		Number of Test Points	Total cell area			Delay			No. of faults/No. of Redundant faults	
	Before insertion	After insertion	Contr. P/ Obs. P	Before insertion	After insertion	Ovhd. [%]	Before insertion	After insertion	Ovhd. [%]	Before insertion	After insertion
C1	95.48	100	4/1	250	266	6.40	12.85	12.65	-1.58	790/0	888/0
C2	76.20	90.40	14/5	967	996	3.00	46.75	43.43	-7.10	3760/0	4155/0
C3	87.81	92.75	5/0	451	460	1.99	17.81	17.81	0.00	1186/52	1362/0

Table 3.3 Experimental results with Optimization 2

Circuit	Fault coverage [%] after 32 K random patterns		Number of Test Points	Total cell area			Delay			No. of faults/No. of Redundant faults	
	Before insertion	After insertion	Contr. P/ Obs. P	Before insertion	After insertion	Ovhd. [%]	Before insertion	After insertion	Ovhd. [%]	Before insertion	After insertion
C1	86.98	99.76	4/1	399	439	10.02	10.59	11.67	10.19	1588/0	1799/0
C2	75.56	89.79	14/5	930	964	3.65	46.29	42.13	-9.98	3632/0	3780/0
C3	84.60	93.48	5/0	473	509	7.61	17.48	17.48	0.00	1404/32	1580/0

Table 3.4 Experimental results: Comparison of test length with Optimization 1

Circuit	Before insertion		After insertion	
	Test length	Max fault coverage	Test length	Max fault coverage
C1	800,000	100	8,000	100
C2	> 10,000,000	80.00	200,000	95.00
C3	> 10, 000, 000	91.31	200,000	94.62

```
use WORK.Test_Points.all;

entity Circuit_C1 is
 port(
    BIST_CLK      : in  STD_LOGIC;
    CNT_EN        : in  STD_LOGIC;
    CNT_INIT      : in  STD_LOGIC;
    LAST_BIT_CNT  : out STD_LOGIC;
    BIT_CNT       : out STD_LOGIC_VECTOR(15 downto 0)
    );
end Circuit_C1;

architecture RTL of Circuit_C1 is

 signal BIT_CNT_MAX : STD_LOGIC_VECTOR(15 downto 0);
 signal RESET       : STD_LOGIC;
 signal BIT_CNT_REG : STD_LOGIC_VECTOR ( 15 downto 0);

function BIT_CNT_INC
     ( BIT_CNT : STD_LOGIC_VECTOR(15 downto 0 ))
return STD_LOGIC_VECTOR IS
  variable TOGGLE : STD_LOGIC;
  variable i      : INTEGER;
variable BIT_CNT_INC_VAL : STD_LOGIC_VECTOR(15 downto 0);

 begin
 BIT_CNT_INC_VAL(0) := not BIT_CNT(0);
 TOGGLE := '1';
for i in 1 to 15 loop
  TOGGLE := BIT_CNT(i-1) and TOGGLE;
  BIT_CNT_INC_VAL(i) := BIT_CNT(i) xor TOGGLE;
end loop;
 return BIT_CNT_INC_VAL;
 end BIT_CNT_INC;
                          .../...
begin
 BIT_CNT_MAX <= "1111111111111111";
 RESET <= CNT_INIT;

 process (BIST_CLK)
 begin
 if (BIST_CLK'event and BIST_CLK = '1') then
     if (RESET = '1') then
           BIT_CNT_REG <= "0000000000000000";
     else
       if (CNT_EN = '1') then
          BIT_CNT_REG <= BIT_CNT_INC(BIT_CNT_REG);
       end if;
     end if;
 end if;
end process;

Bit_CNT <= BIT_CNT_REG;

process(BIT_CNT_REG)
 begin
  if (BIT_CNT_REG = BIT_CNT_MAX) then
   LAST_BIT_CNT <= '1';
  else
   LAST_BIT_CNT <= '0';
  end if;
 end process;
end RTL;
```

Figure 3.20 VHDL specification of circuit C1 before modification.

```
use WORK.Test_Points.all;

entity Circuit_C1 is
 port(
  BIST_CLK      : in  STD_LOGIC;
  CNT_EN        : in  STD_LOGIC;
  CNT_INIT      : in  STD_LOGIC;
-- The extra control input used for control insertion
  TEST_IN_1     : in  STD_LOGIC;
  TEST_IN_2     : in  STD_LOGIC;
  TEST_IN_3     : in BOOLEAN;
  TEST_IN_4     : in BOOLEAN;
  Test_Mode     : in STD_LOGIC;
  LAST_BIT_CNT  : out STD_LOGIC;
  BIT_CNT  : out STD_LOGIC_VECTOR(16 downto 0));
end Circuit_C1;

architecture RTL of Circuit_C1 is
 signal BIT_CNT_MAX : STD_LOGIC_VECTOR(15 downto 0);
 signal RESET     : STD_LOGIC;
 signal BIT_CNT_REG : STD_LOGIC_VECTOR (16 downto 0);

 function BIT_CNT_INC
  ( BIT_CNT : STD_LOGIC_VECTOR(16 downto 0);
    TEST_IN_1: STD_LOGIC;TEST_IN_2: STD_LOGIC;
    TM: STD_LOGIC)
  return STD_LOGIC_VECTOR IS
  variable TOGGLE   : STD_LOGIC;
  variable  i       : INTEGER;
  variable BIT_CNT_INC_VAL, VECT_TOGGLE:
                STD_LOGIC_VECTOR(15 downto 0);
 begin
  BIT_CNT_INC_VAL(0) := not BIT_CNT(0);
  TOGGLE := '1';
 for i in 1 to 15 loop
  TOGGLE := BIT_CNT(i-1) and TOGGLE;
  BIT_CNT_INC_VAL(i) := BIT_CNT(i) xor TOGGLE;
  if (i = 7) then
   -- A Control point is inserted on variable TOGGLE
   TOGGLE :=
     Insert_Control_Point(TOGGLE, OR_POINT, TEST_IN_1, TM);
 end if;
 if (i = 13) then
 -- A Control point is inserted on variable TOGGLE
  TOGGLE :=
     Insert_Control_Point(TOGGLE, OR_POINT, TEST_IN_2, TM);
 end if;
 VECT_TOGGLE(i) := TOGGLE;
end loop;
-- One observation point. It is observed in BIT_CNT_REG(16)
 return ((VECT_TOGGLE(6) & BIT_CNT_INC_VAL));
end BIT_CNT_INC;
                        .../...

begin
 BIT_CNT_MAX <= "1111111111111111";
 RESET <= CNT_INIT;

process (BIST_CLK)
 begin
 if (BIST_CLK'event and BIST_CLK = '1') then
  if (RESET = '1') then
       BIT_CNT_REG <= "00000000000000000";
  else
    if (CNT_EN = '1') then
      BIT_CNT_REG <=
        BIT_CNT_INC(BIT_CNT_REG, TEST_IN_1,
                                  TEST_IN_2, Test_Mode);
    end if;
  end if;
end if;
end process;

 BIT_CNT <= BIT_CNT_REG;

process(BIT_CNT_REG, TEST_IN_3,
                    BIT_CNT_MAX, TEST_IN_4)
  variable V: BOOLEAN;
 begin
--Two control points inserted on the internal
-- signals of the comparator

if(Insert_Control_Equal(BIT_CNT_REG(15 downto 0),
    BIT_CNT_MAX, 2,  (12, 6), (OR_POINT,OR_POINT),
          (TEST_IN_3,TEST_IN_4), Test_Mode)) then
   LAST_BIT_CNT <= '1';
else
   LAST_BIT_CNT <= '0';
end if;
end process;
end RTL;
```

Figure 3.21 The modified VHDL specification of circuit C1.

```
...

Process(ADDRESS_CNT_REG)
begin
 if (ADDRESS_CNT_REG(29 downto 0) =
    (not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR)) then

  CLUSTER0_LAST_SUB_CLUSTER_ADDRESS <= '1';
 else
  CLUSTER0_LAST_SUB_CLUSTER_ADDRESS <= '0';
 end if;
end process;
```

(a)

```
...

process(ADDRESS_CNT_REG, TEST_IN_0, TEST_IN_2,
              TEST_IN_3, TEST_IN_4)
begin
 if (Insert_Control_Equal(ADDRESS_CNT_REG(29 downto 0),
    (not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
     not DECR &
  not DECR), 4, (24, 18, 12, 6), (OR_POINT, OR_POINT, OR_POINT,
 OR_POINT), (TEST_IN_0, TEST_IN_1, TEST_IN_2, TEST_IN_3),
 Test_Mode)) then
  CLUSTER0_LAST_SUB_CLUSTER_ADDRESS <= '1';
 else
  CLUSTER0_LAST_SUB_CLUSTER_ADDRESS <= '0';
 end if;
end process;
```

(b)

Figure 3.22 A portion of the RTL VHDL description of circuit C2 targeted for RTL modification. (a) Before modification. (b) After modification

3.2.8 Conclusions

A new testability analysis and test-point insertion method for RTL VHDL designs was proposed. We analyze and modify the VHDL RTL description of the circuit, to generate an easily testable gate-level circuit by a random vectors in a BIST environment. We

identify hard-to-detect bits of signals/variables that are explicitly declared or implied in the VHDL specification. Test-point insertion is carried out again at RTL. Internal signals of FMs are also analyzed and may be modified at the RTL. Test points are defined using a set of VHDL functions and procedures defined in a package. Since the inserted test points are included in the original VHDL code, they become part of the specification before synthesis. This allows full use of RTL synthesis tools to optimize both the functional and the inserted test logic together within the required design constraints (delay and area). In fact, the performances of the gate-level circuit can be the same with or without the insertion of test points. Another advantage of our method when compared to other existing methods [50, 17, 41] is that we can affect each bit of each signal/variable regardless its type (integer, bit vector, single bit). A number of random-pattern-resistant benchmark circuits were used to demonstrate the effectiveness of our method.

CHAPITRE 4

Implantation de l'algorithme et résultats expérimentaux

4.1 Introduction

L'algorithme d'analyse de testabilité et d'insertion de points de test a été implanté en un outil logiciel en utilisant le langage de programmation C. La complexité du logiciel est d'environ 14,000 lignes de code. Dans ce chapitre, on décrit les étapes principales de l'algorithme proposé ainsi que quelques exemples de circuits-test pour illustrer son application. Trois exemples de circuits-test ont été déjà présentés dans le chapitre 3, section 3.2.7.

4.2 Algorithme proposé

La figure 4.1 résume les étapes principales de l'algorithme d'analyse de testabilité et d'insertion de points de test. Ces étapes consistent d'abord à construire un graphe acyclique dirigé (Directed Acyclic Graph - DAG) à partir de la spécification VHDL du circuit. Cette dernière est analysée par un analyseur VHDL afin de générer une représentation intermédiaire appelée le VIF (VHDL Intermediate Format). Le DAG est ensuite utilisé pour calculer et propager les mesures de testabilité. Durant cette étape, on identifie les bits des signaux et des variables difficiles à contrôler et/ou à observer. Un algorithme de sélection de points de test est utilisé pour sélectionner un ensemble de points de contrôle et d'observation. Ces derniers sont ensuite insérés au niveau de la spécification VHDL.

Enfin, la spécification VHDL résultante est optimisée au niveau portes logiques par l'outil de synthèse de Synopsys. Le circuit obtenu au niveau portes logiques devient plus facilement testable par un ATPG dans un contexte de test aléatoire. Dans les sections suivantes, nous allons décrire en détails la fonction et l'implantation de chacune des étapes principales de l'algorithme proposé.

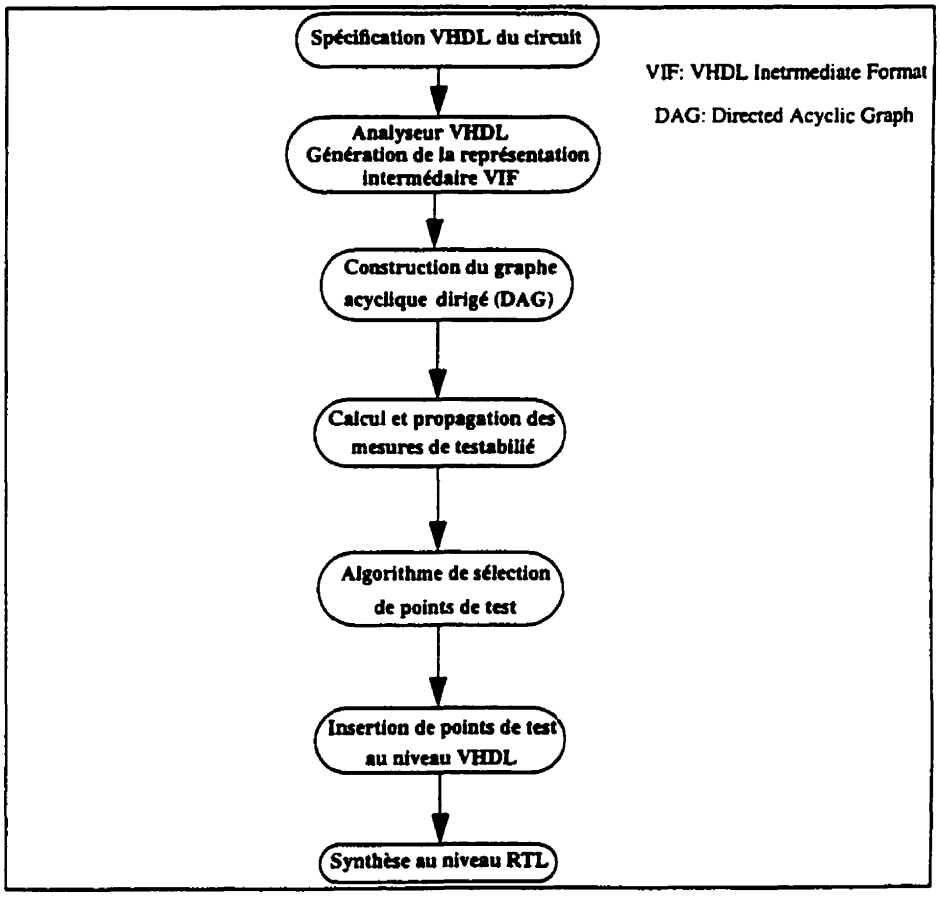

Figure 4.1 Étapes principales de l'algorithme d'analyse de testabilité et d'insertion de points de test.

4.2.1 La représentation intermédiaire VIF

La première étape de l'algorithme consiste à analyser la spécification VHDL du circuit dans le but de générer la représentation intermédiaire VIF. On a utilisé les outils de LEDA [48] pour générer la représentation VIF, accessible à partir d'une interface procédurale codée en langage C. L'analyseur de VHDL supporte toutes les structures VHDL'93 et VHDL'87 du standard IEEE (1076) [137]. Un browser est aussi intégré dans les outils de LEDA afin d'examiner la représentation VIF.

Au niveau conceptuel, la représentation VIF est représentée sous forme d'une structure d'arbre. Chaque noeud de l'arbre possède des attributs qui peuvent être soit une valeur primitive (entier, réel, chaîne de caractères, etc.), soit des liaisons simples à d'autres noeuds de l'arbre, soit une liste de liaisons à d'autres noeuds. Les figures 4.2 et 4.3 montrent un exemple d'une représentation VIF d'une spécification VHDL de la machine de Moore (la spécification VHDL de la machine de Moore est présentée dans la figure 3.2.3, chapitre 3). À l'aide du browser des outils de LEDA, on peut examiner l'organisation de l'information de la représentation VIF ainsi que la manière d'y accéder à cette information. Pour y accéder aux instructions séquentielles d'un processus VHDL, on doit d'abord accéder au noeud d'entité de la spécification VHDL puis au noeud d'architecture. Ce dernier représente l'attribut numéro 3 du noeud d'entité (la figure 4.2). L'architecture est composée d'une liste de processus VHDL (deux processus dans cet exemple) représentée par l'attribut numéro 7 (la figure 4.2). Chacun des processus est composé d'une liste d'instructions séquentielles. Cette dernière est représentée par l'attribut numéro 5 dans le noeud de processus (la figure 4.3).

Pour accéder à l'information contenue dans la représentation VIF, on utilise une interface procédurale des outils de LEDA appelée LEDA Procedural Interface (LPI). Cette dernière est présentée dans la prochaine section.

4.2.2 Interface procédurale

L'interface procédurale est un ensemble de type de données et de fonctions implantés en langage C, nous permettant d'accéder à l'information contenue dans la représentation VIF. Un exemple d'application d'un programme en langage C utilisant l'interface procédurale est montré dans la figure 4.4. Dans cette exemple, on accède au noeud d'entité et à celui d'architecture de la représentation VIF de la figure 4.2. Ensuite, on accède à la liste des processus contenus dans le noeud d'architecture. Dans la prochaine section, nous allons décrire la structure de données utilisée pour organiser l'information contenue dans la représentation intermédiaire VIF.

4.2.3 Implantation de la structure de données

La figure 4.5 montre l'architecture générale de la structure de données utilisée pour organiser l'information contenue dans la représentation intermédiaire VIF. La structure finale est un DAG dont les noeuds sources (puits) représentent les entrées (sorties) primaires et pseudo-primaires. Les noeuds internes représentent les opérations VHDL et les arcs représentent les signaux et les variables de la spécification VHDL. Tous les signaux/variables sont convertis en bit ou vecteurs de bits. Les détails de toutes les étapes de construction du DAG sont présentés en l'annexe 2. Le DAG construit est utilisé pour calculer et propager les mesures de testabilité à travers les noeuds internes du DAG. Cette dernière est présentée dans la prochaine section.

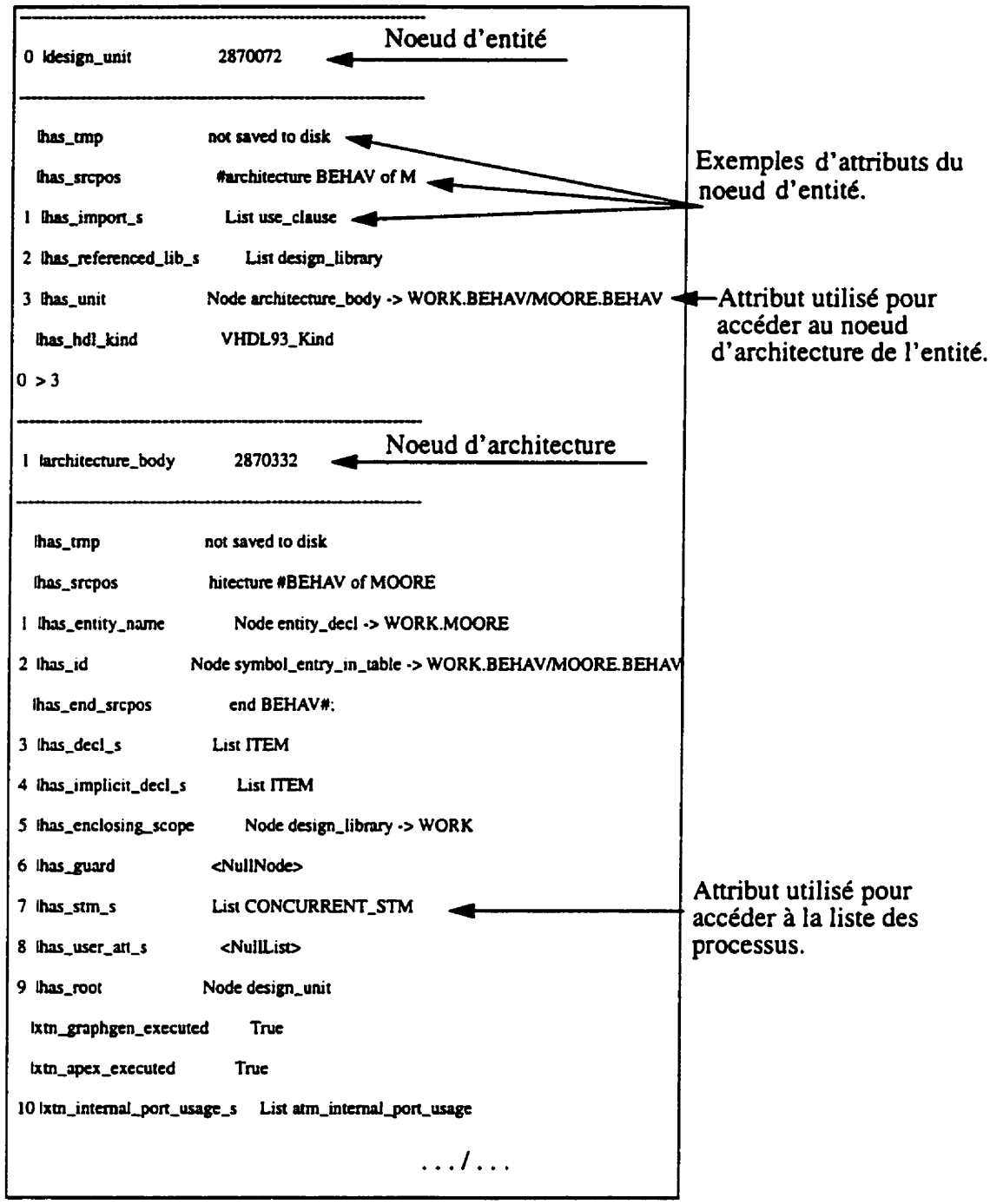

Figure 4.2 Exemple d'organisation de la représentation VIF.

