Department of Electrical and Computer Engineering

# Path Delay Fault Testing for Digital VLSI Circuits Using Specialized Binary Decision Diagrams

Kyriakos A. Christou

A Dissertation

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

at the University of Cyprus

May, 2012

# APPROVAL PAGE

Kyriakos A. Christou

**Path Delay Fault Testing for Digital VLSI Circuits Using Specialized Binary Decision Diagrams**

*The present Doctorate was submitted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy in the Department of Electrical and Computer Engineering, and was approved on May 03, 2012 by the members of the Examination Committee.*

Committee Chair _____
Dr. Theocharis Theocharides

Research Supervisor _____
Dr. Maria K. Michael

Committee Member _____
Prof. Dhiraj K. Pradhan

Committee Member _____
Dr. Chryssis Georgiou

Committee Member _____
Dr. Chrysostomos Nicopoulos

# Abstract

Increasing complexity and speed with billions of transistors on a single integrated circuit has let to circuits with incredible capabilities. Circuit advances in a rigorous evolving environment have let manufacturing testing into new test challenges. This thesis studies delay testing, that is detecting the circuit's timing violations and ensuring it's temporal correctness.

Specifically, this dissertation investigates the identification and test generation of single and multiple Path Delay Faults (PDFs) for enhanced fully scanned digital circuits as well as circuits with no scan capabilities. A crucial problem for the PDF model is the derivation of compact test sets. The PDF identification and test generation for various PDF classifications is investigated. The newly proposed test generation algorithms, utilize variants of special data structures known as Binary Decision Diagrams (BDDs) in a way that explicit enumeration of the PDFs is avoided. Paths and PDFs, are processed in an implicit and non-enumerative manner, which facilitates their usage in terms of dealing with a large number of circuit paths in reasonable time. For the single PDF case, experimental results on the enhanced full-scanned benchmarks, demonstrate clearly the practicality of the method in terms of test compactness for the critical PDF set. Furthermore, for the multiple PDF set, experimental results indicate that only a small number of critical multiple faults, compared to the number of single critical faults, needs and suffices to be examined. Therefore, only a small number of additional test patterns is needed to guarantee a circuit's timing specifications. For circuits with no scan capabilities such as microprocessors, experiments show that the methodology adopted allows reducing the test generation time, by concentrating on suitably classified structurally coherent fault lists and avoiding computation intensive gate level simulations.

Finally, an efficient way of identifying the pairwise physical paths correlation between the paths in a set is proposed. Beyond PDF testing, this metric has important implications in various design automation problems, such as timing analysis, test generation and diagnosis. When considering the complexity and tight timing constraints of modern circuits, this corre-

lation affects both the design process and the testing approaches followed in manufacturing.

# Περίληψη

Η αυξημένη πολυπλοκότητα στο σχεδιασμό, καθώς επίσης και οι υψηλές ταχύτητες λειτουργίας των σύγχρονων κυκλωμάτων, τα οποία περιέχουν δισεκατομμύρια ημιαγωγούς (transistors) σε ένα ενιαίο ολοκληρωμένο κύκλωμα, έχουν οδηγήσει στην παραγωγή κυκλωμάτων με απίστευτες ικανότητες. Η ραγδαία αυτή πρόοδος των κυκλωμάτων, μέσα σε ένα αυστηρώς εξελισσόμενο περιβάλλον, έχει οδηγήσει τη διαδικασία ελέγχου μετά την παραγωγή τους σε νέες προκλήσεις. Το αντικείμενο αυτής της διατριβής έχει ως στόχο τη μελέτη του ελέγχου της ορθής λειτουργίας χρονισμού κυκλωμάτων, συγκεκριμένα την ανίχνευση χρονικών παραβάσεων σε κυκλώματα μετά την παραγωγή τους, έτσι ώστε να διασφαλίζεται η χρονική τους ακρίβεια.

Συγκεκριμένα, αυτή η διατριβή διερευνά τον εντοπισμό (αναγνώριση) και την παραγωγή διανυσμάτων ελέγχου, τόσο για μονά, όσο και για πολλαπλά σφάλματα χρονικών καθυστερήσεων σε μονοπάτια (Path Delay Faults - PDFs), για κυκλώματα πλήρης σάρωσης αλλά και για κυκλώματα χωρίς δυνατότητα σάρωσης. Ένα πρόβλημα ζωτικής σημασίας για το PDF μοντέλο, που μελετάται στη διατριβή αυτή, είναι η παραγωγή συμπαγών διανυσμάτων ελέγχου σε ψηφιακά κυκλώματα πολύ μεγάλης κλίμακας ολοκλήρωσης (VLSI). Η αναγνώριση μονοπατιών και η παραγωγή διανυσμάτων ελέγχου για αυτά, διερευνάται για διάφορες ταξινομήσεις του PDF μοντέλου. Οι αλγόριθμοι για την παραγωγή διανυσμάτων ελέγχου που προτείνονται σε αυτήν την διατριβή, χρησιμοποιούν παραλλαγές εξειδικευμένων δομών δεδομένων, γνωστές ως δυαδικά διαγράμματα αποφάσεων (Binary Decision Diagrams - BDDs), με τέτοιο τρόπο κατά τον οποίο η απευθείας απαρίθμηση (explicit enumeration) των διαφόρων PDFs να αποφεύγεται. Τα υπό εξέταση μονοπάτια, και PDFs, διερευνήθηκαν με άμεσο τρόπο, χωρίς απαριθμήσεις, κάτι που διευκολύνει τη χρήση τους όσον αφορά την αντιμετώπιση μεγάλου αριθμού PDFs σε ένα κύκλωμα, μέσα σε λογικά χρονικά πλαίσια. Για την περίπτωση των μονών PDFs, τα πειραματικά αποτελέσματα που πάρθηκαν από κυκλώματα πλήρης σάρωσης, επιδεικνύουν την πρακτικότητα της μεθόδου όσον αφορά την πυκνότητα των διανυσμάτων ελέγχου για το σύνολο των κρίσιμων PDFs. Επιπλέων, για την περίπτωση των πολλαπλών PDFs, τα αποτελέσματα δείχνουν ότι μόνο ένας μικρός αριθμός πολλαπλών κρίσιμων PDFs, σε σύγκριση με τον αριθμό των μονών PDFs, χρειάζεται και επαρκεί να εξεταστεί. Ως εκ τούτου, μόνο ένας μικρός αριθμός

επιπρόσθετων διανυσμάτων ελέγχου χρειάζεται για να διασφαλιστούν οι χρονικές ενός κυκλώματος. Για κυκλώματα χωρίς δυνατότητες σάρωσης, όπως οι μικροεπεξεργαστές, τα πειράματα δείχνουν ότι η μεθοδολογία που υιοθετείται επιτρέπει τη μείωση του χρόνου παραγωγής των διανυσμάτων ελέγχου, με την επικέντρωση σε κατάλληλα ταξινομημένες δομικά συνεκτικές λίστες σφαλμάτων και την αποφυγή υπολογισμού εντατικών προσομοιώσεων σε επίπεδο πύλης.

Τέλος, προτείνεται ένας αποτελεσματικός τρόπος αναγνώρισης και εύρεσης συσχέτισης μεταξύ των φυσικών μονοπατιών ανά ζεύγος, σε ένα σύνολο μονοπατιών. Πέραν της εφαρμογής του στα προβλήματα αναγνώρισης και ελέγχου για PDFs, αυτό το μέτρο έχει σημαντικές επιπτώσεις σε διάφορα προβλήματα αυτοματοποίησης σχεδιασμού, όπως ο ακριβής καθορισμός της μέγιστης καθυστέρησης, ανίχνευση και διάγνωση σφαλμάτων σε ψηφιακά κυκλώματα. Εξετάζοντας την πολυπλοκότητα και τους στενούς χρονικούς περιορισμούς των σύγχρονων κυκλωμάτων, η συσχέτιση αυτή επηρεάζει τόσο την διαδικασία σχεδιασμού όσο και τις μεθοδολογίες ελέγχου που ακολουθούνται της παραγωγής.

# Acknowledgements

This thesis would not have been possible without the constant support and encouragement of my advisor Dr. Maria K. Michael, Assistant Professor in Electrical and Computer Engineering Department, University of Cyprus. Maria was always available for discussion, guidance, comments, criticism and interest in my work. A valuable colleague and an excellent teacher. For this I am grateful.

I would also like to thank my PhD examination committee members Professor Dhiraj K. Pradhan (University of Bristol, UK), Dr. Chryssis Georgiou (University of Cyprus, CS department), Dr. Theocharis Theocharides as well as Dr. Chrysostomos Nicopoulos (University of Cyprus, ECE department), for their helpful suggestions.

I would, also, like to thank Professor Mateo Sonza Reorda, Dr. Paolo Bernardi, Dr. Ernesto Sanchez and Dr. Michalangello Grosso (Politecnico Di TORINO, Italy) for their valuable collaboration.

Furthermore, during my research, I have had long and valuable collaboration with Dr. Stelios Neophytou. He has been an excellent collaborator and very good friend.

My parents, Andreas and Eleni, my brothers, Chistakis, Marios, Drosos, Nikolaos and friends have provided the support needed to complete a project of this scale. Especially I would like to thank my wife Stella and son Mario for their constant encouragement patience and love.

# Publications

### Published Book Chapters

1. Bernardi P., Christou K., Grosso M., Michael M. K., Sanchez E. and Reorda M. S., "Exploiting MOEA to Automatically Generate Test Programs for Path-Delay Faults in Microprocessors", in Springer Lecture Notes in Computer Science, Vol. 4974/2008, pp. 224-234.

### Published Journal Publications

1. Christou K., Michael M. K. and Tragoudas S.,"On the Use of ZBDDs for Implicit and Compact Critical Path Delay Fault Test Generation", *J. Electronic Testing*,Vol.24, pp.203-222, 2008.

### Published Peer-Reviewed Conference Proceedings

1. Neophytou S., Christou K. and Michael M. K.,"An Approach for Quantifying Path Correlation in Digital Circuits without any Path or Segment Enumeration", *Proc. of IEEE European Test Symposium*, pp.141-146, 2011.

2. Christou K., Michael M. K. and Neophytou S.,"Identification of critical primitive path delay faults without any path enumeration", *Proc. of IEEE VLSI Test Symposium*, pp.9-14, 2010.

3. Christou K., Michael M. K., Bernardi P., Grosso M., Sanchez E. and Reorda M. S., "A Novel SBST Generation Technique for Path-Delay Faults in Microprocessors Exploiting Gate- and RT-Level Descriptions",*Proc. of IEEE VLSI Test Symposium*, pp.389-394, 2008.

4. Christou K., Michael M. K. and Tragoudas S.,"Implicit Critical PDF Test Generation with Maximal Test Efficiency",*Proc. of IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, pp.50-58, 2006.

5. Michael M. K., Christou K. and Tragoudas S.,"Towards finding path delay fault tests with high test efficiency using ZBDDs",*Proc. of IEEE International Conference on Computer Design*, pp.464-467, 2005.

**Journal Publications Under Review**

1. Christou K. and Michael M. K.,"On the use of Decision Diagrams to Identify Critical Primitive Path Delay Faults", *submitted*, May 2012.

2. Neophytou S., Christou K. and Michael M. K.and Tragoudas S.,"A Non-Enumerative Technique for Measuring Path Correlation in Digital Circuits", *submitted*, November 2011.

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# Chapter 1

# Introduction

In this Chapter the thesis motivation and prior work is presented followed by the thesis statement, its major contributions and organization.