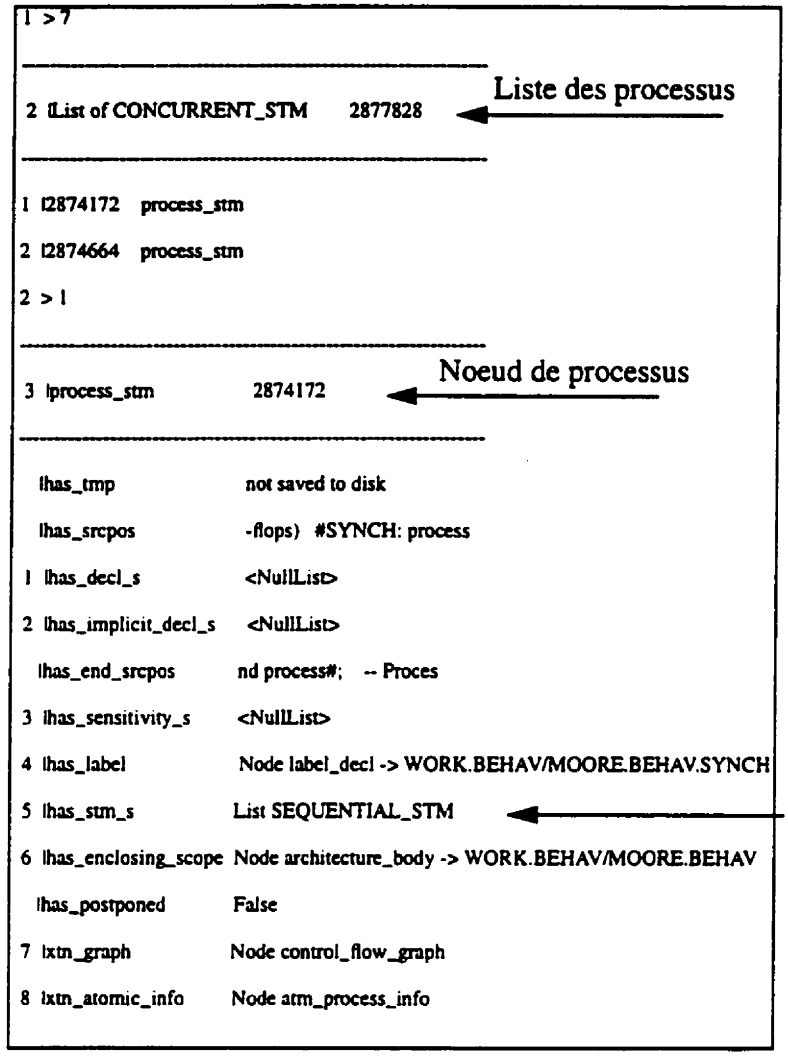

Figure 4.3 Exemple d'organisation de la représentation VIF (suite).

```
/*The intent of this program is simply to access to the list of concurrent statements of an architecture. */
/* It asks for a library name, opens it as the working library (WORK), then asks for a library unit */
/* name, reads it and gets the list of concurrent  statements and puts it in the list L  */

/*     MAIN PROCEDURE  */

void main(void) {

vifLibraryUnit LibUnitNode;          /* Pointer to the read unit */

vifStatus Status;                    /* Status returned by LPI procedures */

vifNode N,Lib;                       /* Nodes of the schema */

char LibName[80] ;

char UnitName[80] ;

vifList Concurrent_stm_List = NULL;

   /* Before macro LPI_START, a call to any LPI routine would be erroneous */

LPI_START

        /*Begin of a LPI Session : now, calls to LPI routines are allowed */

        /* Open the library LibName as the working library (WORK) */

                vifOpenLibraryWork(LibName, Status);

        /* Open the library unit UnitName of the working library in the read mode */

         vifOpenLibraryUnit("work",UnitName,VIF_READ, LibUnitNode,Status);

         /* Get the root node of the library unit (node of type "design_unit") */

                N = vifGetUnit(LibUnitNode);

        /* Get the root node of the architecture "architecture_body"*/

                N = vifGet_has_unit(N) ;

      /* Get the list of the concurrent statements of the root node "architecture_body"*/

                L = vifGet_has_stm_s(N) ;

LPI_END          /* End of the LPI session. An other session may start */

}
```

Figure 4.4 Un exemple d'application de l'interface procédurale.

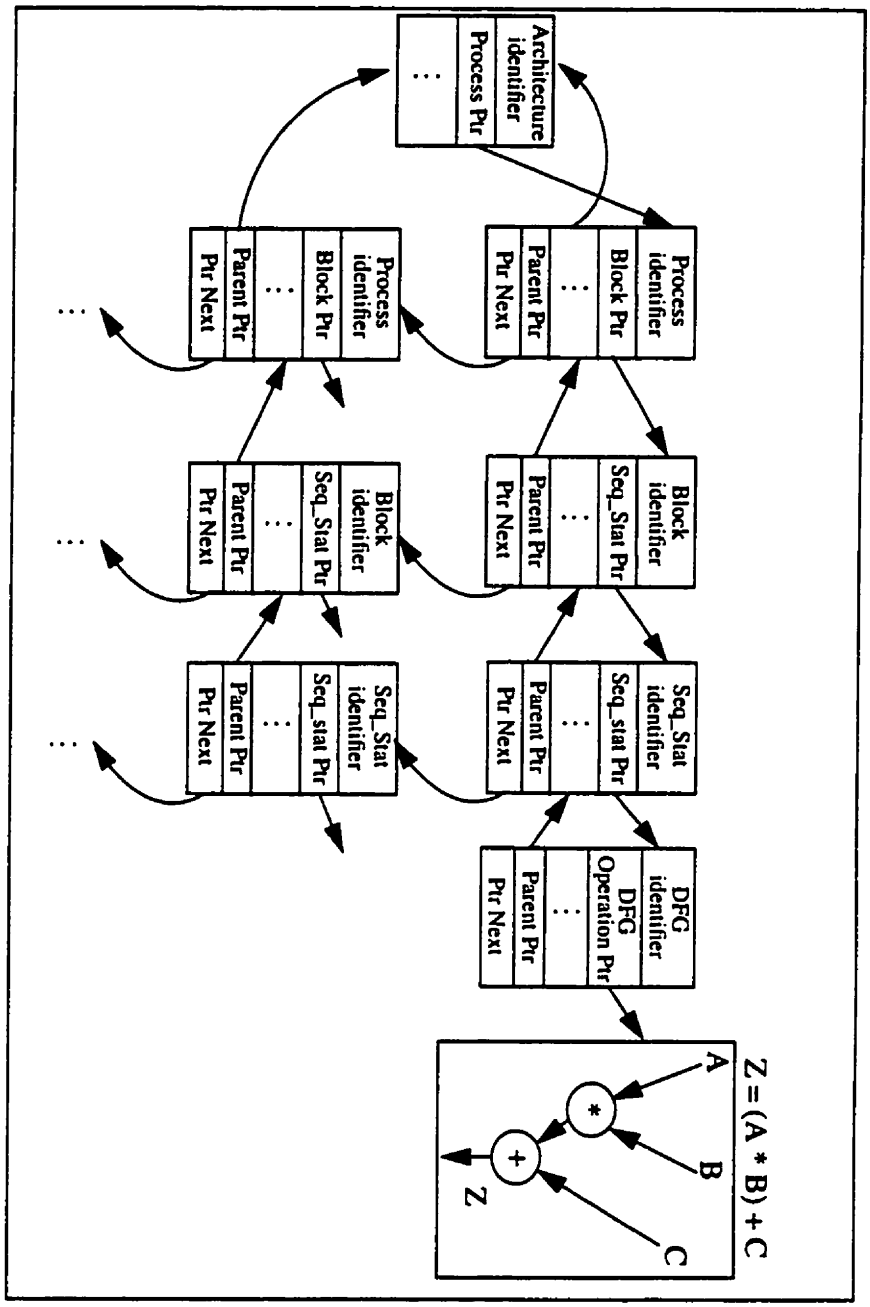

Figure 4.5 Structure de données utilisée pour organiser l'information contenue dans la représentation du VIF.

4.2.4 Calcul et propagation des mesures de testabilité

Après la construction du DAG, on procède au calcul de la contrôlabilité et de l'observabilité de chaque bit pour chaque signal/variable incluant quelques signaux internes des modules fonctionnels (implantation des opérations VHDL tels que l'addition, la comparaison, etc.). La détectabilité d'une panne collée à 0 (collée à 1) dans un bit donné est le produit de la contrôlabilité à 1 (contrôlabilité à 0) de ce bit et de son observabilité. À la fin de cette étape, on détermine les bits des signaux et variables où les pannes sont difficilement détectables. Une panne est difficilement détectable si sa détectabilité est inférieure à une certaine valeur appelée seuil de détectabilité. De la même manière, on défini le seuil de contrôlabilité et le seuil d'observabilité.

La figure 4.6 décrit la procédure de calcul et de propagation des mesures de testabilité. On commence d'abord à calculer la contrôlabilité de chaque bit en traversant le DAG à partir des noeuds sources jusqu'aux noeuds puits. Pour chaque type de noeud interne du DAG (opération VHDL), on a développé une formule qui calcule la contrôlabilité à la sortie du noeud en fonction de celles de ses entrées. Les observabilités sont ensuite calculées et propagées en traversant le DAG à partir des noeuds puits vers les noeuds sources. De même, pour chaque type de noeud du DAG on a développé une formule, calculant l'observabilité de chacune des entrées du noeud, en fonction de l'observabilité de la sortie et des contrôlabilités des autres entrées du même noeud. Toutes les formules de calcul de contrôlabilité et d'observabilité sont présentées en l'annexe 1. À la fin de la procédure de calcul et de propagation des mesures de testabilité, on détermine les bits des signaux et variables où les pannes sont difficilement détectables. D'où la nécessité d'améliorer la contrôlabilité et/ou l'observabilité de ces bits en insérant des

points de test. Dans la prochaine section, on décrit l'algorithme de sélection de points de test.

Entrée: Le DAG, seuil de détectabilité.

Sortie: Liste L des bits des signaux et variables où les pannes sont difficilement détectables.

Étape 1: Initialisation

. Assigner aux noeuds sources du DAG une contrôlabilité de valeur de 0.5 (test aléatoire).

. Assigner aux noeuds puits du DAG une observabilité de valeur de 1.0

. Noeuds sources (puits): entrée (sorties) primaires et pseudo-primaires.

. Noeuds internes: opérations VHDL; arcs: signaux et variables.

Étape 2: Propager les contrôlabilités à partir des noeuds sources vers les noeuds puits à travers les noeuds internes du DAG.

Étape 3: Propager les observabilités à partir des noeuds puits vers les noeuds sources à travers les noeuds internes du DAG.

Étape 4: Calculer la détectabilité de chaque type de panne (collé à 0 et collé à 1) dans chaque bit pour chaque signal et variable.

Étape 5: Mettre dans une liste L tous les bits des signaux et variables où les pannes sont difficilement détectables (c'est à dire, la détectabilité est inférieure au seuil de détectabilité).

Fin.

Figure 4.6 Implantation de la precédure de calcul et de propagation des mesures de testabilité.

4.2.5 Algorithme de sélection de points de test

L'algorithme de sélection de points de test est inspiré de l'algorithme proposé dans [146, 50] et adapté au niveau RTL. La figure 4.7 présente l'implantation de l'algorithme de sélection de points de test. L'entrée de l'algorithme est une liste des bits où la détectabilité de la panne est inférieure au seuil de détectabilité. Ces bits nécessitent une

amélioration de leurs contrôlabilité et/ou de leurs observabilité en introduisant des points de contrôle et/ou d'observation. On introduit d'abord les points de contrôle pour améliorer la contrôlabilité de tous les bits ayant une contrôlabilité inférieure au seuil de contrôlabilité, on recalcule de nouveau les contrôlabilités et les observabilités, puis on introduit les points d'observation pour améliorer l'observabilité des bits ayant une observabilité inférieure au seuil d'observabilité. Les points de contrôle (observation) dont leurs contrôlabilité (observabilité) est inférieure au seuil de contrôlabilité (d'observabilité) sont sélectionnés selon leurs proximités aux entrées (sorties) primaires ou pseudo-primaires. En effet, la région d'influence d'un point d'observation est montrée dans la figure 4.8(a). Un tel point n'affecte que les observabilités des signaux appartenant à son cône d'entrée. Contrairement à un point d'observation, un point de contrôle affecte la contrôlabilité et l'observabilité d'une certaine région du circuit (la figure 4.8(b)). La région R1 correspond à l'ensemble des signaux dont la contrôlabilité change suite à l'insertion d'un point de contrôle en S, alors que la région R2 correspond aux signaux dont l'observabilité a été affectée par le changement de contrôlabilité au signal S.

Il est important de constater qu'un signal difficile à contrôler et/ou à observer et appartenant à la région d'influence d'une autre signal S, peut trouver sa contrôlabilité et/ou son observabilité améliorée suite à l'insertion d'un point de contrôle ou d'observation en S. C'est la raison pour laquelle, on choisit parmi les signaux difficilement contrôlables (observables), ceux plus proches aux entrées (sorties) primaires ou pseudo-primaires.

Entrée: Liste L des bits des signaux et variables où les pannes sont difficilement détectables, seuil de contrôlabilité, seuil d'observabilité, seuil de détectabilité.

Sortie: Un ensemble de points de contrôle et d'observation à insérer dans la spécification VHDL.

Étape 1: Prendre un élément E_i de L dont la contrôlabilité est intérieure au seuil de contrôlabilité, le plus proche aux entrées primaires ou pseudo-primaires et insérer un point de contrôle. Enlever E_i de la liste L.

Étape 2: Propager l'effet du point de contrôle en recalculant la contrôlabilité et l'observabilité de de tous les bits. Répéter l'étape 1 jusqu'à ce que tous les bits des signaux et variables aient une contrôlabilité supérieure au seuil de contrôlabilité.

Étape 3: Mettre dans L tous les bits où la détectabilité est inférieure au seuil de détectabilité. Si L est vide alors fin du programme, sinon aller à l'étape 4.

Étape 4: Prendre un élément E_i de L dont l'observabilité est inférieure au seuil d'observabilité, le plus proche aux sorties primaires ou pseudo-primaires et insérer un point d'observation. Enlever E_i de la liste L.

Étape 5: Propager l'effet du point d'observation en recalculant l'observabilité de tous les bits. Répéter l'étape 4 jusqu'à ce que tous les bits des signaux et variables aient une observabilité supérieure au seuil d'observabilité.

Fin.

Figure 4.7 Implantation de l'algorithme de sélection de points de test.

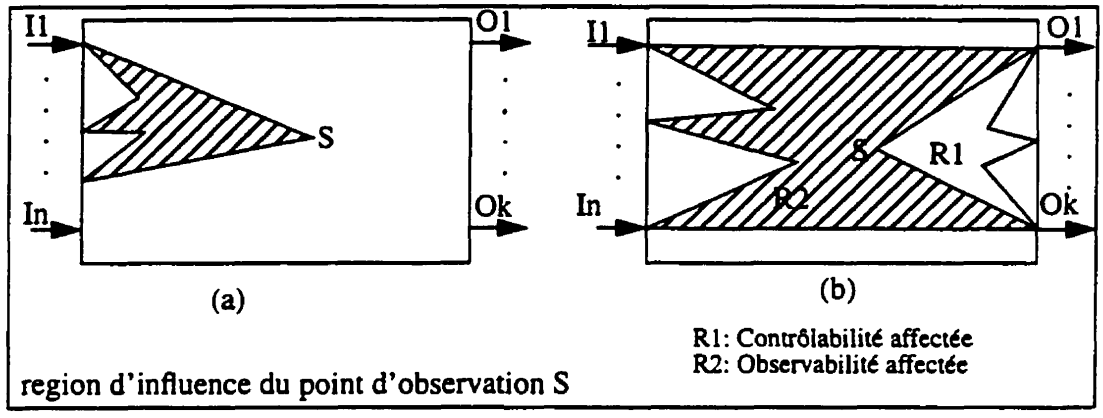

Figure 4.8 Régions d'influence d'un point de test. (a) Point d'observation. (b) Point de contrôle.

4.2.6 Insertion de points de test au niveau VHDL

Les points de test (points de contrôle et d'observation) sélectionnés par l'algorithme de sélection de points de test, sont insérés au niveau VHDL. Chaque point de test est caractérisé par un ensemble de paramètres qui sont utilisés pour l'insertion au niveau du code VHDL tels que: une étiquette indiquant le chemin hiérarchique de la position du point de test au niveau du code VHDL; le type du point de test (point de contrôle ou d'observation), la position du bit à modifier s'il s'agit d'un signal/variable déclaré comme un vecteur de bits, etc. Chaque point de contrôle (observation) est défini par une fonction (procédure) VHDL. Un package incluant les définitions des fonctions et des procédures utilisé pour l'insertion de points de test est défini et présenté en l'annexe 3.

4.2.7 Complexité de l'algorithme

La complexité de l'algorithme de l'analyse de testabilité et d'insertion de points de test est déterminée essentiellement par la complexité du DAG construit. La propagation des contrôlabilités (observabilités) nécessite le parcours du DAG en partant des noeuds

sources (puits) vers les noeuds puits (sources). La complexité de parcours d'un DAG est de O(|V| + |E|), dont V et E sont les ensembles de noeuds et d'arcs, respectivement. L'algorithme de sélection de points de test nécessite un nombre fini, n, d'itération pour déterminer les points de test à insérer au niveau VHDL où chaque itération est un parcours du DAG. En pratique, selon les trois grands circuits-test présentés dans le chapitre 3, section 3.2.7, la valeur de n est inférieure à 30 et le nombre de noeuds est inférieur à 5000. D'où une complexité linéaire de O(n(V + E)).

Le nombre de noeuds internes du DAG représente le nombre d'opérations VHDL dans le spécifications VHDL (addition, soustraction, comparaison, etc.). Le nombre de noeuds du DAG construit pour un circuit décrit au niveau RTL est négligeable par rapport à celui du DAG construit d'un circuit décrit au niveau portes logiques. Dans ce dernier cas, un noeud du graphe représente une porte logique. En effet, une description au niveau RTL permet d'augmenter la complexité du circuit qui est possible à traiter avec des ressources informatiques données, en faisant abstraction de l'implantation des opérations VHDL au niveau portes logiques. Dans notre méthode, les modules fonctionnels (correspondant à l'implantation d'une opération VHDL) sont décomposés en sous-modules fonctionnels élémentaires dans le but d'y accéder à certains signaux internes difficiles à contrôler et/ou à observer. Cette considération des signaux internes des modules fonctionnels, permet d'augmenter la complexité de l'algorithme au prix d'une bonne qualité du test aléatoire. Dans les prochaines sections, nous présentons un exemple d'application de l'algorithme d'analyse de testabilité et d'insertion de points de test et quelques résultats expérimentaux.

4.2.8 Exemple

Dans cet exemple, nous allons illustrer l'application des étapes de l'algorithme. D'autres exemples ont été présenté dans le chapitre 3, section 3.2.6. La figure 4.9(a) montre un exemple simple d'une spécification VHDL ainsi que le DAG correspondant (la figure 4.9(b)). Nous voulons augmenter la contrôlabilité à 1 du bit 1 de la variable V. Cette dernière est montrée en pointillé dans la figure 4.9(b). Un point de contrôle est donc nécessaire pour améliorer la contrôlabilité à 1 du bit 1 de la variable V. Le point de contrôle est caractérisé par un chemin hiérarchique indiquant sa position au niveau VHDL telle que indiquée dans la figure 4.9(b). Ce chemin hiérarchique consiste en une étiquette indiquant le nom de l'entité/l'architecture, le nom du processus ainsi que l'instruction séquentielle où la variable V est utilisée. La figure 4.9(c) montre la spécification VHDL modifiée après l'insertion de point de contrôle au niveau du bit 1 de la variable V. Le point de contrôle consiste en une fonction VHDL appelée "Insert_Control_Point(...)" permettant d'augmenter la contrôlabilité à 1 du bit 1 de V. Le point de contrôle consiste en une opération logique OU combinant comme entrées, le bit 1 de V et un signal supplémentaire TEST_IN. Un autre signal supplémentaire Test_Mode est utilisé pour reconfiguré le circuit en mode normal ou en mode test. Toutes les fonctions utilisées pour l'insertion de points de contrôle sont définies dans un package nommé "Test_Points" (présenté en l'annexe 3). Les points d'observation sont décrits par des procédures VHDL, aussi définies dans le package "Test_Points".

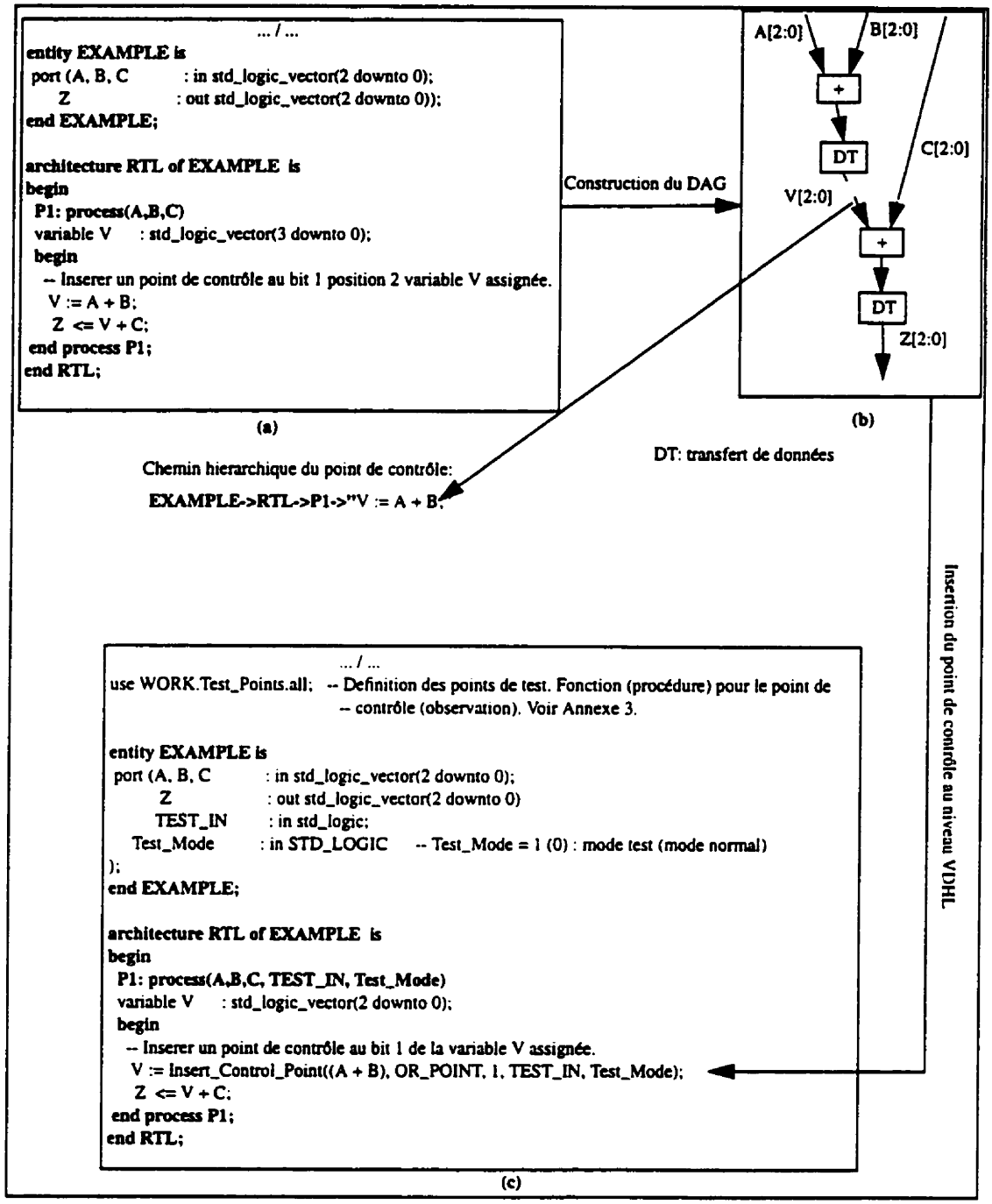

Figure 4.9 Exemple d'insertion de point de contrôle au niveau VHDL. (a) spécification VHDL. (b) Construction du DAG. (c) Spécification VHDL modifiée incluant un point de contrôle.

4.3 Résultats expérimentaux

Dans cette section, on présente quelques résultats expérimentaux que nous avons obtenus sur quelques exemples de circuits-tests. Nous allons d'abord évaluer l'efficacité de notre méthode en vérifiant la corrélation entre l'analyse de testabilité et l'insertion de points de test au niveau RTL en utilisant la génération de vecteurs de test au niveau portes logiques. Ensuite, nous montrerons l'efficacité de considérer les signaux internes des modules fonctionnels pour l'analyse de testabilité et l'insertion de points de test.

Chaque spécification VHDL du circuit-test est synthétisé au niveau portes logiques avant et après la l'insertion de points de test. Un ATPG aléatoire est utilisé pour évaluer la couverture de pannes avant et après l'insertion de points de test. Les couvertures de pannes résultantes sont ensuite comparées. Les circuits-tests (voir la table 4.1) [137] ont été synthétisés en utilisant les outils de Synopsys. Nous utilisons aussi l'ATPG de Synopsys pour évaluer la couverture de pannes dans le contexte du balayage complet et la génération aléatoire. Aucun circuits-test considéré (voir la table 4.1) n'est résistant au test aléatoire, c'est à dire, une séquence de vecteurs de test aléatoires est suffisante pour obtenir une couverture de pannes maximale. Notons que chaque unité de surface correspond à une porte NAND à deux entrées.

Pour chaque circuit, on fixe les valeur du seuil de détectabilité, de contrôlabilité et d'observabilité. Ensuite, on détermine le nombre de points de test à insérer au niveau VHDL. La table 4.2 présente les résultats de l'évaluation de la couverture de pannes en utilisant deux séquences de 5,000 et 10,000 vecteurs aléatoires. Pour chaque circuit, on note la couverture de pannes résultante et le nombre de points de test (nombre de points de contrôle vs. le nombre de points d'observation) requis pour atteindre la couverture de pannes correspondante.

Selon les résultats de la table 4.2, on peut remarquer que la couverture de pannes augmente considérablement avec l'insertion d'un petit nombre de points de test. Ceci permet d'illustrer une bonne estimation de notre méthode d'analyse de testabilité au niveau RTL malgré que la couverture de pannes est évaluée au niveau portes logiques. Aussi, les résultats montrent que la couverture augmente en augmentant la longueur de la séquence de vecteurs de test ce qui permet de justifier en effet, l'amélioration de la testabilité aléatoire du circuit résultant au niveau portes logiques. Notons que pour ces circuits de moyenne complexité, la surface et le délai du chemin critique sont quasiment les mêmes avants et après l'insertion des points de test. D'où l'avantage de considérer l'insertion de points de test avant la synthèse au niveau RTL. En effet, les performances du circuit synthétisés peuvent ne pas changer avec ou sans l'insertion de points de test.

Les circuits considérés et montrés dans la table 4.1 ne sont pas résistants au test aléatoires et ne requièrent pas une insertion de points de test au niveau des signaux internes des modules fonctionnels. Trois grand circuits-tests [94] résistants au test aléatoires ont été utilisés dont les résultats expérimentaux se trouvent dans le chapitre 3, section 3.2.7.

Nous allons montrer dans ce qui suit, l'efficacité de l'analyse de testabilité et l'insertion de points de test au niveau des signaux internes des modules fonctionnels. Ces derniers représentent l'implantation des opérations VHDL (addition, soustraction, comparaison, etc.). Chaque module fonctionnel est décomposé en sous-modules fonctionnels élémentaires connectés par des signaux internes. Ces derniers sont considérés pour l'analyse de testabilité et l'insertion de points de test. La décomposition des différents types de modules fonctionnels (additionneurs, comparateur, etc.) a été discutée dans le chapitre 3, section 3.2.6.2. La figure 4.10(a) montre la spécification VHDL d'un compteur de 16 bits. Le DAG correspondant est montré dans la figure 4.10(b). On a inclus seule-

ment la propagation des contrôlabilités dont les valeurs sont montrées entre parenthèses. Le seuil de contrôlabilité est fixé à 0.01. Nous remarquons qu'une faible contrôlabilité est obtenue à la sortie du comparateur d'égalité qui est de 0.37×10^{-10}.

D'abord, nous n'allons pas considérer l'analyse de testabilité et l'insertion de points de test des signaux internes des modules fonctionnels (qui sont dans cet exemple le comparateur d'égalité, le multiplexeur et l'additionneur). Un point de contrôle (augmentation de la contrôlabilité à 1 par l'opération OU) est donc nécessaire à la sortie du comparateur d'égalité. La spécification VHDL incluant le point de contrôle est montrée dans la figure 4.11. Toutes les fonctions d'insertion de points de contrôle sont définies dans le package "Test_Points". Ce dernier est inclus dans l'annexe 3.

Ensuite, Nous considérons l'analyse de testabilité et l'insertion de points de test au niveau des signaux internes du comparateur. Les signaux internes considérés pour cette analyse ont été discutés dans le chapitre 3, section 3.2.6.2. Deux points de contrôle (augmentation de la contrôlabilité à 1) sont nécessaires au niveau de deux signaux internes dans notre exemple. La figure 4.12 montre la spécification VHDL modifiée incluant les points de contrôle. Une fonction VHDL appelée "Insert_Control_Equal(...)" est utilisée pour décomposer le comparateur en sous-modules afin d'insérer des points de contrôle au niveau des signaux internes. La fonction "Insert_Control_Equal(...)" ainsi que ses paramètres, sont définies dans le package "Test_Points". De même, on définit une procédure permettant d'insérer des points d'observation au niveau des signaux internes du comparateur d'égalité (pas utilisé dans cet exemple).

La table 4.3 présente les résultats expérimentaux concernant l'insertion de points de contrôle dans le compteur de 16 bits. Les trois spécifications VHDL des figures 4.10(a), figure 4.11 et figure 4.12 sont synthétisés par les outils de Synopsys au niveau de portes

logiques. Un générateur de vecteurs de test aléatoires est ensuite utilisé pour évaluer la couverture de pannes résultante. Chaque spécification VHDL est synthétisée en considérant deux contraintes différentes d'optimisation. Dans chaque cas, on applique une séquence pseudo-aléatoire de 32,000 vecteurs. Les paramètres suivants sont notés dans la table 4.3: couverture de pannes résultante, la surface résultante, le délai du chemin critique et le nombre de pannes totales vs. le nombre de pannes redondantes. Notons que ce circuit résiste au test aléatoire.

Selon les résultats de la table 4.3, l'insertion de points de contrôle au niveau des signaux internes du comparateur permet d'avoir une couverture de panne élevée au coût d'une augmentation modeste de la surface et du délai. Aussi, la considération des signaux internes pour l'analyse de testabilité permet de minimiser le nombre de pannes redondantes. .

Tableau 4.1 Description des circuits-test

Circuit	Nb. d'entrées	Nb. de sorties	Flip-Flops	Surface	Déscription
C1	18	8	8	425	Timer/counter: 8 bit registers, a mux, counter and comparator
C2	9	8	4	210	Small state machine of 9 states
C3	8	8	5	246	Large state machine of 17 states
C4	17	17	17	132	Memory mapper

Tableau 4.2 Résultats expérimentaux

Circuit	5,000 vecteurs aléatoires		10,000 vecteurs aléatoires		nb. de points de contrôle/nb. de points d'observation)
	Avant insertion	Après insertion	Avant insertion	Après insertion	
	FC[%]	FC[%]	FC[%]	FC[%]	
C1	99.81	100	100	100	2/0
C2	95.10	97.87	98.69	99.63	2/1
C3	95.88	98.27	97.25	99.23	2/0
C4	81.47	87.43	91.38	96.32	4/0

```
                                    . . .
use WORK.Test_Points.all;

entity COMP is
 port (IN_COUNT                : in STD_LOGIC_VECTOR(15 downto 0);
    CLOCK                      : in Bit;
    TEST_IN                    : in BOOLEAN;
    Test_Mode                  : in STD_LOGIC;
    OUT_COUNT                  : out std_logic_vector(15 downto 0)
    );
end COMP;

architecture RTL of COMP is
begin
 process
 begin
  wait until CLOCK = '1';
-- Insertion d'un point de contrôle à la sortie du comparateur d'égalité
  if (Insert_Control_Points((IN_COUNT = "1111111111111111"), OR_POINT, TEST_IN, Test_Mode)) then
   OUT_COUNT <= "0000000000000000";
  else
   OUT_COUNT <= IN_COUNT + "0000000000000001";
  end if;
 end Process;
end RTL;
```

Figure 4.11 Spécification VHDL modifiée incluant un point de contrôle sans la considération des signaux internes des modules fonctionnels.

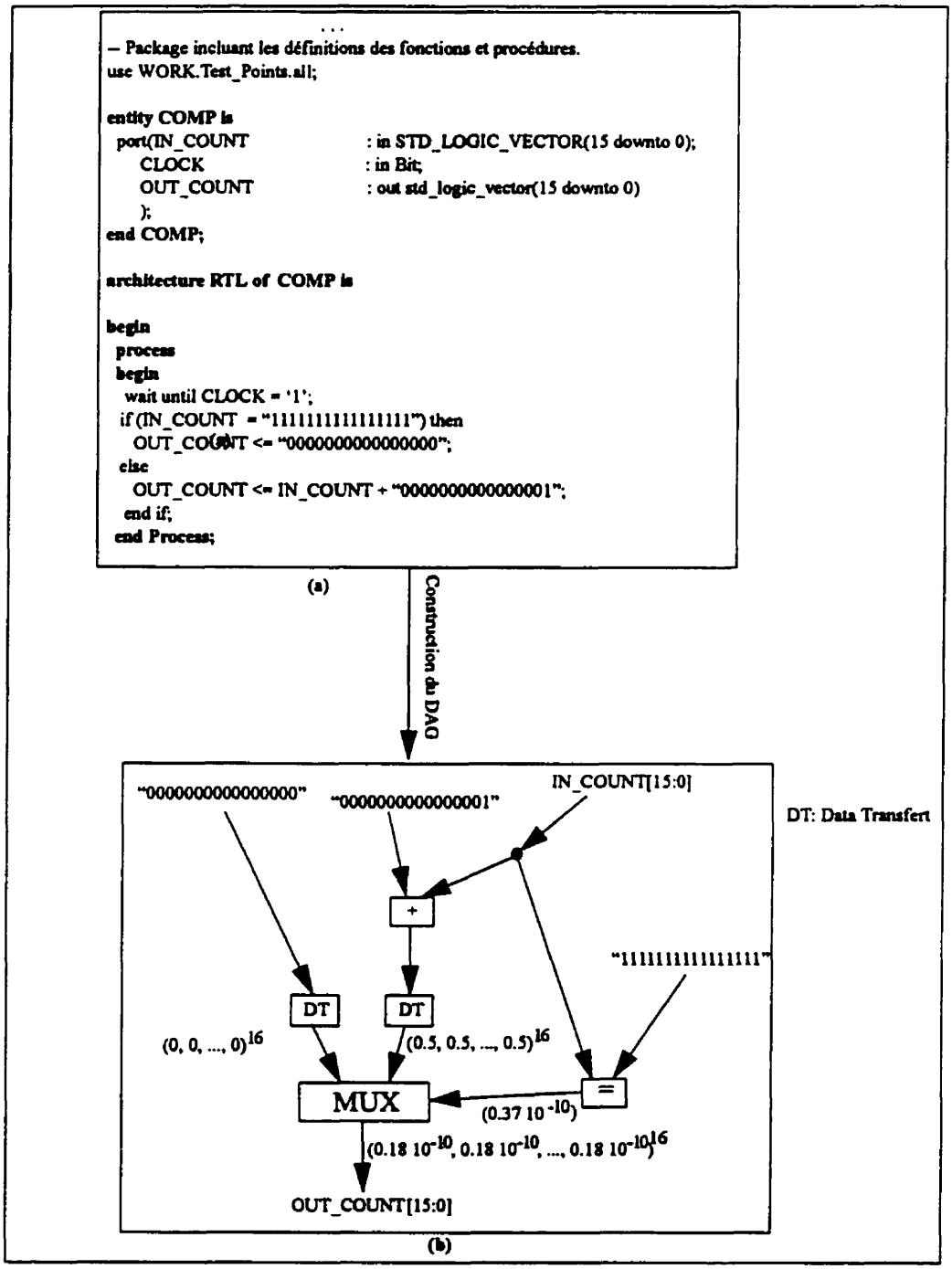

Figure 4.10 (a) spécification VHDL du compteur à 16 bits.
(b) Calcul et propagation des contrôlabilités.

```
...
use WORK.Test_Points.all;

entity COMP is
 port(IN_COUNT                  : in STD_LOGIC_VECTOR(15 downto 0);
     CLOCK                     : in Bit;
     TEST_IN_1, TEST_IN_2      : in BOOLEAN;
     Test_Mode                 : in STD_LOGIC;
     OUT_COUNT                 : out std_logic_vector(15 downto 0)
     );
end COMP;

architecture RTL of COMP is
begin
 process
 begin
  wait until CLOCK = '1';
-- Insertion de deux points de contrôle au niveau deux signaux internes du comparateur d'égalité
  if (Insert_Control_Equal( IN_COUNT, "1111111111111111", 2, (12, 6),(OR_POINT, OR_POINT),
                           (TEST_IN_1,TEST_IN_2), Test_Mode)) then
    OUT_COUNT <= "0000000000000000";
  else
    OUT_COUNT <= IN_COUNT + "0000000000000001";
  end if;
 end Process;
end RTL;
```

Figure 4.12 Spécification VHDL modifiée incluant deux point de contrôle avec la considération des signaux internes des modules fonctionnels.

Tableau 4.3 Résultas expérimentaux du compteur 16 bits

Type de contraintes d'optimisation	Couverture de pannes[%]			Surface			Délai			Nb. de pannes /Nb de pannes redondantes		
	Sans insertion	Après insertion		Sans insertion	Après insertion		Sans insertion	Après insertion		Sans insertion	Après insertion	
		Sans les SI	Avec les SI		Sans les SI	Avec les SI		Sans les SI	Avec les SI		Sans les SI	Avec les SI
1	91.25	96.04	98.96	222	223	226	8.16	8.91	8.94	541/41	628/ 26	606/0
2	87.69	96.04	99.01	240	224	270	9.44	8.76	10.47	634/48	632/ 26	1200/0

SI: signaux internes des modules fonctionnels.

4.3.1 Conclusion

On a présenté les étapes principales de l'algorithme d'analyse de testabilité et d'insertion de points de test. La complexité de l'algorithme est essentiellement déterminée par la complexité de construction du DAG ainsi que son utilisation pour calculer/propager les mesures de testabilité et la sélection des points de test. On a validé notre méthode par quelques circuits-test en montrant la corrélation entre l'analyse de testabilité et l'insertion de points de test au niveau RTL et la couverture de pannes au niveau portes logiques. D'autres résultats expérimentaux ont été présentés dans le chapitre 3, section 3.2.7. Enfin, on a justifié par l'exemple du compteur de 16 bits, la nécessité de considérer l'analyse de testabilité et l'insertion de points de test au niveau des signaux internes des modules fonctionnels. En effet, l'insertion de points de test à ce niveau permet d'augmenter la couverture de pannes au coût d'une augmentation modeste de la surface et du délai du circuit résultant au niveau portes logiques. Aussi, le nombre de pannes redondantes est minimisé ce qui permet d'avoir une bonne qualité de la génération de vecteurs de test.

CHAPITRE 5

Conclusion

Nous avons présenté dans cette thèse une nouvelle méthode d'analyse de testabilité et d'insertion de points de test. On a considéré les circuits décrits au niveau RTL et spécifiés en langage VHDL en supposant la méthode de balayage complet et un environnement de test aléatoire et de test intégré (BIST). La même méthode peut être appliquée pour le cas des circuits spécifiés en langage Verilog.

Nous avons développé des mesures de testabilité au niveau RTL afin d'identifier les bits des signaux et variables qui sont difficilement contrôlables et/ou observables. Un algorithme est ensuite utilisé pour sélectionner un ensemble de points de test. Ces derniers sont ensuite insérés dans la spécification VHDL. Au niveau VHDL, chaque point de test est défini par une fonction pour le point de contrôle ou une procédure pour le point d'observation. Un package a été défini incluant toutes les définitions des fonctions et procédures utilisées pour l'insertion de points de test. Ces points de test sont une partie intégrale de spécification VHDL et deviennent une partie de la spécification. De cette manière, l'outil de synthèse peut optimiser simultanément la logique fonctionnelle du circuit original et la logique des points de test insérés en tenant compte de toutes les contraintes de conception à savoir, la surface, la vitesse du circuit, la testabilité, etc. D'où l'avantage principal de notre méthode proposée.

Nous avons montré durant la validation de notre méthode, l'effet de l'insertion de points de test au niveau RTL sur la couverture de pannes du circuit au niveau portes logi-

ques. Nous avons montré par des résultats expérimentaux qu'une insertion d'un petit nombre de points de test permet d'augmenter la couverture de pannes d'une manière considérable. Ce qui permet en effet, d'illustrer une bonne estimation des mesures de testabilité que nous avons développées. Nous avons aussi démontré qu'une insertion de points de test au niveau RTL (avant la synthèse logique) n'implique pas nécessairement une augmentation de la surface et/ou du délai du chemin critique du circuit après la synthèse.

Nous avons considéré dans notre méthode l'analyse de testabilité et l'insertion de points de test des signaux internes des modules fonctionnels (implantation des opérations VHDL) afin d'augmenter la couverture des pannes. Cependant, cette augmentation de la couverture de pannes est obtenue au détriment d'une complexité additionnelle qui est due à la décomposition des modules fonctionnels en sous-modules fonctionnels élémentaires et à l'analyse de testabilité des signaux internes. Nous avons défini une fonction (procédure) VHDL permettant de décomposer les modules fonctionnels et insérer un ensemble de points de contrôle (d'observation) au niveau des signaux internes.

Nous avons considéré dans notre méthode l'analyse de testabilité et l'insertion de points de test chaque bit de chaque signal/variable indépendamment des autres bits du même signal/variable, ce qui n'est pas le cas pour les autres méthodes. En effet, cette insertion au niveau bit permet de réduire la surface additionnelle et rend la méthode plus proche de l'insertion structurelle. Aussi, notre méthode supporte tous les types de données des signaux/variables (entier, vecteur de bits, bit, etc.). L'accès aux bits individuels d'un signal/variable déclaré de type entier dépend de l'outil de synthèse utilisé pour la synthèse du circuit au niveau portes logiques. En effet, chaque outil de synthèse fournit un ensemble de fonctions pour convertir les types entier (vecteurs de bits) vers vecteurs de bits

(entier). Avec Synopsys, on utilise les packages standard de IEEE où les fonctions de conversion sont définies afin d'accéder au bits d'un signal/variable de type entier.

Enfin, nous avons développé un outil logiciel pour implanter notre méthode d'une complexité de 14,000 lignes en langage C. L'entrée de l'algorithme est une spécification VHDL originale et la sortie de l'algorithme est une spécification modifiée incluant les points de test qui est facilement testable au niveau portes logiques dans un environnement de test aléatoire.