## 1.1 Motivation and Prior Work

Increasing complexity and speed (device size, power domains, high-speed pins, probe cards, interface boards) of modern digital circuits conduce in violations of the performance specifications, inducing the circuit's quality echelon. With billions of transistors on an integrated circuit, the nanometer era, and the integration technology advances have let to circuits having incredible capabilities [120]. The trend is to put as many components of a system in an integrated circuit by taking advantage of the large integration potentials, since manufacturing yield is getting smaller. In the 2011 edition of the International Technology Roadmap for Semiconductors (ITRS) [120], is reported that despite all the effort in reducing the test cost, test cost is becoming an increasing concern for the test community. A survey was contacted in ITRS 2011, asking what is the community's biggest concern and 40% of the respondents have said that their biggest concern is the cost of test, compared to 30% from the ITRS 2009. Furthermore, in the question if the cost of test is expected to become their biggest concern, 85% responded positively. Thus, test cost is proved to be of increasing concern. Moore's law [36] should be maintained by having high quality test procedures and at the same time the testing cost should be kept low. Vertically Stacked die with Through Silicon Vias (TSV), called 3D/TSV, has set new challenges to the test community that concerns testing multiple interconnected devices which will include multiple vendors and technologies. With TSVs being as small as 2 $\mu m^2$ per connection and using 3D interconnects, different dies can be

stacked together vertically. Thus a test engineer must be concern of not only for individual die testing but also for the partial/entire die stack. The ITRS 2011 reveals that all of the above imply great testing challenges and methodologies in a rigorous evolving environment.

Technology scaling imposes tight timing constraints that should be accommodated, without affecting the quality of the integrated circuit. This thesis concentrates on *delay* testing. Detecting timing defects and assuring the circuit's specifications is the primary objective of delay testing. Delay testing has been a hot research topic for more than two decades now. Increasing design speed, advances in VLSI technology and strict timing requirements of modern digital circuits is an increasing driving force that leads delay testing in another era beyond testing small delay defects [1, 33, 63, 94, 116] and faults caused by statistical process variations [5,65]. Extensive research to the delay testing problem resulted in the introduction of several new testing methodologies/schemes and delay fault models.

Five major delay fault models exist: line, gate, transition, path and segment delay fault models [16]. The first three delay fault models represent delay defects lumped at gate while path and segment delay fault models represent defects that are lumped at a gate or distributed along the examined path or segment. The Path Delay Fault (PDF) model [57, 88, 106] addresses tests for paths. A path can be defined as a sequence of connected gates from the primary inputs to the primary outputs. This delay fault model concerns the cumulative delay that may occur along the path. This implies that small delay defects may be detected through this model since it models distributed delay faults. Thus, the main advantage of the PDF model is that it is the most accurate delay fault model. The main drawback of the PDF model is that it is much more complex than the other three models mentioned. For every physical paths of a circuit there exist two PDFs, namely the rising or slow to rise ($R$,$\uparrow$) and the falling or slow to fall ($F$,$\downarrow$) $PDFs$. The number of considered faults can be exponential with respect the lines of the examined circuit, even when restring to the critical set, [85], [102], [114], [37], [27], [70], thus the test size increases which implies longer test application time. Furthermore, test generation and fault simulation are computationally expensive compared to the transition, gate or line delay fault model. A single *PDF* can be classified as robustly testable, non-robustly testable and functionally sensitizable. All the PDFs that do not meet the sensitization criteria mentioned are categorized as functionally unsensitizable or as redundant faults.

Primitive PDFs have appeared in [51, 52, 56, 64, 105, 113]. It has been proven that the primitive PDF set is fault-free if and only if the circuit examined is strongly delay-verifiable. This immediately implies that if the primitive PDF set is identified and tested, then this

would guarantee the circuits performance. Identifying the primitive *PDF* set is a complex and difficult task since it deals with multiple PDFs and the number of multiple PDFs is exponential with the lines of the examined circuit in the worst case.

The testing methodology used relies on the circuit type examined, combinational or sequential. Delay testing for combinational circuits or for the enhanced full-scanned sequential circuit versions, [31], involves the application of a test vector pair at the circuits primary inputs, $V = \{v_1, v_2\}$. This is a necessary step for creating and propagating transitions in the circuit under investigation. The first vector, $v_1$, initializes the circuit under test to the desired values while the second vector, $v_2$, propagates the desired transition to the primary outputs of the circuit. The values observed at the primary outputs are then compared to the already known output values of a faulty-free circuit. Any arbitrary vector pairs can be applied since the two vectors $v_1$ and $v_2$ are uncorrelated. Chapter 3 and Chapter 4 consider the enhanced full-scanned circuit version.

When considering sequential circuits, with limited scan capability, or even in the absence of scan capability, delay testing becomes harder. According the Huffman model [73], a sequential's circuit operation can break into *time frames*. Each time frame represents the combinational logic of the circuit and the application of the vector pair involves values not only for the primary inputs, but for the state lines also (current and next state lines). A number of testing strategies for sequential circuits with scan capabilities exist such as enhanced scan, Launch On Capture (LOC- [97], [2]) and Launch On Shift (LOS - [98]). For the LOS delay test application, also known as skewed-load delay test, vector $v_1$ is arbitrary while vector $v_2$ is derived by a 1-bit shift of $v_1$. For the LOC test application methodology also called broadside or double capture or even functional transition method, $v_1$ is an arbitrary vector while vector $v_2$ is derived from $v_1$ through the circuit function. Circuits with no scan capability, such as the microprocessor circuits in Chapter 5. If a microprocessor design is tested using its functional vectors, i.e., using instruction sequences [58], [18], [59], the input signals to the embedded block of the processor are derived by their functionality. Evolutionary algorithms [28], have been successfully exploited for the automatic generation of program sets for verification, test [29], and diagnosis [95] for processors described at different levels of abstraction. Depending on the testing methodology used, especially for sequential circuits, area, test application time and fault coverage vary.