On doit admettre que la méthode que nous avons développée dans cette thèse représente à notre avis une contribution importante dans le domaine de la testabilité des circuits intégrés numériques. Cependant, plusieurs améliorations et extensions sont envisageables à notre méthode. Une étape importante de validation de notre méthode est de la comparer à une méthode d'analyse de testabilité et d'insertion de points de test au niveau portes logiques. En effet, il serait intéressant de faire une correspondance des bits des signaux/variables du niveau RTL (avant la synthèse logique) au niveau portes logiques (après la synthèse logique). De cette manière, on peut vérifier si un bit donné d'un signal/variable au niveau RTL est difficile à contrôler et/ou à observer, l'est aussi au niveau portes logiques. Un problème majeure à tenir compte est la propagation des constantes durant le processus d'optimisation, certains points de test insérés au niveau RTL peuvent ne pas êtres générés au niveau portes logiques après la synthèse.

Nous avons utilisé dans notre méthode un algorithme vorace (greedy) pour la sélection de points de test. En effet, n'importe quel algorithme plus efficace [119] peut être utilisé pour réduire le nombre de points de test requis tout en augmentant la couverture de pannes.

Nous n'avons pas considéré dans notre méthode l'effet de partage de ressource sur la méthode d'analyse de testabilité et d'insertion de points de test. L'outil de synthèse de Synopsys utilise une technique de partage de ressources au niveau de chaque processus VHDL. Il serait intéressant d'explorer la relation entre le partage de ressources et l'analyse de testabilité au niveau RTL.

Enfin, nous voyons comme extension importante de ce projet de recherche est la vérification des règles de conception pour la testabilité (Design Rule Checking - DRC) au niveau RTL. Comme pour le cas de l'insertion de points de test au niveau RTL, on peut appliquer la technique du DRC au niveau RTL et ainsi modifier le circuit au niveau RTL afin de satisfaire toutes les règles de conception pour la testabilité.

Références

[1] ABRAMOVICI, M., BREUER, M. A. and FRIEDMAN, A. D., (1990), Digital Systems Testing and Testable Design, Revised Printing, IEEE PRESS, ISBN. 0-7803-1062-4.

[2] ABRAMOVICI, M., KULIKOWSKI, J., and ROY, R. K., (1991), The Best Flip-Flops to Scan, Proceedings of IEEE International Test Conference, 166-173.

[3] ABRAMOVICI M., and PARIKH, P. S., (1992), Warning: 100% Fault Coverage May Be Misleading, Proc. of International Test Conference.

[4] ABRAMOVICI, M., PARIKH, P. S., MATHEW, B., and SAAB, D. G., (1993), On Selecting Flip-Flops for Partial Reset, Proc. of IEEE International Test Conference, 1008-1012

[5] AGRAWAL, V.D., et al., (1989),An Economical Scan Design for Sequential Logic Test Generation, Proceedings of the 19th International Symposium on Fault Tolerant Computing, 118-125.

[6] AGARWAL, V. D.., and CERNY, E.,(1981), Store and generate Built-In Self-Testing, Proc. FTCS 11, 35-40.

[7] AGRAWAL, V.D., et. CHAKRABORTY, T.J., (1995), High-performance Circuit Testing with Slow Speed Testers, Proceedings of International Test Conference, 302-310.

[8] AGRAWAL, V.D., CHENG, K-T., JOHNSON, D. D and LIN, T. (1988), Designing Circuits with Partial Scan, IEEE Design & Test of Computers, 8-15.

[9] AGRAWAL, V. D., CHENG, K-T., JOHNSON, D. D., and LIN, T., (1987), A Complete Solution to the Partial Scan Problem, Proceedings of IEEE International Test Conference, 44-51.

[10] AGRAWAL, V., et MERCER, M.R., (1982) Testability Measures-What Do They Tell Us? Proc. International Test Conference, 391-396.

[11] AGRAWAL, V. D., SETH, S. C., and AGRAWAL, P., (1981), LSI Product Quality and Fault Coverage, ACM/IEEE Design Automation Conference, 196-203.

[12] AMAZONAS, J. R., and STRUM, M., (1989), Pseudoexhaustive Test Techniques: A New Algorithm to Partition Combinational Networks, Proceedings of the 1st European Test Conference, 392-397.

[13] ANDO, H., (1980), Testing VLSI with Random Access Scan, Proc. of COMPCON S'80, 50-52.

[14] CHEN, C. H. and MENON, P. R., An Approach to Functional Level Testability Analysis, (1989), International Test Conference, 685-693.

[15] BARDELP, H., McANNEY, W. H., and SAVIR, J., (1987), Built-In Test For VLSI Pseudorandom Techniques, a Wiley Inter-Science Publication, John Wiley & Sons ISBN. 0-471-62463-2.

[16] MAO, W., and GULATI, R. K., (1996),Improving Gate-Level Fault Coverage by RTL Fault Grading , Proc. of International Test Confrence, 463-472.

[17] Gu, X. et al., (1994), Testability Analysis and Improvement from VHDL Behavior Specification, Proc. EURO-DAC, 644-649.

[18] CHEN, C. H. and SAAB, D. G., (1992), Behavioral Synthesis for Testability, International Conference on CAD, 612-615.

[19] BEN BENNETTS, R. G., (1994), Progress in Design for Test: A personal View, IEEE Design and Test of Computers, 53-59.

[20] BENNETTS, R. G., MAUNDER, C. M., and ROBINSON, G. D., (1981), CAMELOT: A Computer-Aided Measure for Logic Testability, IEE Proc., Vol. 128, Part E, No. 5, 177-189.

[21] BOUBEZARI, S., and KAMINSKA, B., (1996), A New Reconfigurable Test Vector Generator for Built-In Self-Test Applications, Journal of Electronic Testing: Theory and Applications, Vol. 8, 153-164.

[22] BHATACHARYA, S. et al., (1993), Transformation and Resynthesis for Testability of RT Control Data Path Specification, IEEE Transactions on VLSI systems, 1(3):304-318.

[23] DEY, S., and POTKONJAK, M., (1994), Transforming Behavioral Specifications to Facilitate Synthesis of Testable Designs, International Test Conference, 184-193.

[24] BREWER, M. A., (1978), New Concepts in Automated Testing of Digital Circuits, Proc. EEC Sym. on CAD of Digital Electronic Circuits and Systems, Brussels, 69-92.

[25] BRIERS, A. J., and TOTTON, K.E., (1986), Random Pattern Testability by Fast Fault Simulation, Proceedings of the IEEE International Test Conference, 274-281.

[26] BRGLEZ, F., BRYAN, D., and KOZMINSKI, K., (1989). Combinational Profiles of Sequential Benchmark Circuits, International Symposium on Circuits and Systems .

[27] BRGLEZ, F., POWNAL, P., and HUM, R., (1985). Accelerated ATPG and Fault Grading via Testability Analysis," in Proc. of IEEE International Symposium on Circuits and Systems, 695-698, 1985.

[28] BRGLEZ, F., POWNAL, P., and HUM, R., P., (1984), Applications of Testability Analysis: from ATPG to Critical Delay Path Tracing, Proc. IEEE International Test Conference, 705-712.

[29] VISHAKA, P., et al, (1993), AMBIANT: Automatic Generation of Behavioral Modifications for Testability, IEEE ICCD, 63-66.

[30] CHEN, C. H. and SAAB, D. G., (1993), A Novel Behavioral Testability Measure, IEEE Transaction on CAD, Vol. 12, No. 12, 1960-1970.

[31] CHENG, T-T., and AGRAWAL, V., (1992), Initializability Consideration in Sequential Machine Synthesis, IEEE Transactions on Computers, Vol. 41, 374-379.

[32] CHENG, T-T., and AGRAWAL, V., (1990), A Partial Scan Method for Sequential Circuits with Feedback, IEEE Transactions, Vol. 39, No. 4, 544-548,

[33] CHENG, T-T., and AGRAWAL, V., (1989), An Economical Scan Design for Sequential Logic Test Generation, Proceedings of the IEEE Fault Tolerant Computing Symposium, 28-35.

[34] CHENG, T-T., and AGRAWAL, V., (1988), Designing Circuits with Partial Scan, IEEE Design and Test of Computers, 8-15.

[35] CHENG, T-T., and BREUER, A., (1985), Automatic Design for Testability via Testability Measures, IEEE Transactions on Computer-Aided Design, Vol. CAD-4, 3-11.

[36] CHENG, D. I., CHENG, K-T., WANG, D., and MAREK-SADONSKA, M., (1996) A New Hybrid Methodology for power Estimation, 33rd ACM/IEEE Design Automation Conference.

[37] CHENG, K.-T., and LIN, C.-J., (1995) Timing-Driven Test Insertion for Full-Scan and Partial-Scan BIST, Proceedings of Internationa Test Conference, 506-514.

[38] CHIKERMANE, V., and PATEL, J.-H., (1991), A Fault Oriented Partial Scan Design Approach, Proc. of International Conference on Computer-Aided Design, 332-335.

[39] CHIKERMANE, V., et PATEL, J.-H., (1990), An Optimization Based Approach to the Partial Scan Design problem, Proceedings of IEEE International Test Conference, 377-386.

[40] CHIN, C. K., and McCLUSKEY, E. J., (1985), Test Length for Pseudo-Random Testing, Proceedings of the IEEE International Test Conference, 94-99.

[41] CHEN, H., KARNIK, T., and SAABb, D. G., (1994), Structural and Behavioral Synthesis for Testability Techniques, IEEE Transaction on CAD Vol 13, No. 6, 777-785.

[42] DASGUPTA S., , GOEL P., WALTER R.T., and WILLIAMS T., (1982), A Variation of LSSD and its Implication on Design and Test Pattern Generation in VLSI, International Test Conference, 63-66.

[43] EICHELBERGER E. B., and WILLIAMS T. W., (1977), A Logic Design Structure for LSI Testability, Proc. of the 14th Design Automation Conference, 206-212.

[44] ERCOLANI S., FAVALLI M, DAMIANI M., OLIVO P., and RICCO B., (1992), Testability Measures in Pseudorandom Testing, IEEE Transactions on CAD, Vol. CAD-11, 794-800.

[45] FOX J. R., (1977), Test Point Condensation in the Diagnosis of Digital Circuits, Proceedings IEE, Vol. 124, No. 2.

[46] FUJIWARA H., and YAMAMOTO A., (1992), Parity-scan Design to Reduce the Cost of Test Application, Proceedings of International Test Conference, 283-292.

[47] FUNATSU S., WAKATSUKI N., and YAMAD A., (1978), Designing Digital Circuits with Easily Testable Considerations, Proc. of International Test Conference, 98-102.

[48] LEDA VHDL System, Version 4.0.3., LEDA S.A. 35 Avenue du Granier 38240 Meyland France.

[49] GAGE R., Structured CBIST in ASICs, (1993), Proceedings of International Test Conference, 332-338.

[50] PAPACHRISTOU, C. and CARLETTA, J., (1995), Test Synthesis in the Behavioral Domain, IEEE International Test Conference, 693-702.

[51] VISHAKANTAIAH, P. et al., (1992), Automatic Test Knowledge Extraction From VHDL (ATKET), IEEE Design Automation conference, 273-278.

[52] GOEL, P., (1980), Test Cost Analysis and Projections, Proceedings of 17th Design Automation Conference, 77-84.

[53] GOLDSTEIN, L. M., and THIGEN, E. L., (1980), SCOAP: Sandia Controllability/Observability Analysis Program, Proc. 17th Design Automation Conference, 190-196.

[54] GOLDSTEIN, L. H., (1979), Controllability/Observability Analysis of Digital Circuits, IEEE Transactions on Circuits and Systems, Vol. CAS-26, No. 9, 685-693.

[55] GRASSON, J., (1979), TMEAS, A Testability Measurement Program, 16th Design Automation Conference, 156-161.

[56] GUNDLACH, H. H., and MULLER-GLASSER, K-D., (1990), On Automatic Test Point Insertion in Sequential Circuits, Proceedings of the IEEE International Test Conference, 1072-1079.

[57] CHO, C. H., and ARMSTRONG, A., (1994), B-algorithm: A Behavior Test Generation Algorithm, IEEE International Test Conference, 968-979.

[58] HAYES, J. P., (1974), On Modifying Logic Networks to Improve their Diagnosability, IEEE Transactions on Computers, Vol. C-23, No. 1.

[59] HAYES, J. P., and FRIEDMAN, A. D., (1974), Test Point Placement to Simplify Fault Detection, IEEE Transactions on Computers, Vol. C-23, 727-735.

[60] HAYES, J.P., and McCLUSKEY, E. J., (1980), Testability Considerations in Microprocessor-Based Designs, Computer.

[61] HUISMAN, L. M., (1988), The Reliability of Approximate Testability Measures, IEEE Design and Test of Computers, 57-67.

[62] WAGNER, K. D., and DEY, S., (1996), High-Level Synthesis for Testability: A Survey and Perspective, In Proc. Design Automation Conference, 131-136.

[63] IYENGAR, V. S., and BRAND, D., (1989), Synthesis of Pseudo-Random Pattern Testable Design, Proceedings of the IEEE International Test Conference, 501-508.

[64] CHIKERMANE, V., LEE, J., and PATEL, J. H., (1994), Addressing Design for Testability at the Architectural Level, (1994), IEEE Transactions on Computer-Aided Design, Vol. 13, No. 7, 920-934.

[65] JONE, W.-B., and PAPACHRISTOU, C.A., (1988), On Partioning for Pseudo-exhaustive Testing of VLSI Circuits, Proceedings of the IEEE International Test Conference on Circuits and Systems, 1843-1846.

[66] KAMINSKA, B., SAVARIA, Y., YOUSSEF, M., and KOUDIL, M., (1991), Test Point Insertion for Pseudo-Random Testing, Proceedings of the IEEE International Test Conference on Circuits and Systems.

[67] KARSNIESKI, A., (1991), Can Redundancy Enhance Testability, Proceedings of IEEE International Test Conference, 483-491.

[68] KARSNIESKI, A., and PILARSKI, S., (1989), Circular Self-Test path: A Low-Cost BIST Technique for VLSI Circuits, IEEE Transactions on Computers-Aided Design, 46-55.

[69] STEENSMA, J., CATTOOR, F., and De MAN, H., (1991), Partial Scan at the Register-Transfer Level, IEEE International Test Conference, 488-497.

[70] THOMAS, T., VISHAKANTAIAH, P., and Abraham, J. A., (1994), Impact of Behavioral Modifications For Testability, IEEE VLSI Test Synposium, 427-432.

[71] KONEMANN, B., et al., (1979), Built-In Logic Block Observation Techniques, International Test Conference 37-41

[72] KONEMANN, B., MUCHA, J., and ZWIEHOFF, G., (1980), Built-In Test for Complex Digital Integrated Circuits, IEEE Journal of Solid State Circuits, 315-318.

[73] KRISHNAMURTY, B., (1987), A Dynamic Programming Approach for the Test Point Insertion Problem, Proc. of the 24th Design Automation Conference, 695-705.

[74] LEE, T. C, JHA, N. K., and WOLF, W. H., (1993), Behavioral synthesis of Highly Testable DataPaths Under Non-Scan and Partial Scan Environments, IEEE Design Automation Conference, 292-297.

[75] LIN, C-J., ZORIAN, Y., and BHAWMIK, S., (1993), PSBIST: A Partial Scan Based Built-In Self Test Scheme, Proc. of IEEE International Test Conference, 507-516

[76] POTKONJAK, M., DEY, S., and ROY, R., (1995), Behavioral Synthesis of Area Efficient Testable Designs Using Interaction Between Hardware Sharing and Partial Scan, IEEE Transactions on Computer-Aided Design, Vol. 14, No. 9, 1141-1154.

[77] LISANKE, R., BRGLEZ, F., DEGEUS, A.J., and GREGORY, D., (1987), Testability-Driven Random Test Pattern Generation, IEEE Transactions on Computer-Aided Design, Vol. CAD-6, 1082-1087.

[78] CHIU S. K., and PAPACHROSTOU, C., (1991), A Built-In Self-Testing Approach For Minimizing Hardware Overhead, IEEE International Conference on Computer Design.

[79] AVRA, L., (1991), Allocation and Assignment in High-Level Synthesis for Self-Testable Data Paths, IEEE International Test Conference.

[80] NOURANI, M., and PAPACHRISTOU, C., (1997), Structural BIST Insertion Using Behavioral Test analysis, European Design Automation Conference, 64-68.

[81] BHATATACHARYA, S., and DEY, S., (1996), A High Level Alternative to Full Scan Testing With Reduced Reduced Area and Test Application Overheads, IEEE VLSI Test Synposium.

[82] NORWOOD, R. B., and McCLUSKEY, E. J., (1996), Low Overhead Scan Data Paths, IEEE International Test Conference, 659-668.

[83] ABADIR, M. S., and BRUER, M. A., (1985), A Knowledge Based System for Designing Testable VLSI Chips, IEEE Design & Test of computers, Vol. 2, No. 4, 56-68.

[84] MAZUMDER, P., and PATEL, J.H., (1987), An Efficient Built-In Self Testing for Random Access Memory, Proceedings of the IEEE international Test Conference, 1072-1077.

[85] McCLUSKEY, E. J., (1984), Verification Testing - A pseudo-Exhaustive Test Technique, IEEE Transactions on Computers,.

[86] BHATTACHARYA, S., DEY, S., and SENGUPTA, B., (1997), An RTL Methodology to Enable Low Overhead Combinational Testing, Europeen Design Automation Conference, 146-152.

[87] BHATIA, S., and JHA, N. K., (1994), Genesis: A Behavioral Synthesis System for Hierarchical Testability, Europeen Design Automation Conference, 272-275.

[88] MERCER, M. R., and UNDERWOOD, B., (1984), Correlating Testability with Fault Detection Proc. International Test Conference, 697-704.

[89] GHOSH, I., RAGHUNATHAN, A., and JHA, N. K., (1997), Design for Hierarchical Testability of RTL Circuits Obtained by Behavioral Synthesis, Europeen Design Automation Conference, 173-179.

[90] MURADALI, F., AGARWAL, V., and NADEAU-DOSTIE, B., (1990), A New Procedure for Weighted Random Built-In Self-Test, Proc. of International Test Conference, 660-669.

[91] MURADALI, F., NISHIDA, T., and SHIMUZU, T., (1995), A Structure and Technique for Pseudorandom based Testing of Sequential Circuits, Journal of Electronic Testing: Theory & Applications, 107-115.

[92] MURADALI, F., and RAJSKI, J., (1996), A Self-Driven Test Structure for Pseudorandom Testing of Non-Scan Sequential Circuits, Proc. of 14th IEEE VLSI Test Symposium, 17-25.

[93] MUROGA, S., (1982), VLSI System Design: When and How to Design Very-Large-Scale-Integrated Circuits, Willey-Interscience, New York (N. Y.).

[94] LogicVision, Inc., Ottawa, Canada.

[95] SOUFI, M., (1997), Caractérisation et amélioration de la testabilité séquentielle pseudo-aléatoire des circuits VLSI, Thèse de Doctorat, département de génie électrique et de génie informatique, École Polytechnique de Montéal.

[96] BOUBEZARI, S., and KAMINSKA, B., (1995), A Deterministic BIST Based on Cellular Automata Structures, IEEE Transaction on Computers Vol. 44, No. 6, 805-816.

[97] BOUBEZARI, S., and KAMINSKA, B., (1993), Cellular Automata Synthesis Based on Precomputed Test Vectors For BIST, IEEE International conference on CAD, 578-583.

[98] OHLETZ, M., WILLIAMS, J., and MUCHA, J.P., (1991), Overhead in Scan and Self-Testing Design, Proc. of IEEE International Conference on Computer-Aided Design, 376-380.

[99] PAPOULIS, A., (1991), Probability Random Variables and Stochastic Processes, Third Edition McGraw Hill.

[100] PARKER, K.P., and McCLUSKEY, E. J. (1975), Probabilistic Treatment of General Combinational Circuits, IEEE Transactions on Computers, 668-670 .

[101] BREWER, J. E., (1994), A single-Chip Digital Signal Processing Subsystem, International Conference on Wafer Scale Integration, 265-272.

[102] Test Synthesis Seminar, *Digest of Papers, IEEE ITC*, 1994.

[103] LEE, T. C., WOLF, W. H., JHA, N. K., and ACKEN, J. M., (1993), Behavioral Synthesis for Easy Testability in Data Data Path Allocation, Proc. ICCD.

[104] RATIU, I., SANGIOVANNI-VINCENTELLI, M.A., and PETERSON, D.O., (1982) VICTOR: A Fast VLSI Testability Analysis Program, Proc. International Test Conference, 397-401.

[105] LEE D. H and REDDY S. M., (1990), On Determining Scan Flip-Flops in Partial-Scan Design, Proc. ICCAD, 322-325.

[106] LEE, T. C., WOLF, W. H., and JHA, N. K., (1993), Behavioral Synthesis for Easy Testability in Data Data Path Scheduling, Proc. ICCD, 616-619.

[107] GHOSH, I., JHA, H. K., and DEY, S., (1997), A Low Overhead Design for Testability adn Test Generation Technique for Core-based Systems, IEEE International Test Conference, 50-59.

[108] RUDNICK, E. M., CHIKERMANE, V., and PATEL, J. H., (1994),An Observability Enhancement Approach for Improved Testability and At-Speed Test, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 13, No. 8, 1051-1056.

[109] RUTHMAN, R. A., (1992), Fault Detection Test Generation for Sequential Logic Heuristic Tree Search, IEEE Computer Repository Paper, No. R-72-187.

[110] SAAB, D. S., SAAB, Y. G., and ABRAHAM, J. A., (1992), CRIS: A Test Cultivation Program for Sequential VLSI Circuits, Proc. of the Int. Conf. on Computer-Aided Design, 216-219 .

[111] SARMA, S., (1991), Built-In Self-Test Considerations in High-Performance, General-Purpose Processor, Proceedings of IEEE International Test Conference, 21-27.

[112] SAVARIA, Y., (1988), Conception et Vérification des Circuits VLSI, Chapître 6, Editions de l'École Polytechnique de Montréal, ISBN 2-553-00207-, 276.

[113] SAVARIA, Y., and KAMINSKA, B., (1988), Force-Observe, A New Design for Testability Approach, Proceedings of the International Symposium on Circuits and Systems, 193-197.

[114] SAVARIA, Y., LAGUE B., and KAMINSKA, B., (1989), A Pragmatic Approach to the Design of Self-Testing Circuits, Proceedings of the IEEE International Test Conference, 745-754.

[115] SAVARIA, Y., YOUSSEF, M., KAMINSKA, B., and KOUDIL, M., (1991), Automatic Test Point Insertion for Pseudo-random Testing, Proc. of European Test Conference, 253-262.

[116] SAVIR, J., (1983), Good Controllability and Observability Do Not Guarantee Good Testability, IEEE Transactions on Computers, Vol. C-32, No 12, 1198-2000.

[117] SAVIR, J., DITLOW, G., and BARDELL, P., (1983), Random Pattern Testability, Proc. Fault Tolerant Computing Symposium, 80-89.

[118] SCHOTTEN, C., and MEYR, H., (1995), Test Point Insertion for Area Efficient BIST, Proceedings of International Test Conference, 515-523.

[119] SEISS, B. H., TROUBOST, P. M., and SCHULZ, M. H. (1991), Test Point Insertion for Scan-Based BIST, Proceedings of European Test Conference, 253-262.

[120] SETH, S., PAN, L., and AGRAWAL, V., (1985), PREDICT: Probabilistic Estimation of Digital Circuit Testability, Proc. of International Fault Tolerant Computing Symposium, 220-225.

[121] SHEDLETSKY, J. J., and McCLUSKEY, E. J., (1976), The Error Latency of a Fault in Sequential Digital Circuits, IEEE Transactions on Computers, Vol. C-25, No. 6, 655-659.

[122] SHEDLETSKY, J. J., and McCLUSKEY, E. J., (1975), The Error Latency of a Fault in Combinational Digital Circuits, 5th Annual Fault-Tolerant Computing Symposium, 210-214, Paris, France.

[123] PAPACHRISTO, C. A., CHIU, S., and HARMANANI, H., (1991), A Data Path Synthesis Method for Self-Testable Designs, Proc. Design Automation Conference, 378-384.

[124] PENG, Z., and KUCHCINSKI, K., (1994), Automated Transformation of Algorithms into Register-Transfer level Implementations, IEEE Transactions on CAD, 150-165.

[125] DEY, S., GANGARAM, V., and POTKONJAK, M., (1995), A Controller-Based Design-for-Testability Technique for Controller-Data Path Circuits, Proc. ICCAD, 534-540.

[126] AITKEN, R. C., (1995), An Overview of Test Synthesis Tools, IEEE Design & Test, 8-15.

[127] GU, X., (1992), Testability Analysis and Improvement in High-Level Synthesis Systems, Thèse de Doctorat, Linkoping, Sweden.

[128] SARFERT, T. M., Markgraf, R., Trischler, E., Schulz, M. H., (1989), Hierarchical Test Pattern Generation Based on High-Level Primitives, Proc. International Test Conference, 470-470.

[129] DE MICHELI, G., Synthesis and Optimization of Digital Circuits, (1994), McGraw-Hill, ISBN 0-07-016333-2

[130] SYNOPSYS Inc. (1996), Design Compiler.

[131] STEPHENSON, J. E. and GRASON, J., (1976), A Testability Measure for Register Transfer Level Digital Circuits, Proc. International Symposium on Fault-Tolerant Computing, 101-107.

[132] TAMARAPALLI, N and RAJSKI, J., (1996), Constructive Multi-Phase Test Point Insertion for Scan-Based BIST, Proc. of IEEE International Test Conference, 649-657.

[133] TRISCHLER, E., (1983), Testability Analysis and incomplete Scan Path, Proc. of International Conference on Computer-Aided Design, 38-39.

[134] TRISCHLER, E., (1980), Incomplete Scan Path with an Automatic Test Generation Methodology, Digest of Papers 1980 Test Conference, 153-162.

[135] TOUBA, N. A, and McCLUSKEY, E. J., (1995), Test Point Insertion Based on Path Tracing, Proc. International Test Conference.

[136] TOUBA, N. A., and McCLUSKEY, E. J., (1994), Automated Logic Synthesis of Random Pattern Testable Circuits, Proc. of IEEE International Test Conference, 174-183.

[137] AIRIAU, R., BERGE, J. M., and OLIVE, V., Circuit Synthesis with VHDL, Kluwer Academic Publishers, ISBN 0-7923-9429-1.

[138] Test Compiler and Test Compiler Plus, Reference Manual, Version 3.0, December 1992, Synopsys Inc.

[139] WAGNER, K. D., et al., (1987), Pseudo-Random Testing, IEEE Transactions on Computers, Vol. c-36, No. 3, 332-343.

[140] WAICUKAUSKI, J.A., (1985), A Statistical Calculation of Fault Detection Probabilities by Fault Simulation, Proc. of IEEE International Test Conference, 779-784.

[141] WAICUKAUSKI, J. A., and LINDBLOOM, E., (1988), Fault Detection Effectiveness of Weighted Random Patterns, Proc. of International Test Conference, 245-255.

[142] WANG, L. T., and McCLUSKEY, E. J., (1986), Condensed linear Feedback Shift register (LFSR) Testing: A Pseudo-Exhaustive Test Technique, IEEE Transactions on Computers, C-35, 367-370.

[143] WUNDERLICH, H-J., (1988), Multiple Distributions for Biased Random Test Patterns, Proc. of International Test Conference, 236-244.

[144] WUNDERLICH, H-J, (1985), PROTEST: A Tool for Probabilistic Testability Analysis, Proceedings of ACM/IEEE Design Automation Conference, 204-211.

[145] YOUSSEF, M., (1991), Insertion de points de test et implantation des chaines de balayage partielles dans les circuits séquentiels, Mémoire de Maîtrise, département de génie électrique, Ecole Polytechnique de Montréal.

[146] YOUSSEF, M., SAVARIA, Y., and KAMINSKA, B., (1993), A Methodology for Efficiently Inserting and Condensing Test Points, IEE Proceedings-E, Vol. 140, No. 3, 154-160.

Annexes

Annexe 1. Controllability and Observability computation

1.0 Introduction

Dans cette annexe nous allons présenter toutes les formules de calcul de mesures de testabilité que nous avons développé dans notre méthode d'analyse de testabilité et d'insertion de points de test. Ces mesures de testabilité sont définies et calculées au niveau fonctionnel pour les différents types d'opérations VHDL sans connaître aucun détail de leur implantation au niveau portes logiques. Ceci nous permet de réduire les erreurs de calcul dues aux problèmes de reconvergence dans les circuits décrits au niveau portes logiques. Ces mesures consistent à calculer la contrôlabilité (observabilité) des sorties (entrées) de chaque opération VHDL. Cet annexe présenté en anglais est un rapport technique interne.

1.1 Controllability Calculation

1.1.1 Controllability computation for an n-bit adder

We wish to compute the Controllability on the outputs of an n-bit adder, given the Controllability on its inputs, assuming that the inputs are independent. When two n-bit binary numbers a and b are added, each bit of the sum s is a function of the corresponding bits of a and b and of the carry from the next less significant bits. The truth table for the i^{th} sum bit s_i and the $(i+1)^{th}$ carry bit c_{i+1} are shown in Table 1.

Table 1: The truth table of 1-bit adder

c_i	a_i	b_i	s_i	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

The 1-Controllability measures $C_1(s_i)$ and $C_1(c_{i+1})$ can be computed by summing up the minterms leading to 1, as follows:

$$C_1(s_i) = C_0(c_i) \times (C_1(a_i) \times C_0(b_i) + C_0(a_i) \times C_1(b_i)) + C_1(c_i) \times (C_0(a_i) \times C_0(b_i) + C_1(a_i) \times C_1(b_i)).$$

$$C_1(c_{i+1}) = C_0(c_i) \times C_1(a_i) \times C_1(b_i) + C_1(c_i) \times (1 - C_0(a_i) \times C_0(b_i))$$

Since $C(c_i) = 1 - C_1(c_i)$, $C_0(a_i) = 1 - C_1(a_i)$ and $C_0(b_i) = 1 - C_1(b_i)$, $C_1(s_i)$ and $C_1(c_{i+1})$ can be rewritten as follows:

$$C_1(s_i) = \alpha + C_1(c_i) - 2 \times (\alpha \times C_1(c_i)) \quad \text{(A.1.1)}$$

$$C_1(c_{i+1}) = \alpha \times C_1(c_i) + C_1(a_i) \times C_1(b_i) \quad \text{(A.1.2)}$$

$$\text{where } \alpha = C_1(a_i) + C_1(b_i) - 2 \times C_1(a_i) \times C_1(b_i). \quad \text{(A.1.3)}$$

Note that α is the probability that $(a_i \oplus b_i) = 1$ and, consequently, $C_1(s_i)$ is the probability that $(a_i \oplus b_i \oplus c_i) = 1$.

To compute the Controllability of each output of an n-bit adder, we can use a cascade of n full adders configured as a ripple-carry adder which is reproduced again in Figure A.1.4. There is no reconvergent fanout in this circuit and all inputs are independent, hence the Controllability computed on this tree structure is exact. To find the Controllability measure $C_1(s_i)$ at the output s_i, Equations (A.1.1), (A.1.2) and (A.1.3) can be used for each 1-bit adder, and bit i can be evaluated when bits 0, 1, ..., $i-1$ have been computed. The calculation can be made in linear time in terms of the number of inputs. The same structure can be used to compute the Controllability of a subtractor using 2's-complement representation of negation.

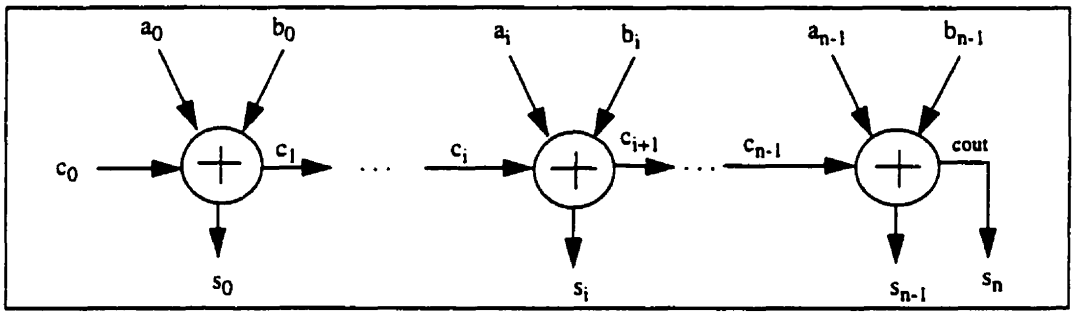

Figure A.1.4 A ripple-carry adder composed of n full adders

1.1.2 Controllability of the outputs of an n-bit multiplier

We wish to compute the Controllability on the outputs of an n-bit multiplier as shown in Figure A.1.5. This computation is much more complicated that for an adder due to the dependencies between the multiplier blocks. Unlike the adder, the multiplier cannot be decomposed into independent blocks.

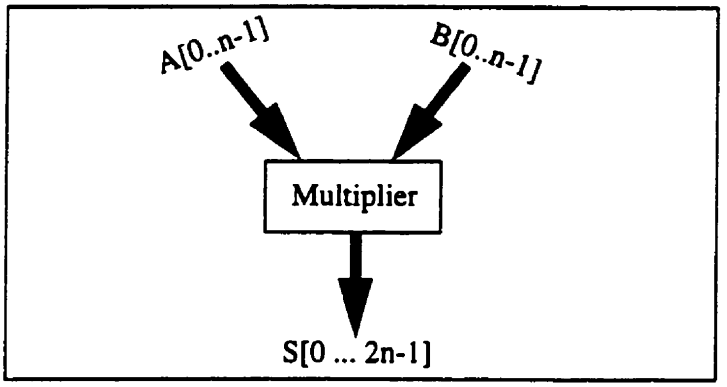

Figure A.1.5 An n-bit multiplier

Given the controllability measures of the inputs, the exact controllability measures on the outputs can be found by performing 2^{2n} operations.

The computation of the probability of occurrence of each number A ($0 \leq A < 2^n-1$) is based on the controllability measure on each input bit. Let us give an example. Suppose that n = 2, then for the inputs:

P(A=0) = P(A[0]=0) * P(A[1]=0) P(B=0) = P(B[0]=0)*P(B[1]=0)

P(A=1) = P(A[0]=1)*P(A[1]=0) P(B=1) = P(B[0]=1)*P(B[1]=0)

P(A=2) = P(A[0]=0)*P(A[1]=1) P(B=2) = P(B[0]=0)*P(B[1]=1)

P(A=3) = P(A[0]=1)*P(A[1]=1) P(B=3) = P(B[0]=1)*P(B[1]=1)

The probability to have S = 2 is computed as follows:

P(S=2) = P(A=1)*P(B=2) + P(A=2)*P(B=1).

This method based on the truth table is exact for an *n*-bit multiplier with $n \leq 10$. However, when *n* exceeds 10, the memory requirements and the CPU time can become very excessive.

One solution to reduce this complexity is to decompose the multiplier into a number of smaller multipliers. For example, an n-bit multiplier can be composed using 4 m-bit multipliers (with m = n/2) as compactly described in Figure A.1.6. This method does not compute the exact controllability but gives, in our opinion, a good measure of controllability.

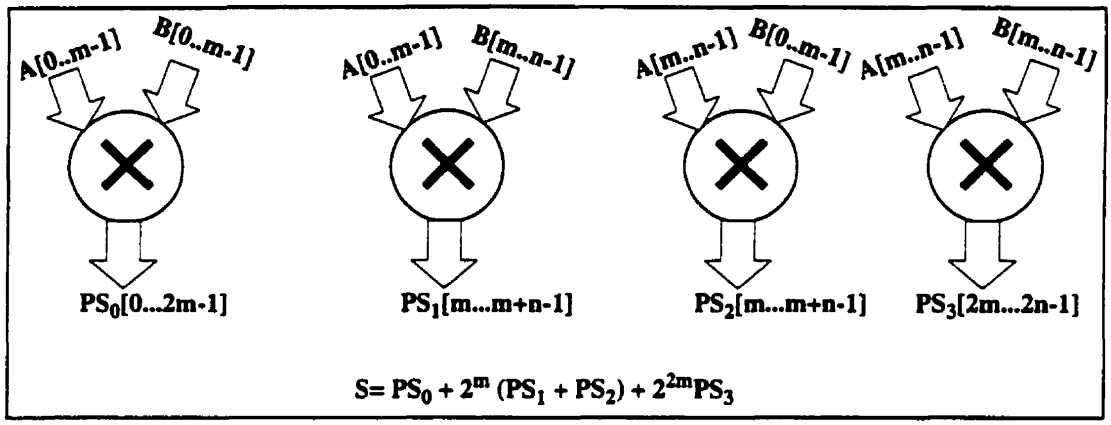

Figure A.1.6 n-bit multiplier composed of 4 m-bit multiplier (m = n/2).

1.1.3 Controllability of the output of an n-bit comparator

When two unsigned numbers A and B are compared, the relations of interest are (A > B), (A >= B), (A = B), ($A \neq B$), (A <= B) and (A < B). We compute the Controllability of (A = B) and (A < B), and deduce the Controllability of the other ones, e.g., $C_1(A > B) = C_0(A <= B)$. Let the two n-bit numbers to be compared have the form: $A = A_{n-1}\ A_{n-2}\ ..\ A_0$ and $B = B_{n-1}\ B_{n-2}\ ...\ B_0$.

The 1-Controllability of the output of a 1-bit equality comparator is:

$$C_1(A_i = B_i) = 1 - C_1(A_i \oplus B_i) \qquad (\forall\ \ i = 0, \ldots, n-1) \qquad \text{(A.1.7)}$$

Therefore the Controllability of the output of an n-bit equality comparator becomes:

$$C_1(A=B) = \prod_{i=0}^{n-1} C_1(A_i = B_i) \qquad \text{(A.1.8)}$$

The logic equation for (A < B) may be written as:

$$(A<B) = ((A_{n-1}<B_{n-1}) \vee (A_{n-1}=B_{n-1}) \wedge (A_{n-2}<B_{n-2})) \ldots \vee$$

$$((A_{n-1}=B_{n-1}) \wedge (A_{n-2}=B_{n-2}) \wedge \ldots \wedge (A_1=B_1)) \wedge (A_0<B_0)\,. \qquad \text{(A.1.9)}$$

Thus, 1-Controllability of (A < B) is given by the following formula:

$$C_1(A<B) = C_1(A_{n-1}<B_{n-1}) + \qquad \text{(A.1.10)}$$

$$\sum_{i=1}^{n-1}\left(C_1(A_{n-1-i}<B_{n-1-i}) \times \prod_{j=1}^{i} C_1(A_{n-j}=B_{n-j}) \right)$$

The other comparator expressions can be derived as follows:

$$C_1(A \le B) = C_1(A<B) + C_1(A=B) \qquad \text{(A.1.11)}$$

$$C_1(A>B) = 1 - C_1(A \le B) \qquad \text{(A.1.12)}$$

$$C_1(A \ge B) = 1 - C_1(A<B) \qquad \text{(A.1.13)}$$

1.1.4 Controllability of the output of a multiplexer

Consider a multiplexer of n control inputs ($c_0\, c_1 \ldots c_{n-1}$) and 2^n possible data inputs, ($A_0\, A_1 \ldots A_{2^n-1}$) in which each input A_i and the corresponding output consist of m bits. The 1-Controllability of the output is taken as the sum of the 1-Controllabilities of the inputs combined with the 1-Controllabilities of the control inputs. The general formula to compute the Controllability at the output is as follows:

$$C_1(S(k)) = \sum_{i=0}^{2^n-1} C_1(A_i(k)) \times C_1(control, A_i(k)) \text{ , where} \qquad \text{(A.1.14)}$$

$k = 0, 1, \ldots m-1$, $i = 0, 1, \ldots, 2^n - 1$, m is the number of bits of each input and the output, and $C_1(control, A_i(k))$ is the 1-Controllability on the control inputs such that data input $A_i(k)$ is selected to the output $S_i(k)$.

1.1.5 Controllability of a fan-out stem

A fan-out stem is a point where a driving signal is connected to more than one combinational or register inputs, these inputs are then the fan-out branches.

The Controllability of a fan-out branch is equal to the Controllability of its stem.

1.1.6 Controllability computation for standard logic gates [15]

In the following section we give the 1-Controllability and Observability computation of standard logic gates, with n inputs $I_0, I_1, \ldots, I_{n-1}$ and one output S.

And gate

$$C_1(S) = \prod_{i=0}^{n-1} C_1(I_i) \qquad \text{(A.1.15)}$$

Or gate

$$C_1(S) = 1 - \prod_{i=0}^{n-1} (1 - C_1(I_i)) \qquad \text{(A.1.16)}$$

Nand gate

$$C_1(S) = 1 - \prod_{i=0}^{n-1} C_1(I_i) \qquad \text{(A.1.17)}$$

Nor gate

$$C_1(S) = \prod_{i=0}^{n-1} (1 - C_1(I_i)) \qquad \text{(A.1.18)}$$

Inverter gate (one input I)

$$C_1(S) = 1 - C_1(I) \qquad \text{(A.1.19)}$$

Xor gate with two inputs I_0 and I_1

$$C_1(S) = C_1(I_0) + C_1(I_1) - 2 \times C_1(I_0) \times C_1(I_1) \qquad \text{(A.1.20)}$$

1.2 Observability calculation

Combinational Observability $O(l, S)$ of a line l is defined as the probability that a signal change on this line will result in a signal change on an output S. For multiple output modules, the Observability of a line must be computed relative to each output and the maximum Observability value over all outputs is used.

1.2.1 Observability computation for an n-bit adder

Consider again the n-bit ripple-carry adder shown in Figure A.1.4, and let us compute the Observability of each input. According to the truth table of a 1-bit adder, the change on any input a_i, b_i or c_i is always observable at s_i. It follows that:

$$O(a_i, s_i) = O(b_i, s_i) = O(c_i, s_i) = O(s_i), \quad i = 0, \ldots, n-1 \qquad \text{(A.1.21)}$$

The Observability of an input at level i at the other outputs k, $k > i$, depends on the propagation of the carry from stage i to these outputs.

For instance, to observe a_i at s_k such that $k = i+1, \ldots, n$, we have to set $(b_i \oplus c_i) = 1$ and $(a_j \oplus b_j) = 1$, for all j such that $j = i+1, \ldots, k-1$. Equation (A.1.22) gives the general formula to compute the Observability of each input a_i at outputs k, such that $k = i+1, \ldots, n$.

$$O(a_i, s_k) = \left[C_1(c_i \oplus b_i) \times \prod_{j=i+1}^{k-1} C_1(a_j \oplus b_j) \right] \times O(s_k), \quad i = 0, \ldots n-1, \quad \text{(A.1.22)}$$

The resulting Observability value of input i is the maximum value of Equations (A.1.21) and (A.1.22), i.e., $O(a_i) = max_{i \le k \le n}[O(a_i, s_k)]$. (A.1.23)

Similar formulas can be used to compute the $O(b_i, s_k)$ and $O(c_i, s_k)$ (in particular for c_0).

1.2.2 Observability of the inputs of an n-bit multiplier

To compute the Observability of the multiplier, we use the same method used for computing the Controllability of the inputs. For an n-bit multiplier with n <= 10, we use the corresponding truth table of the multiplier to compute the observability of each bit input. However, when n is over 10, we decompose the multiplier into small blocks as shown in Figure A.1.6 in this Appendix. Then the observability is propagated through the multipliers and adders blocks.