Binary Decision Diagrams (*BDDs*), have been used long time now for the representation of choice [3, 60]. The work of Bryant [14], which focuses on boolean functions and the Boole-Shannons expansion, reveals the advantages of *BDDs* with their two main

properties.Zero-suppressed *BDDs* (*ZBDDs*) are *BDD* variants where the absence of a variable implies a zero value for that variable and not a don't care value as in a *BDD* would.

BDDs have been proposed for test generation under the stack-at fault model, [103]. As the number of stack-at faults in a circuit is linear to the number of lines in the circuit, thus structural techniques with a test per fault approach tend to behave better than BDD-based approaches in terms of time and memory. This dissertation exploits the PDF fault model where the number of faults considered are exponential with respect to the number of lines of the circuit in the worst case. Thus, a test per fault approach is not acceptable. A tool that can capture in an efficient and compact manner this huge fault space utilizes ZBDDs [84]. Hence, this thesis takes advantage of such a tool and incorporates it appropriately with BDDs in order to identify and test PDFs in an efficient manner, avoiding PDF enumeration at any stage.

## 1.2 Thesis Organization and Major Contributions

This thesis concentrates on *delay* testing and examines permanent errors caused by manufacturing defects. The thesis investigates the *complete identification* and *efficient test generation* of single and multiple Path Delay Faults (PDFs) for enhanced fully-scanned digital circuits as well as circuits without any scan capabilities. The PDF identification, for both the single and multiple case, is addressed in an implicit (non-enumerative) manner. Moreover, the generation of test sets that can test a large number of PDFs in reasonable time is examined. Furthermore, this thesis proposes an efficient way to identify the pairwise *physical paths* correlation between paths in a set. This metric has important implications in various design automation problems, such as timing analysis, test generation and diagnosis. The results presented in Chapter 3 are derived from the work appearing in papers [25], [24], [26]. The results of Chapter 4 appeared in paper [23] while a journal version is currently under submission. The results of Chapter 5 have appeared in [8] and [22], while the results of Chapter 6 appeared in paper [81] while a journal version is currently under submission.

Chapter 2 of the thesis gives basic notation and preliminaries. The first main problem addressed in this work, Chapter 3, considers the PDF classification of [21], where faults can be tested robustly, non-robustly, or with functional sensitization. The proposed framework includes function-based formulations, with appropriate representations and test generation algorithms. In particular, it investigates the use of Zero-Suppressed Binary Decision Diagrams (ZBDDs) [77] in compact and efficient PDF test generation, concentrating on critical

PDFs. In order to derive a compact test set, tests with high test efficiency must be generated. *Test Efficiency (TE)* is defined here as the *number of new PDFs detected by a test*. For each sensitization type, a Boolean function is formulated whose solution is the set of all targeted sensitizable PDFs along with their corresponding set of sensitization cubes (PDF tests). The function produced is represented by a ZBDD-based canonical data structure. Both, circuit paths and PDF tests are captured in a non-enumerative fashion.

The first challenge is to show how the tests and the corresponding sensitized PDFs can be mapped by a single Boolean function and represented appropriately and efficiently. For completeness and clarity, it is first shown how such a function can be constructed for all the PDFs in a circuit, not just the critical ones. In this case, the problem of exact identification of sensitizable PDFs (also, that of unsensitizable PDFs) is solved non-enumeratively. This is a major problem in PDF ATPG, as demonstrated in [21, 38, 62, 72, 85, 101, 109], among many others. Consequently, it is shown how the proposed data structure can be constructed for any subset of PDFs, such as the potentially critical PDFs. In this case, the proposed PDF-sensitization function is restricted such that only sensitizable PDFs from a targeted set of PDFs are included.

The motivation is to investigate how the generated data structure, containing all the targeted sensitizable PDFs and their corresponding tests, can be efficiently manipulated to derive a compact test set. A challenging task here involves exploiting the properties of the structure to generate a test with high *TE*, a compact test, without enumerating any of the paths or tests represented by the structure. A *major contribution* of this work involves deriving a test with maximal *TE* in linear time to the size of the data structure, using appropriate graph traversals. Additional tests with high *TE* are derived by removing already detected PDFs from the data structure and repeating the method to derive another test with maximal *TE*. Thus, each generated test guarantees to detect a large number of PDFs that have not been already detected. The experimental results clearly demonstrate the practicality of the method and its superiority over existing methods in terms of high test efficiency for critical PDFs. The *main contributions* of this work are given below:

- all (critical) singly sensitizable PDFs of a circuit are incorporated in a newly proposed specialized variant of BDDs called ISOP/ZBDD, in an implicit and compact manner,
- *function formulations* are given for three sensitization types namely robust, non-robust and functionally sensitization,
- find a methodology ($\mathscr{T}$-Graph generation) for deriving a test with *maximal Test Ef-*

*ficiency(TE)*, in linear time to the size of the data structure, using appropriate graph traversals is proposed,

- *additional tests with high TE* are derived by removing already detected PDFs from the data structure and repeating the methodology proposed,
- experimental results clearly demonstrate the practicality of the method and its superiority over existing methods in terms of *high test efficiency for critical* PDFs.

Chapter 4 considers the problem of finding the testable critical primitive *PDF* set in combinational or enhanced fully-scanned sequential circuits. Even though the problem of identifying testable critical *PDFs* has been extensively addressed in the literature, see [25, 64–66, 85, 114] among many others, none of these methods considers primitive *PDFs* explicitly. This work is the *first* to define and identify critical primitive *PDFs*, integrating both aspects of criticality and primitivity in a common framework. A major challenge in such a problem is the large number of paths needed to be considered in order to identify primitive faults, even when the problem is restricted to a small number of critical faults. It is important to note that the work here is independent from the critical *PDF* selection phase; any critical *PDF* set selection algorithm and delay model can be applied to derive the set of single critical *PDFs* which is used as input to our methodology.