1.2.3 Observability of the inputs of an n-bit comparator

Consider a comparator with two inputs A and B of the form $A = A_{n-1} A_{n-2} .. A_0$ and $B = B_{n-1} B_{n-2} ... B_0$, and output S.

Equality comparator (A = B)

The Observability of each input A_i or B_i at the output S of the comparators (A = B) and $(A \neq B)$ is computed using Equation (A.1.24). The same Equation can be used to compute the Observability of B_i by substituting B_i for A_i.

$$O(A_i, S) = \left[\prod_{j=0/i\neq j}^{n-1} C_1 (A_j = B_j) \right] \times O(S) \qquad \text{(A.1.24)}$$

Less than comparator (A < B)

To compute the observability of inputs of the (A < B) comparator, let $A_+ = A_{n-1}A_{n-2}...A_{i+1}$ and $A_- = A_0A_1...A_{i-1}$. Define B_+ and B_- in a similar fashion. The operation of this comparator is not commutative and thus different formulas are used to compute the Observability of A and B. To observe the input bit A_i, we have to maintain $(A_+ = B_+)$ and depending on the value of bit B_i we have to inspect the relation between A_- and B_-.

The Observability of A_i is computed using the following Equations:

$O(A_i, S) = O(S) \times C_1 (A_+ = B_+) \times \alpha$ for $i = 1, 2, \ldots, n-2$, where

$\alpha = C_1 (A_- \geq B_-) \times C_1 (B_i) + C_1 (A_- < B_-) \times (1 - C_1 (B_i))$,

$O(A_i, S) = O(S) \times C_1 (A_+ = B_+) \times C_1 (B_i)$ for $i = 0$, and

$O(A_i, S) = O(S) \times \alpha$ for $i = n-1$, where

$\alpha = C_1 (A_- \geq B_-) \times C_1 (B_i) + C_1 (A_- < B_-) \times (1 - C_1 (B_i))$.

By the same analysis, we compute the Observability value of the other comparator types which are presented next.

Less than or equal comparator (A <= B)

The Observability of A_i is computed using the following Equations:

$O(A_i, S) = O(S) \times C_1(A_+ = B_+) \times \alpha$ for $i = 1, 2, \ldots, n-2$, where

$\alpha = C_1(A_- \leq B_-) \times C_0(B_i) + C_1(A_- > B_-) \times C_1(B_i)$,

$O(A_i, S) = O(S) \times C_1(A_+ = B_+) \times C_0(B_i)$ for $i = 0$, and

$O(A_i, S) = O(S) \times \alpha$ for $i = n-1$, where

$\alpha = C_1(A_- \leq B_-) \times C_0(B_i) + C_1(A_- > B_-) \times C_1(B_i)$.

Greater than comparator (A > B)

The Observability of A_i is computed using the following Equations:

$O(A_i, S) = O(S) \times C_1(A_+ = B_+) \times \alpha$ for $i = 1, 2, \ldots, n-2$, where

$\alpha = C_1(A_- \leq B_-) \times C_0(B_i) + C_1(A_- > B_-) \times C_1(B_i)$,

$O(A_i, S) = O(S) \times C_1(A_+ = B_+) \times C_0(B_i)$ for $i = 0$, and

$O(A_i, S) = O(S) \times \alpha$ for $i = n-1$, where

$\alpha = C_1(A_- \leq B_-) \times C_0(B_i) + C_1(A_- > B_-) \times C_1(B_i)$.

Greater than or equal comparator (A >= B)

The Observability of A_i is computed using the following Equations:

$O(A_i, S) = O(S) \times C_1(A_+ = B_+) \times \alpha$ for $i = 1, 2, \ldots, n-2$, where

$\alpha = C_1(A_- < B_-) \times C_0(B_i) + C_1(A_- \geq B_-) \times C_1(B_i)$,

$O(A_i, S) = O(S) \times C_1(A_+ = B_+) \times C_1(B_i)$ for $i = 0$, and

$O(A_i, S) = O(S) \times \alpha$ for $i = n-1$, where

$\alpha = C_1(A_- \leq B_-) \times C_0(B_i) + C_1(A_- > B_-) \times C_1(B_i)$.

1.2.4 Observability of multiplexer inputs

Consider again a multiplexer consisting of n control inputs ($c_0\, c_1 ... c_{n-1}$) and 2^n data possible inputs, ($A_0\, A_1\, ...\, A_{2^n-1}$), in which each input A_i and the output consist of m bits. The Observability computation for the data inputs is:

$O(A_i(k)) = O(S(k)) \times C_1(control, A_i(k))$, where $C_1(control, A_i(k))$ is the 1-Controllability of the control inputs such that $A_i(k)$ is connected to $S_i(k)$,

$k = 0, 1, ...m-1$ and $i = 0, 1, ..., 2^n - 1$.

The observability of the control inputs is more complicated than of the data inputs. For example, consider the case of a 4:1 multiplexer composed of 4 inputs A_0, A_1, A_2, A_3, one output S, and two control inputs c_0, c_1. For simplicity assume, m = 1. To observe c_0, we must consider the possible values of c_1. If c_1 is 0 a change on c_0 from 0 to 1 will change the output from A_0 to A_1. Therefore, to observe c_0 at S when c_1 is 0 requires $(A_0 \oplus A_1) = 1$. Similarly, $c_1 = 1$ requires $(A_2 \oplus A_3) = 1$. The same analysis can be used for the control input c_1 by considering the possible values of c_0. The Observability formulas for c_0 and c_1 are thus as follows:

$$O(c_0, S) = O(S) \times [\,(1 - C_1(c_1)) \times C_1(A_0 \oplus A_1) + \quad C_1(c_1) \times C_1(A_2 \oplus A_3)\,].$$

$$O(c_1, S) = O(S) \times [\,(1 - C_1(c_0)) \times C_1(A_0 \oplus A_2) + \quad C_1(c_0) \times C_1(A_1 \oplus A_3)\,].$$

For a general multiplexer, the observability of a control input c_i is computed using the following formula:

$$O(c_i) = Max_k\left(\sum_{p=0}^{2^{n-i-1}-1}\left(\sum_{j=p\times 2^{i+1}}^{p\times 2^{i+1}+2^i-1} C_1(\beta_j)\times C_1(control,\beta_j)\right)\times O(S(k))\right) \quad \text{(A.1.25)}$$

where $k = 0, 1, \ldots m-1$, $\beta_j = \left[A_j(k) \oplus A_{j+2^i}(k)\right]$ and

$C_1(control, \beta_j)$ is the 1-Controllability of the control inputs such that the inputs $A_j(k)$ and $A_{j+2^i}(k)$ are mutually exclusive, i.e., $\beta_j = 1$.

1.2.5 Observability computation of a fanout stem [15]

The observability of a fanout stem, s, is related to the observability at each fanout branch of the fanout stem. Let n be the number of fanout branches for a fanout stem and let $O(b_i)$ be the observability value at the i^{th} branch. Then the observability of the fanout stem is computed using the following Equation:

$$O(s) = 1 - \prod_{i=1}^{n}(1 - O(b_i)) \quad \text{(A.1.26)}$$

1.2.6 Observability computation for logic gates [15]

Consider gates with n inputs $I_0, I_1, \ldots, I_{n-1}$ and one output S.

And gate

$$O(I_j) = O(S)\times \prod_{(i=0)/(i\neq j)}^{n-1} C_1(I_i) \quad \text{(A.1.27)}$$

Or gate

$$O(I_j) = O(S) \times \prod_{(i=0)/(i\neq j)}^{n-1} (1 - C_1(I_i)) \qquad \text{(A.1.28)}$$

Nand gate

$$O(I_j) = O(S) \times \prod_{(i=0)/(i\neq j)}^{n-1} C_1(I_i) \qquad \text{(A.1.29)}$$

Nor gate

$$O(I_j) = O(S) \times \prod_{(i=0)/(i\neq j)}^{n-1} (1 - C_1(I_i)) \qquad \text{(A.1.30)}$$

Invert gate

$$O(S) = O(I) \qquad \text{(A.1.31)}$$

Xor gate of n inputs

$$O(I_j) = O(S) \quad \text{with} \quad j = 0, 1, \ldots n-1 \qquad \text{(A.1.32)}$$

1.3 Conclusion

Nous avons présenté les formules de calcul de mesures de testabilité au niveau fonctionnel des opérations VHDL. Ces mesures sont propagées à travers le graphe acyclique dirigé (DAG) afin d'identifier tous les bits des signaux et variables qui sont difficiles à tester.

Annexe 2. Construction of Directed Acyclic Graph

2.0 Introduction

Dans cet annexe qui constitue un rapport technique interne, nous décrivons toutes les étapes de construction du graphe acyclique dirigé (Direct Acyclic Graph - DAG). Ce dernier est obtenu à partir de la spécification VHDL du circuit. Les noeuds internes du DAG représentent les différentes opérations VHDL et les arcs représentent les signaux et les variables de la spécification VHDL. Les noeuds sources (puits) du graphe représentent les entrées (sorties) primaires et pseudo-primaires du circuit. Les entrées/sorties pseudo-primaires sont représentées par des registres qui sont synthétisés à partir de la spécification VHDL du circuit. Le DAG est utilisé pour propager les mesures de testabilité qui sont décrites en annexe 1. Un certain nombre d'exemples sont présentés pour illustrer la construction du DAG.

2.1 DAG construction steps

The construction of the DAG is based on the steps listed below. Next, we describe each step in detail.

1.Generate Control and Data Flow Graph (CDFG) for each process of the VHDL specification.

2.Unroll all for...loops statements by adding new nodes to the CDFG.

3.Expand procedures/functions by adding new nodes to the CDFG.

4.Convert data types to bits.

5.Translate the resulting CDFG into a DAG.

6.Connect DAG graphs of individual process to produce the global DAG.

2.1.1 Generation of CDFG

Each VHDL process can be transformed into a Control Flow Graph (CFG) to represent the control flow of operations. The CFG is a directed graph defined as:

$$CFG = (V, E) \text{ , where}$$

V is a set of nodes corresponding to the different VHDL sequential statements:

$$V = V_w \cup V_b \cup V_l \cup V_e \text{, where}$$

V_w is the set of synchronization nodes (wait statements),

V_b is the set of conditional statement nodes (if, case),

V_l is the set of for...loop statement nodes,

V_e is the set of other nodes (signal/variable assignments, procedure calls,...).

and E is the set of edges representing the flow of control.

Each node of a CFG is associated with a Data Flow Graph (DFG). Each node of DFG represents one operation in the VHDL specification, and DFG edges represent signals and variables connecting the nodes. There is an edge from operation o_i to operation o_j if the result of operation o_i is input to operation o_j. A CFG in which each node is a DFG is called a CDFG. Figure A.2.1 shows a VHDL specification and its corresponding CDFG. The node numbers are indicated as comment (starting with "--"). Node S1 represents the "wait" statement and node S2 corresponds to the conditional "if" statement. Signal assignment (variable assignments) are represented by nodes S4, S6 and S7 (S3 and S5). Each node of CDFG is associated with a Data Flow Graph describing the data dependencies in the statement, an example is given in Figure A.2.1 for node S5.

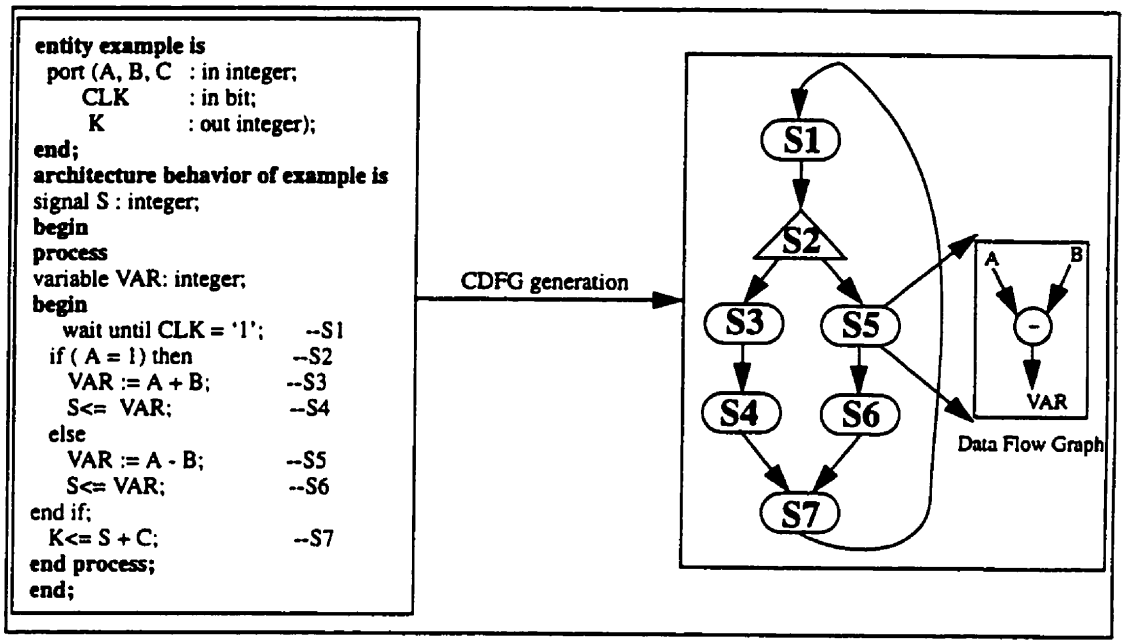

Figure A.2.1 An example of a CDFG generation.

2.1.2 Loop unrolling

For...loop statements are used to repeat a sequence of operations for a constant number of times. The result of synthesis would be a replication of the hardware corresponding to the statements inside the loop, one for each iteration. Two sequential statements are used only with loops: the **next** statement, which skips the remainder of the current loop iteration, and the **exit** statement, which terminates the loop. These statements are modeled as conditional if-branches. After unrolling the loop, the CDFG is augmented by new nodes corresponding to each iteration of the loop.

2.1.3 Expansion of procedures/functions

Each procedure call is expanded in-line. The contents of the procedure are first copied into the process in place of the call. Then, the actual parameters are substituted for the formal parameters of the procedure.

Functions are expanded in a similar fashion except that a function is expanded immediately before the expression that calls it. Here also, the CDFG is augmented by new nodes due to procedures/functions expanding.

2.1.4 Data type conversion

If already not declared as such, all VHDL data types are converted into bits. The main issue with enumerated types is the encoding of the possible values. By default we can convert them into bit-vectors whose length is determined by the minimum number of bits required to code the number of enumerated values, and the actual code assigned to each value corresponds to the binary representation of the position in the type declaration (starting from zero).

Integer subtypes are defined using subranges that impose bounds on the possible values. They are encoded using bit-vectors whose length is the minimum necessary number of bits to hold the defined range. If the range includes negative numbers, it is encoded as a 2's-complement bit-vector. In general, bits of an integer type value are not directly accessible and depends on the synthesis tool we use. No assumption is made on the location of these bits. Specific functions translating an integer into bit vector and back are defined and exported by packages provided in the synthesis tool environment. In our method, we use the standard packages of IEEE in which functions translating an integer to bit_vector and back are defined. We use these function in order to access a specific bit of a signal/variable declared as an integer data type.

2.1.5 Translation of the CDFG into DAG

The next step is to translate the resulting CDFG of VHDL process into a DAG. Synchronous registers are inferred on signals and some variables assigned in a clocked process. However, once we separate the present and the next states of registers into separate nodes, we obtain a DAG. The source nodes are the primary inputs and present state values of registers and the sink nodes are the primary outputs and the next-state values. The pri-

mary inputs (outputs) of the DAG are all the input (output) signals of the entity port declaration, and they correspond to all signals that are only read (assigned) and not assigned (read) in the VHDL specification.

Pseudo-primary inputs (outputs) correspond to synthesized register outputs (inputs).

Identification of synthesized registers

Registers in a synthesized RTL VHDL specification are inferred on some signals/ variables used in clocked processes. We thus have to first identify clocked processes in the VHDL specification. In the following, we give some rules that determine when registers are inferred on signals/variables:

Rule1: All signals that are assigned new values within a clocked process are synthesized as outputs of registers.

Figure A.2.2 gives an illustration of this rule: Signals Q1 and Q2 in the specification in Figure A.2.2(a) are synthesized as registers (D-FFs) in Figure A.2.2(b). In the case of unclocked processes, all signals are synthesized as internal wires.

Other registers correspond to some of the variables assigned new values in clocked processes. Figure A.2.3 gives an example of a clocked process in which the variable VAR in Figure A.2.3(a) is synthesized as a wire in the circuit of Figure A.2.3(b). This is because the variable VAR is always the target of an assignment before being read. Therefore, no register is required to memorize its value from one clock cycle to the next. This interpretation of variables is expressed in the following rule. Like signals, all variables assigned new values in unclocked processes are synthesized as internal wires.

Rule2: Between two synchronization statements, if a variable is always the target of an assignment before being read, this variable does not infer any register and is mapped to a wire.

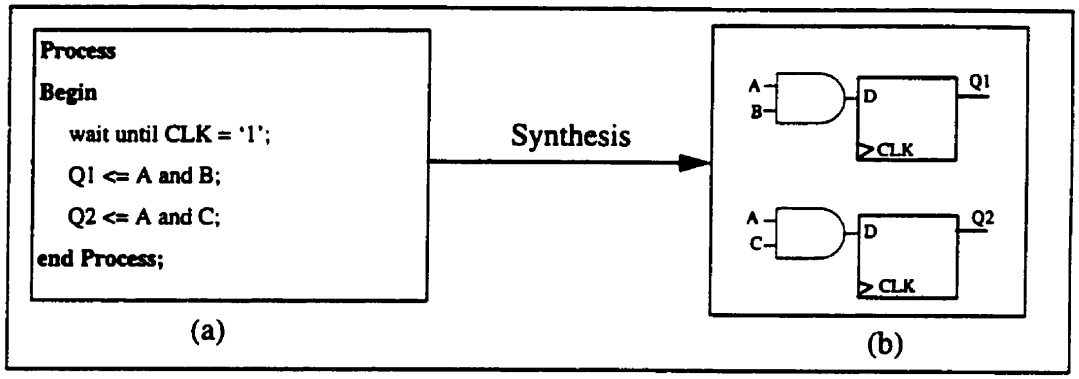

Figure A.2.2 Illustration of Rule 1.

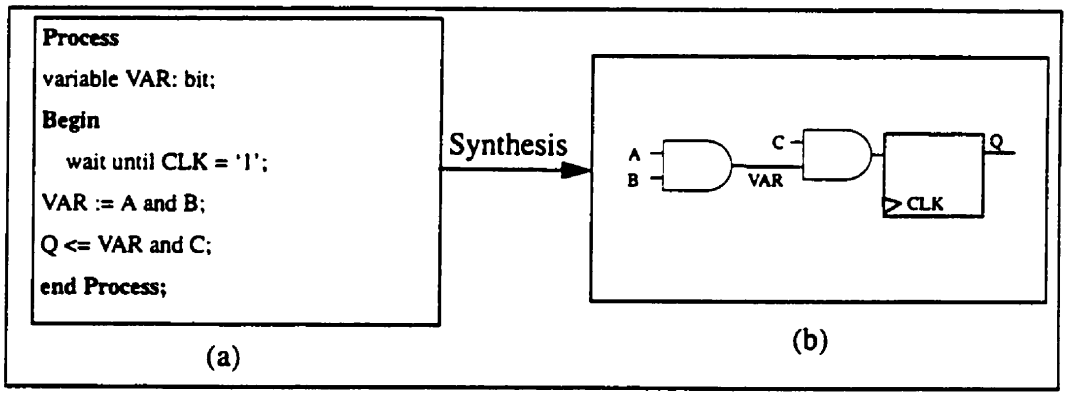

Figure A.2.3 Illustration of Rule 2

Now we give an example in which a variable in a clocked process is synthesized as a register, as shown in Figure A.2.4. Between two synchronization statements, the variable VAR in the VHDL specification in Figure A.2.4(a) is read at least once before appearing as the target of an assignment. Therefore, a register is needed to store the value of VAR. Note that in this example, reading a variable does not necessarily mean having it in the right hand side of an assignment. A "case" expression, an "if" condition or an "in" parameter of a procedure or a function are other ways of reading it. This leads to the following rule.

Rule 3: Between two synchronization statements, if a variable is read at least once before appearing as the target of an assignment, this variable implies storage of data and thus a register is synthesized.

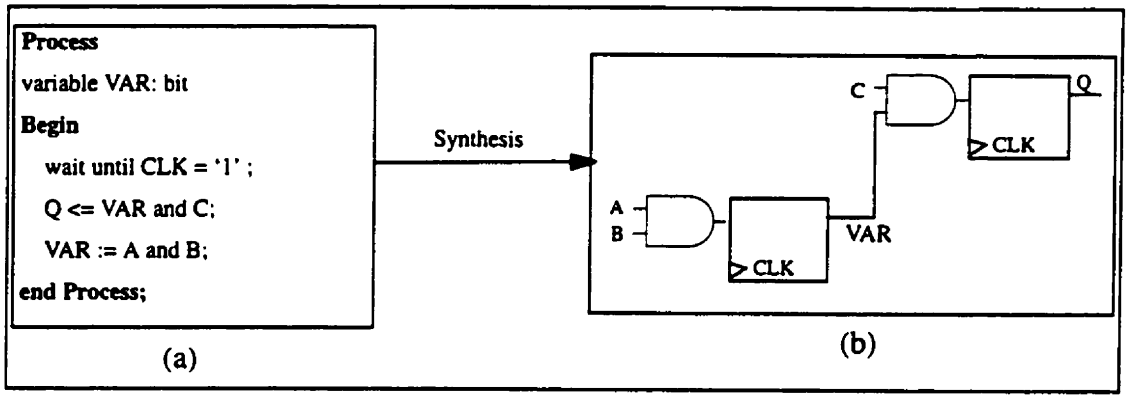

Figure A.2.4 Illustration of Rule 3.