The proposed methodology utilizes function-based formulations with appropriate data structures (Zero-Suppressed Binary Decision Diagrams - ZBDDs) for implicit and compact representation of paths such that the targeted set of faults is identified in a non-enumerative manner (no path, path-segment, or fault is ever enumerated). ZBDDs have been previously proposed for the simpler version of this problem for traditional (critical) *PDFs* [25, 85] however, none of these methods can be trivially extended to primitive faults since the standard ZBDD operators they utilize cannot handle multiple faults. This work presents new operators, polynomial to the size of the ZBDD, for efficient and non-enumerative manipulation of multiple faults which are necessary for identifying all critical primitive faults. The *major contributions* of this work are:

- testable critical primitive *PDFs* are defined,
- the targeted faults are identified efficiently, avoiding enumeration of faults or paths which can be prohibitive for large circuits,
- any delay model for identifying the potentially testable critical (single) *PDF* set can be considered,

- the generated data structure that represents the targeted faults also contains test generation data (all tests per identified testable fault) and, hence, the proposed method can be easily incorporated in a very efficient test generation framework, since the necessary tests are already generated (test generation is not the main focus of this work, however, some indicative test generation results are reported),

- the reported experimental results show that only a small number of multiple primitive $PDFs$ is testable (when compared to the set of single primitive $PDFs$), implying that a small number of additional tests suffices to guarantee the circuit's timing correctness under the multiple fault criterion.

Chapter 5, exploits the usage of PDFs when considering sequential circuits with no scan capability. Specifically, this Chapter presents an *innovative* approach for the generation of functional programs to test path delay faults within microprocessors. The proposed method takes advantage of both the gate- and RTlevel description of the processor. The former is used to build Binary Decision Diagrams for deriving fault excitation conditions; the latter is exploited for the automatic generation of test programs able to excite and propagate fault effects, based on an evolutionary algorithm and fast RTL simulation. Experimental results show that this methodology allows reducing the test generation time, by concentrating on suitably classified structurally coherent fault lists and avoiding computation-intensive gate-level simulations. The employed evolutionary algorithm takes advantage of the introduced BDD-based fitness evaluation functions for directing the test programs generation flow towards optimal solutions. The obtained coverage results are comparable to manual/deterministic approaches in literature. The *major contributions* of the third main problem addressed in this thesis are given below:

- build *BDDs for deriving fault excitation conditions* and compute a *fitness value* based on the gate(logic) level description of the microprocessor core,

- *drop un-testable* faults using the BDD-based fitness evaluation function,

- use of a newly introduced *BDD-based fitness evaluation function* for better (faster) directing the test programs generation flow towards a solution

- experimental results show that this methodology allows *reducing the test generation time*, by concentrating on suitably classified structurally coherent fault lists and avoiding computation-intensive gate-level simulations.

Chapter 6 exploits an efficient way of identifying the pairwise correlation between the paths of a given set, for example the physical paths of a digital circuit. The correlation between the physical paths of a digital circuit has important implications in various design automation problems, such as timing analysis, test generation and diagnosis. When considering the complexity and tight timing constraints of modern circuits, this correlation affects both the design process and the testing approaches followed in manufacturing. In this work the diversity of a set of paths (or path segments) is quantified, let these be critical I/O paths, error propagation paths for various fault models, or paths traced for diagnostic purposes. Circuit paths are encoded using Zero-Suppressed Binary Decision Diagrams (ZBDDs). The proposed method consists of a sequence of standard ZBDD operations to provide a measure of the overlap of the paths under consideration, that is, a comprehensive statistical characterization for a given path set. The *main contribution* of the presented method is that path or path segment enumeration is entirely avoided and, hence, a large number of paths can be considered in practical time. The proposed ZBDD method has a polynomial, to the size of the diagram, complexity. The effectiveness of the proposed measure can be seen though experimentation using two different approaches. The first one shows how the proposed technique identifies similarities among various I/O critical path sets and can distinguish their characteristics based on just two measure values per test. The second one uses the proposed methodology to compare the propagation paths between multiple-detect and $n$-detect test sets in relation to their corresponding defect coverage. The main contributions in this Chapter are listed below:

- a new methodology based on standard ZBDD operations that gives a comprehensive *statistical characterization* for a given path set of *polynomial complexity* and avoids enumeration and, hence, a large number of paths can be considered in practical time,
- the effectiveness of the methodology is demonstrated via two different experimentations:
  - the first one shows how the proposed technique identifies similarities among various I/O critical path sets and can distinguish their characteristics based on just two measure values per test,
  - the second one uses the proposed methodology to compare the propagation paths between multiple-detect and n-detect test sets in relation to their corresponding defect coverage.

Chapter 7 summarizes this thesis and discusses future research directions.

# Chapter 2

# Preliminaries

In this Chapter necessary background is provided. First, a summary of delay testing and fault models is given. Then a general introduction on decision diagrams is presented and in the last section of this chapter the different test application methodologies are described.

## 2.1 Delay Testing and Fault Models

The primary objective of delay testing is to assure the circuit's specifications by detecting timing defects. Strict timing requirements are being imposed by the advances in the modern digital circuits speed, complexity and size. Delay testing for combinational circuits, involves the application of a test vector pair at the circuits primary inputs, $V = \{v_1, v_2\}$, to first create and then propagate transitions at the circuit's primary outputs, where the values are compared with the responses of a non-faulty circuit. If the circuit under test is a sequential circuit then according to the Huffman model [73], a sequential's circuit operation can break into time frames, where each time frame corresponds to the combinational logic of the circuit. Several testing strategies for sequential circuits exist namely enhanced scan, standard scan, at speed and slow-fast-slow. Area, test application time and fault coverage vary depending on the testing methodology used.

As mentioned in the introduction, five major delay fault models are considered namely line,gate,transition,path and segment delay fault models. The first three delay fault models represent delay defects lumped at gate while path and segment delay fault models represent defects that are lumped at a gate or distributed along the examined path. It is assumed that there exists a nominal delay for each gate for a transition to reach it's output pin, from the gates input pins, and a nominal delay interconnect delay for the transition to move from an

output pin to an input pin [16].