Up to now we only considered processes with one wait statement placed as the first or as the last statement in the process. The same rules can be applied to identify registers in the case of clocked processes with one wait statement placed anywhere in the process. Figure A.2.5 gives an illustration of this case in which the variable X is synthesized as a wire, since between two consecutive wait statements, it is always set before being read. However, the variable Y infers a register since it is read at least once before being set to a new value. Note that signals Q1 and Q2 are synthesized as registers since they are assigned new values in the given process (Rule 1).

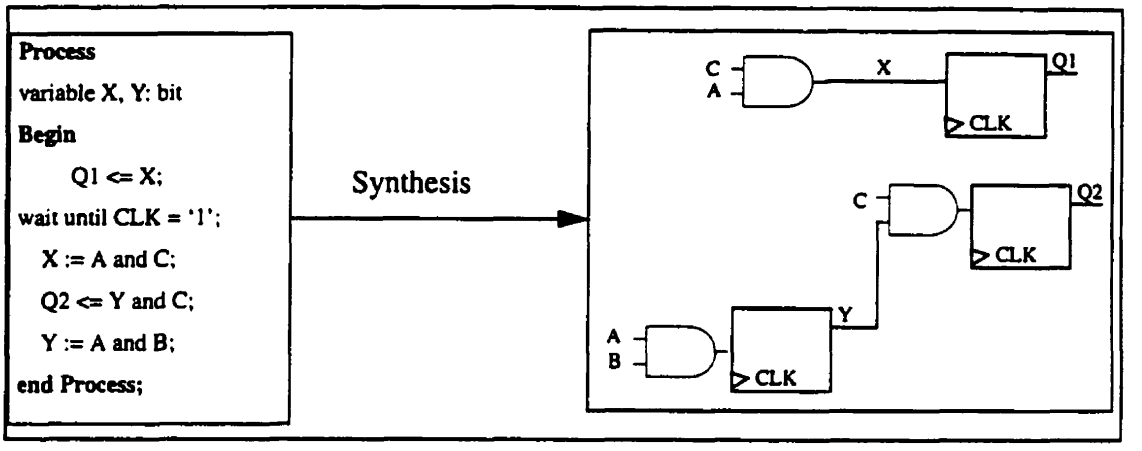

Figure A.2.5 Illustration of rule 3

Finally, Figure A.2.6 shows the case of a process with three wait statements synchronized on the same clock signal, CLK. Like clocked processes with one wait statement, the same rules can be applied to identify signals and variables that are synthesized as registers, i.e., between two consecutive synchronization statements we must analyze the use of variables. In Figure A.2.6, variable X is synthesized as a wire, because between two consecutive wait statements (w1 and w2), it is always set before being read. Variable Y is synthesized as a register, however, because it is read at least once before being set (between wait statements w2 and w3). Note that in the multiple wait statements, an implicit state variable is inferred whose bit-width is determined by the number of bits required to encode the number of wait statements in the process. In Figure A.2.6, two registers are required to encode the state of the process (w1, w2 and w3).

```
Process
Variable X, Y: BIT;
begin
w1: wait until CLK = '1';
        X := A and B;
        R <= C or X;
w2: wait until CLK = '1';
        S <= C or Y;
        Y := A and B;
w3: wait until CLK = '1';
        Y := A and C;
        Z <= C or Y;
end process;
```

Figure A.2.6 A VHDL process with multiple wait statement.

Register identification algorithm

We have seen how registers are inferred on some signals and variables in VHDL specifications. In the following, we give the main steps of an algorithm for identifying these signals and variables in a clocked process. We assume in the first step that a clocked process has been recognized in the VHDL specification. The algorithm will interface the intermediate representation (VIF) produced by the VHDL analyzer. Note that this algorithm is the same as used by most synthesis tools.

Algorithm 1: identification of registers in VDHL specification

```
Input: VHDL specification
begin
    Identify all clocked process in the VHDL specification
   For all clocked processes do {
     For all signals assigned in the process do
          infer a register;
     For all variables assigned in the process do {
          if between two consecutive wait statements the variable is always set before read then
                    not infer a register;
          else
                    infer a register;
    }
}
end;
```

Identification of multiplexers

In this section we give a translation for the VHDL conditional statements ("if" and "case"). A new node type is added to the DAG representing the multiplexer operation. In fact, each conditional statement ("if" or "case") is translated into a set of multiplexers with one output for each signal/variable assigned within the conditional statement. The condition expression translates into a set of nodes which feed the control input of each multiplexer. The data inputs to each multiplexer are fed from the nodes of the corresponding expression being assigned. Thus, to translate each conditional statement to a multiplexer operations, we have to find its scope, i.e., its corresponding end-statement. The CDFG generated by LEDA tools [48] needs some transformations in order to translate each conditional statement ("if", "case") to a multiplexer operation. Let us take an example to illustrate this transformation. Figure A.2.7(a) gives a CDFG with three conditional statements in which the corresponding end-statements of each conditional statements are not included in the original CDFG generated by LEDA tools.

The new CDFG including the end-statement nodes (shown shaded) of each conditional branch is given in Figure A.2.7(b). For each signal/variable used between an if-conditional statement and its corresponding end-statement, we associate a multiplexer operation. This transformation is implemented in an algorithm.

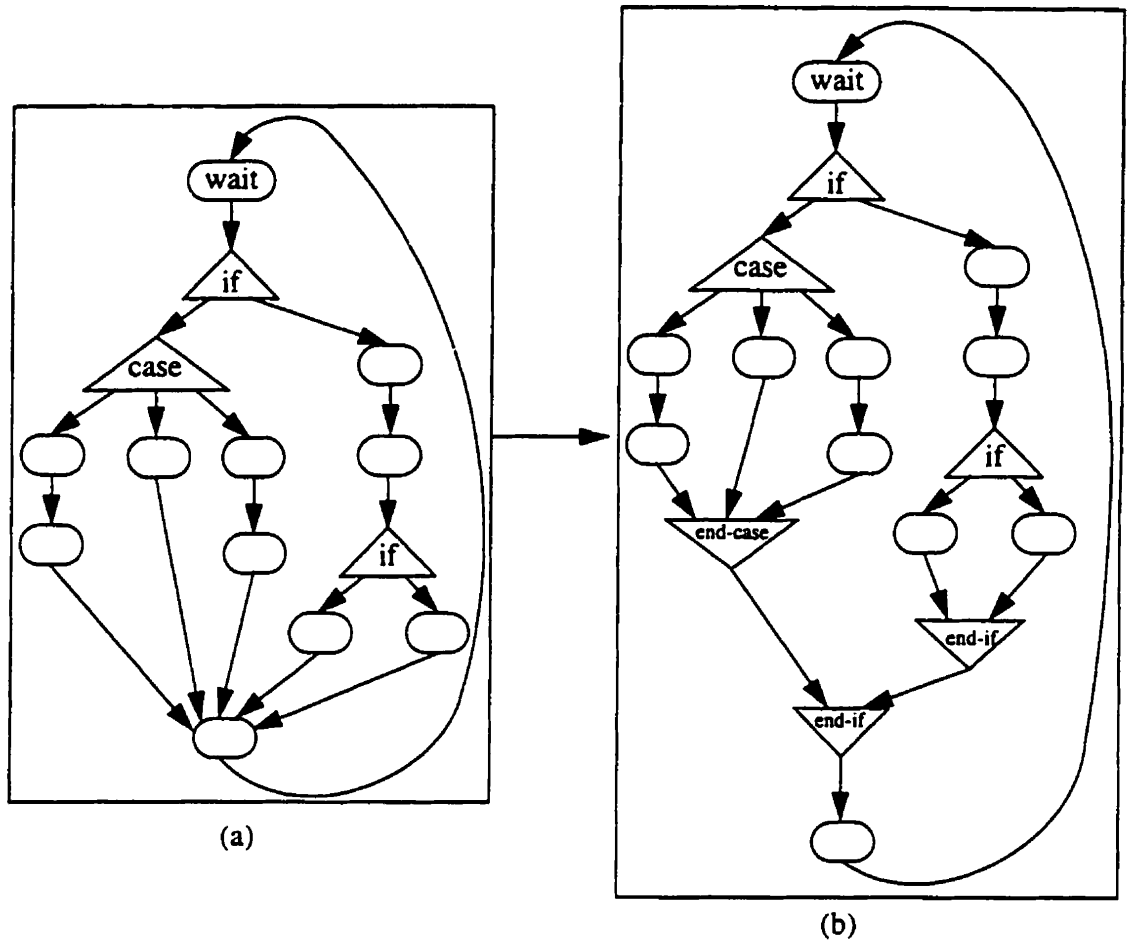

Figure A.2.7 (a) An example of a CDFG without end-statement nodes. (b) example of a modified CDFG including the corresponding end-statements

2.1.6 Connecting of processes

A VHDL specification consists of a set of interacting processes (clocked and unclocked). Entity ports and internal signals (declared in the VHDL architecture) are used to communicate between the processes. The connection between processes is represented as a directed graph. However, we have seen that each VHDL process can be transformed into a CDFG which is represented as a DAG. Hence, the global VHDL specification is also represented as a DAG.

2.1.7 Application

In this section we give some application examples of constructing the DAG used for TMs computation. Figure A.2.8 gives a VHDL example in which the complete DAG used for TMs computation is also presented. In this clocked process, signal S1 and K are synthesized as registers whereas the variable VAR as a wire. Note that signal S1 will be considered as a pseudo-primary input/output since it is declared as an internal signal in the VHDL specification. The next example is presented in Figure A.2.9. In this example, the variable VAR is assigned different expressions in the process and used by more than one signal. In this example, the variable VAR is read before it is used by signal S1 and thus it is synthesized as a register. Thus, the VAR expression to be used by S1 will be the last one taken by VAR when the process execution resumes. However, for signal K, variable VAR is assigned before being used by this signal and it is used as a wire for this signal. Figure A.2.10 illustrates a VHDL specification in which an if-statement is used. The complete DAG for TMs computation is also given in the corresponding Figure A.2.10. Note that for conditional statements (if, case, exit and next), a number of multiplexers will be added for appropriate signals. As in the example in Figure A.2.10, a mux is associated to signal K since its expression to be assigned depends on the condition in the if-statement. Figure A.2.11 gives a VHDL example with an unspecified if-statement. Signals and variables that are not assigned in all conditions of the if-statement will infer sequential loops to retain their last values. Figure A.2.12 gives an example of a for...loop statement with fixed and known bounds. Note that we unroll each loop declared in the VHDL specification and our method can support nested loops. Figure A.2.13 and Figure A.2.14 give the corresponding DAG for the next and exit statements respectively. Finally, Figure A.2.15 gives the DAG when a procedure is used in the VHDL specification in which a graph is associated to it.

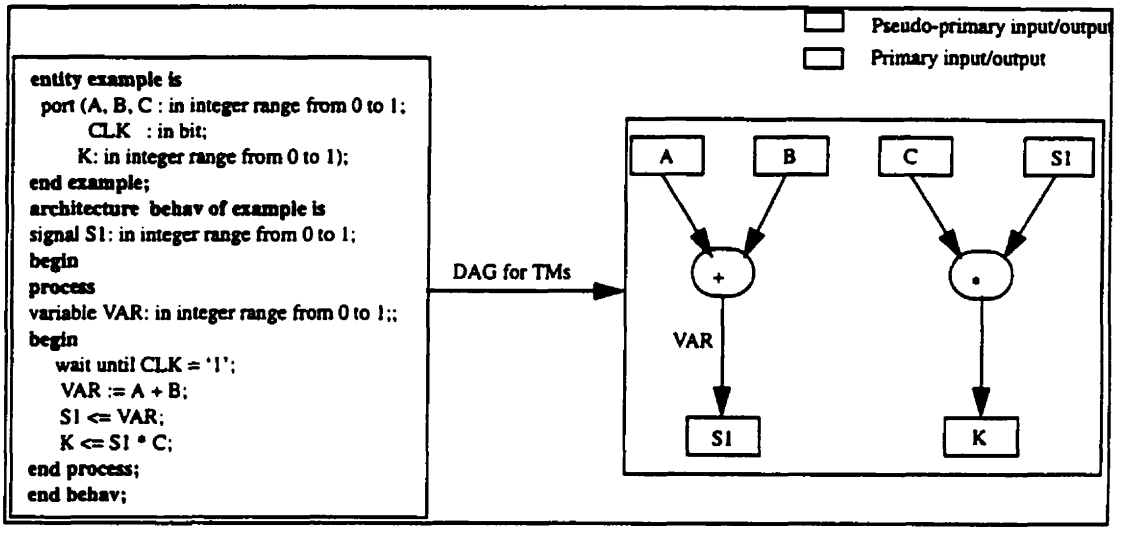

Figure A.2.8 A VHDL example of simple assignment statements.

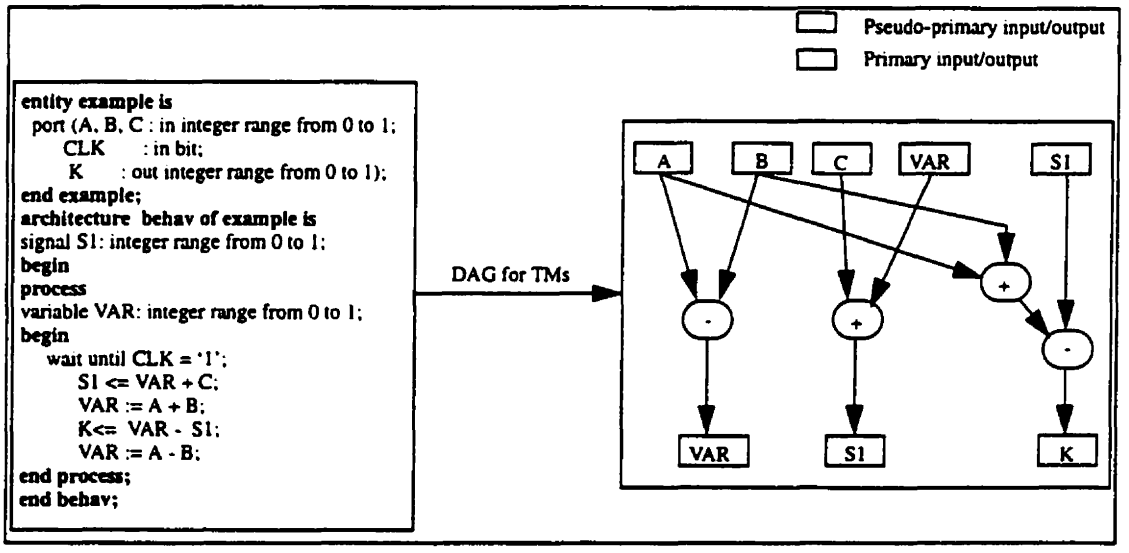

Figure A.2.9 A VHDL example of simple assignment statements.

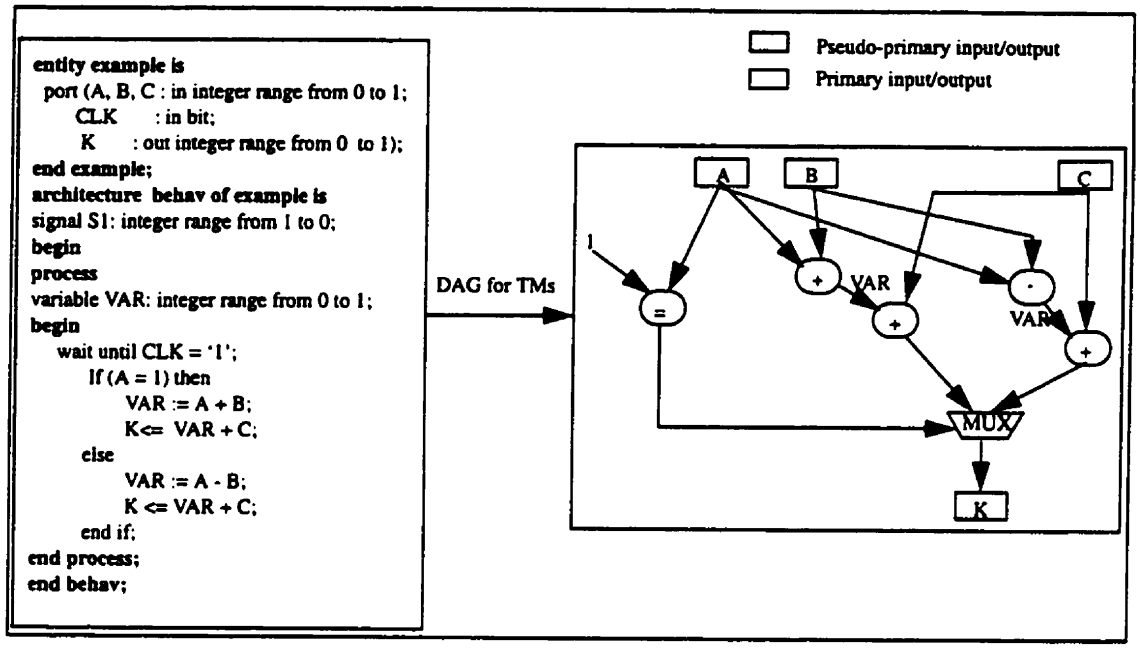

Figure A.2.10 A VHDL example with if-statement.

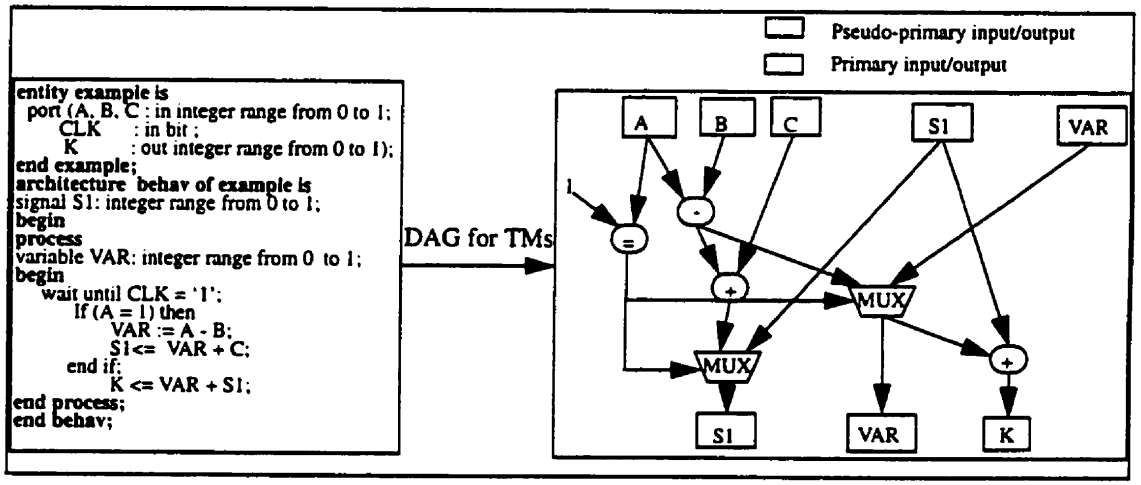

Figure A.2.11 A VHDL example with an unspecified if-statement.

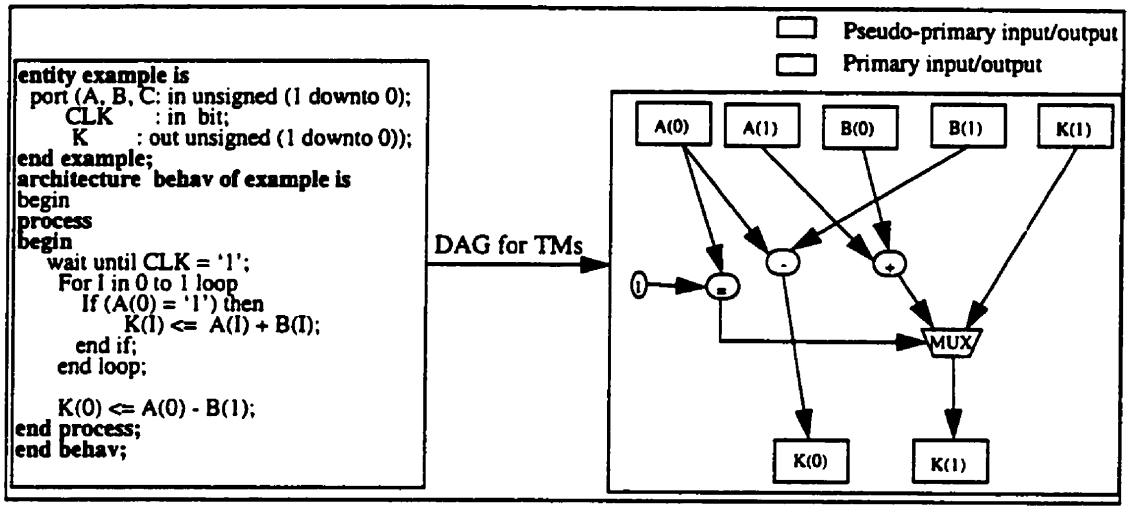

Figure A.2.12 A VHDL example with for...loop statement.