The transition delay fault model [20] assumes that the delay defect of the gate examined, slow to rise or slow to fall, is large enough so that all the transitions from the specific gate will be delayed. This implies that small or distributed delay defects along a circuit's path may not be captured. Nowadays, small delay defects [1, 33, 63, 94, 116], become a big issue since moving to nanometer designs there exist more small delay defects than large delay defects. Moreover smaller cycle times imply more sensitivity to small delay defects. On the other hand this model is practical since its fault list and coverage metric are similar to the stuck-at-fault model. It can also detect defects missed by the stuck-at-fault model like opens, shorts and coupling defects. The main difference of the gate delay fault model [44, 93] with the transition delay fault model is that it takes timing into account. It is assumed that all long paths going through the defective gate might affect the performance. The gate delay fault model does not assume that a large delay on a gate will defenetly affect the performance of the circuit. It takes into account the propagation path through the defective gate. In order for a test for the gate delay fault model to be able to detect a defect, a threshold for determining the smallest delay fault size has to be specified. The limitations of the gate delay fault model is similar to the limitations of the transition delay fault model. The line delay fault model [67] is a variation of the gate delay fault model that assumes that a fault occurring at a specific line, propagates through the longest sensitizable path that includes that line. Thus the line delay fault model may detect faults distributed along certain paths. The main advantage of these three delay fault models is that they have a linear number of faults with respect to the number of gates in the circuit.

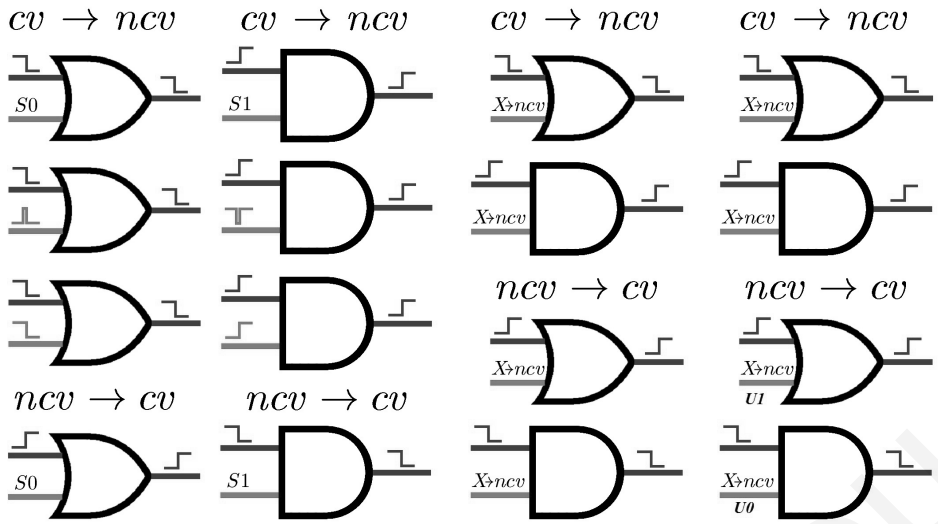
The Path Delay Fault (**PDF**) model [57, 88, 106], addresses tests for paths. A path can be defined as a sequence of connected gates from the primary inputs to the primary outputs. This delay fault model concerns the cumulative delay that may occur along the path. This implies that small delay defects may be detected through this model since it models distributed delays on circuit's paths. Thus the main advantage of the PDF model, is that it is the most accurate delay fault model [16]. The disadvantages of the PDF model is that it is much more complex than the other three models mentioned. The number of considered faults can be exponential thus the test size increases which implies longer test application time. Furthermore, test generation and fault simulation is computationally more expensive compared to the transition, gate or line delay fault model. Another disadvantage of the PDF model is that there exist a lot of path delay faults that are redundant, PDF faults that do not affect the circuits performance. The segment delay fault model [39, 40] is a variation of the

PDF model along with the transition fault model. It breaks the path into segments. If the path breaks to the number of the gates of the path is the same as the transition fault model and if the path starts at a primary input and ends on a primary output is the same as the path delay fault model. Thus the test size maybe linear to exponential and the test generation process depends on the segments length. Moreover, if we consider an untestable path, a part of it maybe testable. Based on this observation, a lot of research has been carried out that determines the segment length value.

For every physical paths of a circuit there exist two Path Delay Faults (PDFs), namely the rising or slow to rise ($R$,$\uparrow$) and the falling or slow to fall ($F$,$\downarrow$) PDFs. A PDF can be classified as robustly testable, non-robustly testable and functionally sensitizable. All the PDFs that do not meet the sensitization criteria mentioned are categorized as functionally unsensitizable or as redundant faults. A lot of research work has been done in identifying untestable PDFs, that do not affect the circuits performance and thus they do not need to be tested [38], [21], [62], [101]. It has been proven that a lot of redundant faults can be identified through logic implications. In order to detect a PDF fault, a rising or falling transition, must be propagated along the path under consideration. A gate along that path is called an on-input gate while all the other are called site-input gates. Similarly a multiple input gate that is also an on-input gate, has a single on-input fan-in and all the others are called site-input or off-input fan-ins. Furthermore a controlling value ($cv$) for a gate is defined to be the value that determines the output of the gate. For example, the value 0(1) for the $AND$($OR$) gate is a controlling value while 1(0) is a non-controlling value ($ncv$). Moreover, $X$ denotes a "don't care" value. The different sensitization conditions/criteria are described below.

A PDF is ***robustly testable*** [61, 106] if it guarantees to detect the delay fault on the targeted path independent of all other delays in the circuit. If we have a transition on the path examined $cv \rightarrow ncv$, all the site inputs on a gate that belongs to the examined path will have to be either steady at the $ncv$ of the gate, or finally to take a $ncv$ or even to have the same transition as the on-input. If the transition is from a $ncv$ to a $cv$, $ncv \rightarrow cv$, then all the site input should settle to a non-controlling value. All the conditions just described, for the $AND$ and $OR$ logic gates, can be seen in Fig. 2.1(a).

The set of ***non-robustly testable*** PDFs [61] is a superset of the robustly testable PDFs. Non-robust PDFs are obtained by relaxing the sensitization conditions at the off-input fan-ins of the gates along the PDF examined. In the non-robustly testable set of PDFs conditions are reposed so that even in the $ncv \rightarrow cv$ case the only requirement is that the side inputs in the second vector are settled to their $ncv$, on vector $v_2$. The sensitization conditions for




(a) Robust (b) Non-Robust (c) Functional

Figure 2.1: PDF Test Conditions for the Robust, Non-Robust and Functionally Testable.

the non-robust sensitization case can be seen in Fig. 2.1(b). The sensitization conditions are identical with the static sensitization conditions of [7], justifying the name statically sensitizable PDFs. This implies that if we can obtain the delays of the circuit, we can have better non-robust tests (there exist a large number of non-robust tests for a PDF) if we could make sure that the transitions on the site-inputs of the PDF examined have less eventuality to mask the on-input transitions. A non-robust test is guaranteed to be valid if no other path has a delay fault.

***Functionally sensitizable*** PDFs [51, 52] include both robustly and non-robustly testable PDFs as a superset. The sensitization conditions for the functionally sensitizable conditions can be seen in Fig. 2.1(c). Examining the $cv \rightarrow ncv$ case on the on-input fan-in of a gate along the PDF examined, the requirement is that the site inputs final settle to their *ncv*. The $ncv \rightarrow cv$ allows the site-inputs not only to settle to the finally *ncv* but also to settle to their *cv* with hazards. Take for example an *OR* gate. The second case, $ncv \rightarrow cv$, is to have a rising transition on the on-input fan-in. Thus not only $X \rightarrow ncv$ ($ncv = 0$ for the *OR* gate), a final logic value of 0 is allowed, but it is also allowed to have a final logic 1 value with hazards. This means that the site inputs can have any value except the staple at *cv*, in our example this implies that the off input on the *OR* gate can not be *S*0, stable at 0. This expressed with $U_{cv} = X_{cv} - S_{cv}$, $U0 = X0 - S0$ for the *AND* gate and $U1 = X1 - S1$ for the *OR* gate. This observation implies that the functionally sensitizable PDF set includes PDFs that have to be multiply tested. In the case of the robustly and non-robustly testable PDFs the considered PDF set includes only singly testable delay faults. Indirectly this suggests that functionally

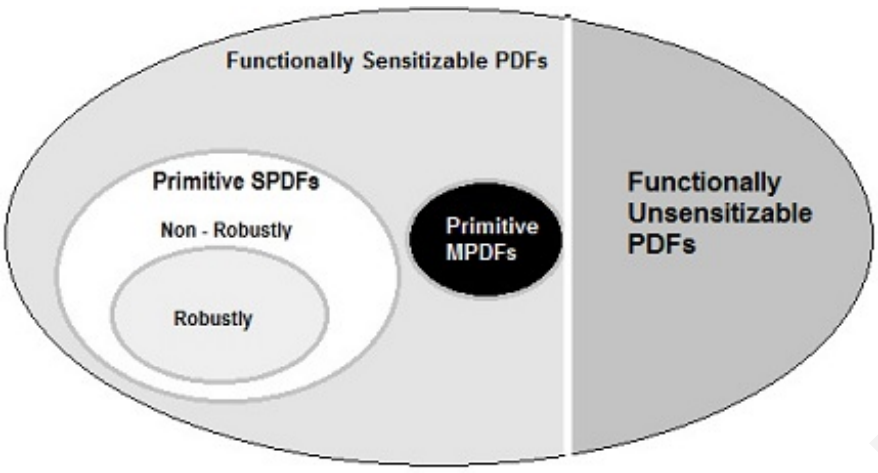



Figure 2.2: Classification of the PDF Set.

sensitizable PDF set is a much more bigger set.

As we have previously mentioned, the ***functionally unsensitizable*** or ***redundant*** PDFs [21, 34, 108] are the ones that do not affect the circuits performance and thus they do not needed to be tested. This set includes PDFs that their on-input transitions are always masked by the site input transitions either by settling to the stable at $cv$ on the site input or finally settling at the controlling value (if the transition is from $cv \rightarrow ncv$). If this PDF set is identified and removed, it becomes easier to deal with the rest of the PDFs since the number of the PDFs is in the worst case exponential. Furthermore, the need for lowering the PDF number considered has directed researchers in finding a smaller PDF set to consider in order to guarantee the circuits performance, the primitive PDF set.

Primitive PDFs have appear for example, in [51, 52, 56, 64, 105, 113]. It has been proved that the primitive PDF set is faulty free if and only if the circuit examined is strongly delay-verifiable. This immediately implied that if we could identify and test the primitive PDF set we could guarantee the circuits performance. Identifying the primitive PDF set is a complex task as we shall see later in Chapter 4.

Fig. 2.2 categorizes the set of all PDFs. The set of PDFs can be partitioned into two disjoin subsets, the PDFs that can be sensitized under specific sensitization conditions and the PDFs that can not be sensitized. The later subset is exactly the functionally un-sensitizable PDFs. The first subset of PDFs can be further categorized based on the sensitization type of the faults. Again two possible sensitization type groups can be identified, namely the group that can be statically sensitized and the the group that can not be statically sensitized. In the first group the singly sensitized PDFs is identified with two possible single sensitization

types, namely the robustly sensitizable and the non-robustly sensitizable PDFs. Based on the PDF classification of [21] the set of non-robustly sensitized PDFs include the set of robustly testable PDFs. The group that can not be statically sensitized is exactly the set of functionally sensitizable PDFs.

The primitive PDF set includes PDFs from the functionally sensitizable PDFs and the statically sensitized PDFs. The primitive PDF set can be partitioned into two big sub-groups, namely the singly sensitized PDF group denoted by *SPDF* and the multiply sensitized group *MPDF*. The primitive *SPDF* set is exactly the singly non-robustly sensitized PDF set. The *MPDF* is a fraction of the functionally sensitizable PDF set.