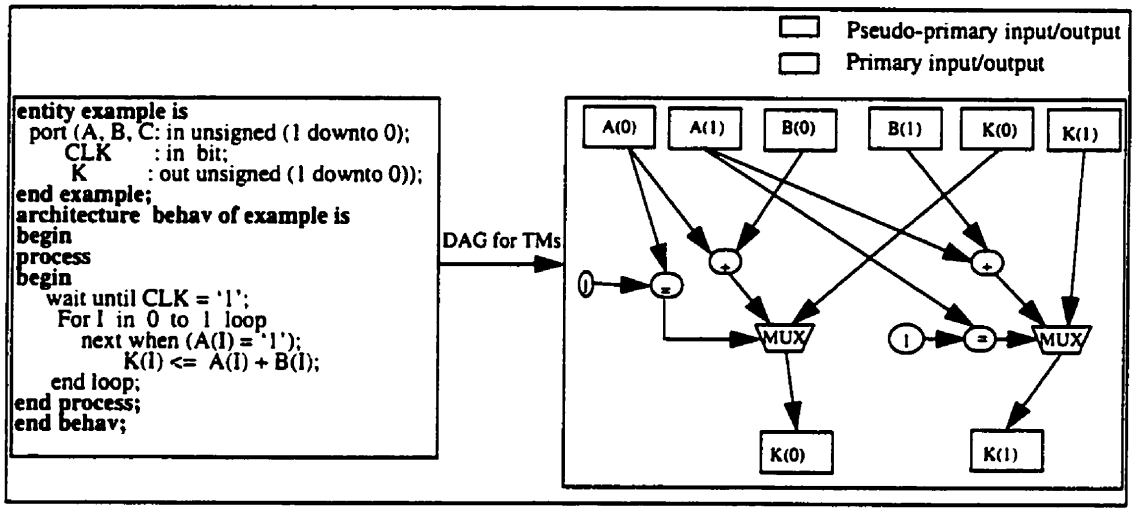

Figure A.2.13 A VHDL example with next statement.

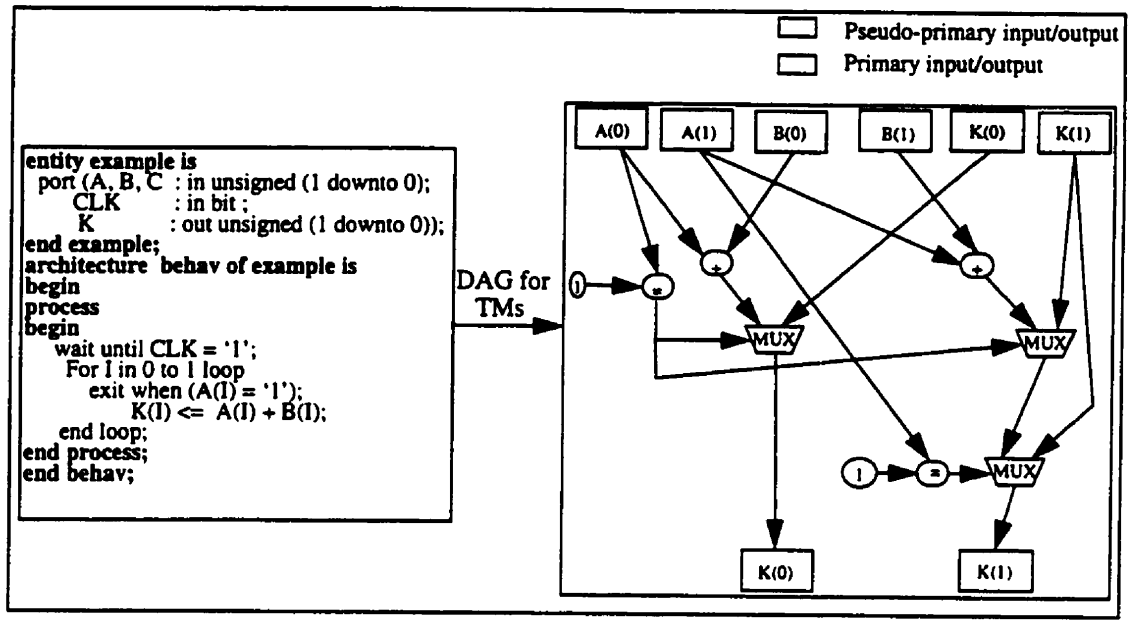

Figure A.2.14 VHDL example with exit statement.

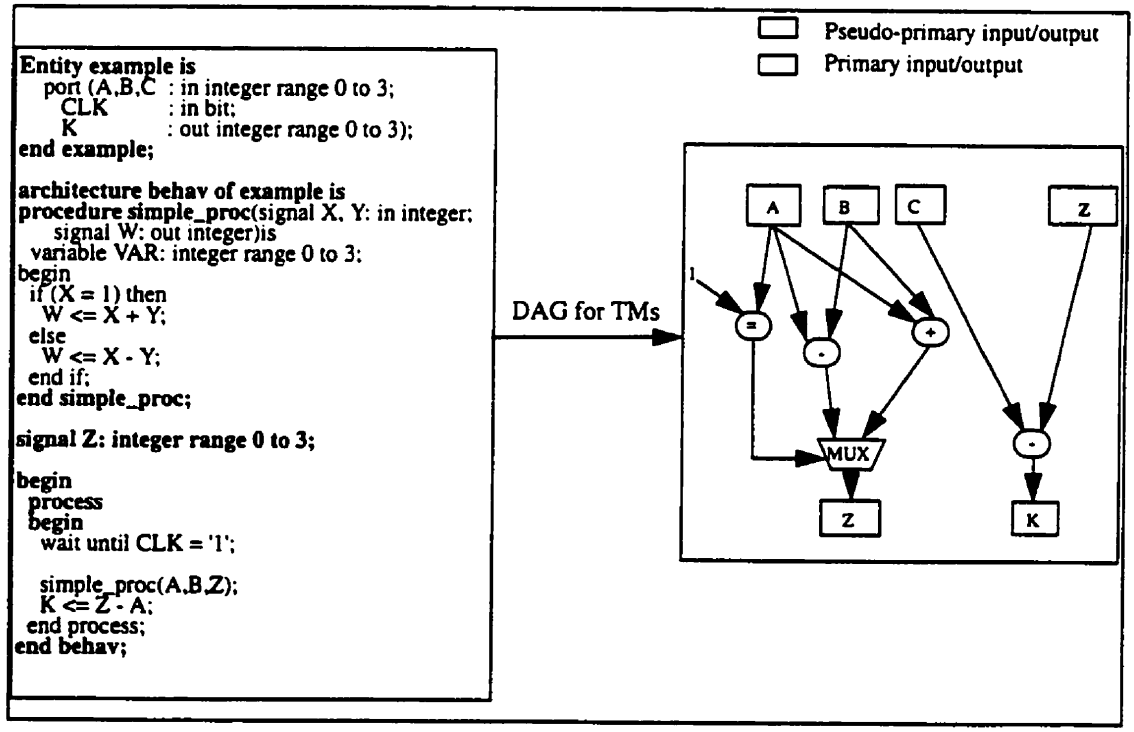

Figure A.2.15 VHDL example with a procedure call.

2.1.8 Conclusion

Dans cet annexe nous avons présenté les étapes principales de construction du DAG. Ce dernier est ensuite utilisé pour propager les mesures de testabilité à travers les opérations VHDL.

Annexe 3. Définition du package incluant les points de test

Dans cet annexe, nous décrivons le package incluant les définitions des points de test insérés au niveau VHDL. Chaque point de contrôle (d'observation) est décrit par une fonction (procédure) VHDL synthétisable appelée "Insert_Control_Point(...)" (Insert_Observation_Point(...)). On a aussi inclu une fonction (procédure) appelée Insert_Control_Equal(...) (Insert_Observation_Equal(...)), décrivant l'insertion de points de contrôle (observation) au niveau des signaux internes du comparateur d'égalité.

```
--------------------------------------------------------------------------------
-- Title      : Package of Test Points (Control and Observation points),
--              Copyright 1996.
-- Authors    : Samir Boubezari, PhD student, Ecole polytechnique de
--              Montréal
-- Supervisors : Eduard Cerny (Université de Montreal ) and Bozena Kaminska
--              (Ecole polytechnique)
-- Note       : Test Point Insertion project: logicVison and Ecole
--              polytechnique de Montréal
--------------------------------------------------------------------------------

Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
```

```
package Test_Points is

 -- Declaration of an enumerated type for the Control point
 -- (operations: OR, AND)
 type CONTROL_POINT is (OR_POINT, AND_POINT);
 -- Declaration of the positions of control points as an array
 -- of integers. This is only used in the modification of
 -- functional modules
 type POSITION_ARRAY is ARRAY(INTEGER range <>) of INTEGER;
 -- Declaration of an array of control points
 type Cont_TYPE_ARRAY is ARRAY(INTEGER range <>) of CONTROL_POINT;
 -- Declaration of an array of signals used as control inputs for
 -- test point insertion
 type SIG_ARRAY is ARRAY(INTEGER range <>) of BOOLEAN;
 -- Function to insert a control point in a signal/variable
 -- SIG_VAR declared as STD_LOGIC_VECTOR.

function Insert_Control_Point(
  SIG_VAR    :STD_LOGIC_VECTOR;
  CONT_TYPE  :CONTROL_POINT;
  POSITION   :INTEGER;
  TEST_IN    :STD_LOGIC;
  TM         :STD_LOGIC) return STD_LOGIC_VECTOR;
```

```
-- Function to insert a control point in a signal/variable
-- SIG_VAR declared as an INTEGER.
function Insert_Control_Point(
  SIG_VAR   :INTEGER;
  SIZE      :INTEGER;
  CONT_TYPE :CONTROL_POINT;
  POSITION  :INTEGER;
  TEST_IN   :STD_LOGIC;
  TM        :STD_LOGIC) return INTEGER;

-- Function to insert a control point in a signal/variable
-- SIG_VAR declared as BOOLEAN
function Insert_Control_Point(
  SIG_VAR   :BOOLEAN;
  CONT_TYPE :CONTROL_POINT;
  TEST_IN   :BOOLEAN;
  TM        :BOOLEAN) return BOOLEAN;

-- Function to insert a control point in a signal/variable
-- SIG_VAR declared as STD_LOGIC
function Insert_Control_Point(
  SIG_VAR   :STD_LOGIC;
  CONT_TYPE :CONTROL_POINT;
  TEST_IN   :STD_LOGIC;
  TM        :STD_LOGIC) return STD_LOGIC;
```

```
-- function called from function Insert_Control_Equal()
function unsigned_Insert_Control_Equal(
  L,R       :UNSIGNED;
  N         :INTEGER;
  P         :POSITION_ARRAY;
  CONT_TYPE  :Cont_TYPE_ARRAY;
  TEST_IN    :SIG_ARRAY) return BOOLEAN;

--function to insert N control points of type CONT_TYPE at positions
-- P on the internal signals of the equality comparator.
function Insert_Control_Equal(
  L,R       :STD_LOGIC_VECTOR;
  N         :INTEGER;
  P         :POSITION_ARRAY;
  CONT_TYPE  :Cont_TYPE_ARRAY;
  TEST_IN    :SIG_ARRAY;
  TM        :BOOLEAN) return BOOLEAN;

-- Procedure to insert an Observation point in a signal/variable
-- SIG_VAR declared as STD_LOGIC_VECTOR
Procedure Insert_Observation_Point(
  SIG_VAR     :STD_LOGIC_VECTOR;
  POSITION    :INTEGER;
  signal SCAN  :out STD_LOGIC);
```

```
  -- Procedure to insert an Observation point in a signal/variable
  -- declared as an INTEGER
  Procedure Insert_Observation_Point(
    SIG_VAR      :INTEGER;
    SIZE         :INTEGER;
    POSITION     :INTEGER;
    signal SCAN  :out STD_LOGIC);

  -- Procedure to insert an Observation point in a signal/variable
  -- SIG_VAR declared as STD_LOGIC
  Procedure Insert_Observation_Point(
    SIG_VAR      :STD_LOGIC;
    signal SCAN  :out STD_LOGIC);

  -- Procedure to insert an Observation point in a signal/variable
  -- SIG_VAR declared as BOOLEAN
  Procedure Insert_Observation_Point(
    SIG_VAR      :BOOLEAN;
    signal SCAN  :out BOOLEAN);
  -- Procedure to insert N Observation points of type CONT_TYPE
  -- at positions P on the internal signals of the equality comparator
  Procedure Insert_Observation_Equal(
    L, R         :STD_LOGIC_VECTOR;
    variable V   :out BOOLEAN;
    N            :INTEGER;
    P            :Position_array;
    signal SCAN  :out Sig_array);
end Test_Points;
```

```
package body Test_Points is

 -- A function to insert a control point at a given bit position
 -- of a signal/variable SIG_VAR which is declared as an array of bits
 -- (STD_LOGIC).
 function Insert_Control_Point(
   SIG_VAR   :STD_LOGIC_VECTOR; -- The signal/variable to be modified
   CONT_TYPE :CONTROL_POINT;    -- The type of the control point (AND, OR)
   POSITION  :INTEGER;          -- The position of the bit in SIG_VAR to modify
   TEST_IN   :STD_LOGIC;        -- Extra control input used for control insertion
   TM        :STD_LOGIC         -- TM = 1(0): Test Mode (Normal mode)
   ) return STD_LOGIC_VECTOR is
  variable VAR    : std_logic_vector(SIG_VAR'left downto SIG_VAR'right);
begin
  VAR := SIG_VAR;
  if (CONT_TYPE = OR_POINT) then
    VAR(POSITION) := (TM and TEST_IN) or VAR(POSITION);
  else
    VAR(POSITION) := (not TM or TEST_IN) and VAR(POSITION);
  end if;
  return (VAR);
end Insert_Control_Point;

-- A function to insert a control point at a given bit position
-- of a signal/variable SIG_VAR which is declared as an integer.
-- The function needs to convert the corresponding signal/variable
-- into a number of bits. This number is specified by the parameter
```

```
-- "SIZE". Then we insert a control point of type CONT_TYPE at a
-- given position "POSITION" and then converts the bit_vector back
-- to the integer type.
function Insert_Control_Point(
  SIG_VAR   :INTEGER;       -- The signal/variable to be modified
  SIZE      :INTEGER;       -- the number of bits required to encode SIG_VAR
  CONT_TYPE :CONTROL_POINT; -- The type of the control point (AND, OR)
  POSITION  :INTEGER;       -- The position of the bit in SIG_VAR to modify
  TEST_IN   :STD_LOGIC      -- Extra control input used for control insertion
  TM        :STD_LOGIC      -- TM = 1(0): Test Mode (Normal mode)
  ) return INTEGER is
  variable VAR: std_logic_vector(SIZE-1 downto 0);
  variable V   : std_logic;
  variable V_INT: integer;
begin
  VAR := conv_std_logic_vector(SIG_VAR,SIZE);
  if (CONT_TYPE = OR_POINT) then
    V := (TM and TEST_IN) or VAR(position);
    VAR(POSITION) := V;
  else
    V := (not TM or TEST_IN)  and VAR(POSITION);
    VAR(POSITION) := V;
  end if;
  V_INT := CONV_integer(VAR);
  return (V_INT);
end Insert_Control_Point;
```

```
-- A function to insert a control point at a signal/variable
-- SIG_VAR which is declared as BOOLEAN type.
function Insert_Control_Point(
  SIG_VAR    :BOOLEAN;       -- The signal/variable to be modified
  CONT_TYPE  :CONTROL_POINT; -- The type of the control point (AND, OR)
  TEST_IN    :BOOLEAN;       -- Extra control input used for control insertion
  TM         :BOOLEAN        -- TM = 1(0): Test Mode (Normal mode)
  ) return BOOLEAN is
 variable V    : BOOLEAN;
begin
 V := SIG_VAR;
 if (CONT_TYPE = OR_POINT) then
   V := (TN and TEST_IN) or V;
 else
   V := (not TM or TEST_IN) and V;
 end if;
 return (V);
end Insert_Control_Point;

-- A function to insert a control point at a signal/variable
-- SIG_VAR which is declared as STD_LOGIC type.
function Insert_Control_Point(
  SIG_VAR    :STD_LOGIC;     -- The signal/variable to be modified
  CONT_TYPE  :CONTROL_POINT; -- The type of the control point (AND, OR)
  TEST_IN    :STD_LOGIC;     -- Extra control input used for control insertion
  TM         :STD_LOGIC      -- TM = 1(0): Test Mode (Normal mode)
  ) return STD_LOGIC is
 variable V    : STD_LOGIC;
```

```
 begin
  V := SIG_VAR;
  if (CONT_TYPE = OR_POINT) then
   V := (TM and TEST_IN) or V;
  else
   V := (not TM or TEST_IN) and V;
  end if;
  return (V);
 end Insert_Control_Point;

 -- Function called from the function Insert_Control_Equal()
 -- which is used to insert a given number N of control points in
 -- the equality comparator.
 function unsigned_Insert_Control_Equal(
   L,R       :UNSIGNED;
   N         :INTEGER;
   P         :POSITION_ARRAY;
   CONT_TYPE  :Cont_TYPE_ARRAY;
   TEST_IN    :SIG_ARRAY;
   TM        :BOOLEAN
   ) return BOOLEAN is
variable V:BOOLEAN;
 begin
```

```
   V := (L(L'left downto (P(0)+1+L'right)) = R(R'left downto (P(0)+1+R'right)));
   For i in (N-1) downto 0 loop
if (i = N-1) then
   V := V and (Insert_Control_Point ((L((P(i) + L'right) downto L'right) =
       R((P(i) + R'right) downto R'right)),CONT_TYPE(i), TEST_IN(i),TM));
else
  V := V and (Insert_Control_Point ((L((P(i) + L'right) downto (P(i+1) + 1 +L'right)) =
    R((P(i) + R'right) downto (P(i+1) + R'right + 1))),CONT_TYPE(i), TEST_IN(i),TM))
end if;
end loop;
 return V;
 end unsigned_Insert_Control_Equal;

 -- A function to insert a given number N of control points inside
 -- the structure of the equality comparator.
 function Insert_Control_Equal(
   L,R      :STD_LOGIC_VECTOR; -- The two operands of the equality comparator
   N        :INTEGER;          -- The number of control points to insert
   P        :POSITION_ARRAY;   -- The positions of the control points
   CONT_TYPE :Cont_TYPE_ARRAY; -- The type of each control point
   TEST_IN   :SIG_ARRAY;       -- The extra control signals used for control insertion
   TM       :BOOLEAN           -- TM = 1(0): Test Mode (Normal mode)
   ) return BOOLEAN is
  constant length: INTEGER := R'length;
 begin
```

```
  return unsigned_Insert_Control_Equal
   (UNSIGNED(L), UNSIGNED(SXT(R,length)), N, P, CONT_TYPE, TEST_IN,TM);
end Insert_Control_Equal;

-- A procedure to insert an observation point at a given bit position of
-- a signal/variable SIG_VAR which is declared as an array of bits.
Procedure Insert_Observation_Point(
  SIG_VAR    :STD_LOGIC_VECTOR;  -- The signal/variable to observe
  POSITION   :INTEGER;           -- The bit position to observe
  signal SCAN :out STD_LOGIC     -- The signal in which the SIG_VAR is observed
  ) is
begin
 SCAN <= SIG_VAR(POSITION);
end Insert_Observation_Point;

-- A procedure to insert an observation point at a given bit position of
-- a signal/variable SIG_VAR which is declared as an integer. In this case
-- we need to convert the corresponding signal/variable into a number of bits.
-- This number is specified by the parameter "SIZE".
Procedure Insert_Observation_Point(
  SIG_VAR    :INTEGER;    -- The signal/variable to observe
  SIZE       :INTEGER;    -- The number of bits required to encode SIG_VAR
  POSITION   :INTEGER;    -- The bit position to observe in SIG_VAR
  signal SCAN :out STD_LOGIC -- The signal in which the SIG_VAR is observed
  )is
 VARIABLE V: std_logic_vector(SIZE-1 downto 0);
begin
```

```
 V := conv_std_logic_vector(SIG_VAR, SIZE);
 SCAN <= V(POSITION);
end Insert_Observation_Point;

-- A procedure to insert an observation point at a given bit position of
-- a signal/variable SIG_VAR which is declared as STD_LOGIC.
Procedure Insert_Observation_Point(
  SIG_VAR     :STD_LOGIC;     -- The signal/variable to observe
  signal SCAN  :out STD_LOGIC   -- The signal in which the SIG_VAR is observed
  ) is
begin
 SCAN <= SIG_VAR;
end Insert_Observation_Point;

-- A procedure to insert an observation point at a given bit position of
-- a signal/variable SIG_VAR which is declared as BOOLEAN
Procedure Insert_Observation_Point(
  SIG_VAR       :BOOLEAN;    -- The signal/variable to observe
  signal SCAN    :out BOOLEAN  -- The signal in which the SIG_VAR is observed
  ) is
begin
 SCAN <= SIG_VAR;
end Insert_Observation_Point;
```

```
-- A procedure to insert a given number of observation points inside
-- the structure of the equality comparator.
Procedure Insert_Observation_Equal(
  L, R       :STD_LOGIC_VECTOR; -- The two operand of the equality comparator
  variable V  :out BOOLEAN;     -- V := (L = R)
  N          :INTEGER;          -- The number of observation points to insert
  P          :Position_array;   -- The positions of the observation points
  signal SCAN  :out Sig_array   -- An array of signals used for observation
  ) is
 variable V0,V1:BOOLEAN;
begin
  V0 := (L(L'left downto (P(0)+1+L'right)) =   R(R'left downto (P(0)+1+R'right)));
For i in 0 to (N-1) loop
 if (i = N-1) then
   V1 := (L((P(i)+L'right) downto L'right) = R((P(i)+R'right) downto R'right));
   SCAN(i) <= V1;
   V0 := V0 and V1;
 else
   V1 := (L((P(i) + L'right) downto (P(i+1) + 1 +L'right)) =
                  R((P(i) + R'right) downto (P(i+1) + R'right + 1)));
  SCAN(i) <= V1;
  V0 := V0 and V1;
 end if;
end loop;
 V := V0;
 end Insert_Observation_Equal;
end Test_Points;
```