## 2.2 Decision Diagrams

Binary Decision Diagrams (BDDs) have been used for the representation of choice [3, 60]. The work of Bryant [14], which focuses on boolean functions and the Boole-Shannons expansion, reveals the advantages of BDDs with their *two* remarkable properties, that have influenced the world of *testing, equivalence checking, circuit optimization* and many *more*. The first property is about the remarkable power that BDDs have in effectively representing very large combinatorial sets. The second property refers to the canonicity of the BDDs. Under certain conditions BDDs are a canonical representation of functions. A form is canonical if and only if the representation of a function in that form is unique. For example 2 circuits are equivalent if their corresponding BDDs are identical. This immediately implies that equivalence checking is easy.

A BDD is a single rooted Directed Acyclic Graph (DAG) with 2 different node types. The first node type called terminal nodes has 2 elements, namely terminal node **1** and terminal node **0**. All the other nodes, internal nodes, have 2 outgoing edges (out-degree of each node is 2). The first outgoing edge namely the **T** (THEN or 1) edge and the **E** (ELSE or 0) edge. A route or path in the BDD that ends up on terminal **1** is a solution of the function that the BDD represents. An ***ordered*** BDD implies that all paths starting from the root node to the terminal nodes appear in the same order. A ***reduced*** BDD implies that all redundant nodes have been removed, nodes that their 1 and 0 edges end up on the same node, and that isomorphic subgraphs have been identified and merged. This implies that absence of a node (variable) in the BDD entails a don't care value of that variable.

Some important properties of BDDs are discussed below. The number of internal nodes of a BDD can be exponential in the worst case. Moreover the complexity of the logical

*AND* and *OR* operations of two BDDs is polynomial in terms of the size of the operants. Complementation, satisfiability and tautology can be solved in constant time. On the other hand BDDs sizes rely on the variable ordering used. Finding a good variable ordering is hard and has also been a critical research issue. Two BDDs that represent the same function with different variable ordering may differ in the size of the produced BDD, in terms of number of nodes used. Moreover a bad ordered BDD produces a difficult to read and understand DAG in contrast with a good ordered BDD. Furthermore, in some cases the Sum Of Products (SOP)/Product Of Sums (POS) function representation form, are more compact and more close in the final circuit implementation.

Other decision diagrams have been proposed by simply applying different reduction rules. Zero-suppressed BDDs (*ZBDDs*) is a BDD variant where absence of a variable implies a zero value for that variable and not a "don't care" value as the BDD would. Thus in cases where you deal with functions that have a lot of zero values this decision diagram performs best due to the applied reduction rule. *ZBDDs* efficiently represent problems expressed in set theory and they perform better in the cover representation problem. Moreover in sparse sets, where a lot of elements have a zero value, *ZBDDs* represent these sets more compactly than BDDs.

### 2.2.1 ZBDD Representing the PDFs of a Circuit

Applications for the representation of sparse combinatorial sets with *ZBDDs* have been used in [84], to efficiently and compactly represent the path delay faults of a given circuit, even for path-intensive circuits. *ZBDDs* can also store Boolean functions by introducing some additional variables but without increasing the number of paths in the *ZBDD* [77]. Namely, two variables are introduced for each variable in the function, and an appropriate encoding protocol is activated so that when both variables are suppressed, the original variable is a don't care. A *ZBDD* where such pairs of variables are used for each original variable is a *ZBDD*-based representation of an Ir-redundant Sum-of-Products (*ISOPs*) [77, 79, 96]. Some necessary definitions and an example of how PDFs can be represented using *ZBDDs* follow.

*Path* variables are defined, which encode the PDFs. Let $\mathscr{P}$ define the set of path variables, *PI* the set of primary inputs, and *L* the set of circuit lines other than primary inputs. There is one path variable per line in *L*, and two path variables per line in *PI* to represent the rising and falling transitions on the primary inputs. Hence $|\mathscr{P}| = |L| + 2 \times |PI|$. Consider the circuit in Fig. 2.3 which has 7 internal lines $\{d, e, f, g, h, i, j\}$ and 3 primary inputs $\{a, b, c\}$.

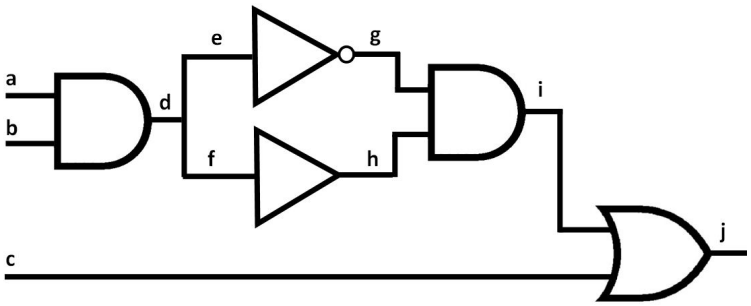



Figure 2.3: Example circuit $C_1$

There are 13 path variables denoted by $\mathscr{P} = \{aR, aF, bR, bF, cR, cF, d, e, f, g, h, i, j\}$, where $iR$ ($iF$) is the rising (falling) transition variable for primary input $i$. A PDF is encoded by a combination over the variables of $\mathscr{P}$. For example, the rising PDF on path $a-d-e-g-i-j$ is represented by $aR \cdot d \cdot e \cdot g \cdot i \cdot j$. Missing path variables assume a 0 value. The *ZBDD* representation of all the PDFs for the circuit $C_1$ in Fig. 2.3, is shown in Fig. 2.4. The variable ordering follows the topological order of the lines in the circuit. There are exactly 10 paths from the root node $aR$ to the terminal-1 node, one for each PDF in the circuit.

## 2.3 Delay Test Application Methodologies

The testing methodology used relies on the circuit type examined, combinational or sequential. Regarding test generation addressing path-delay faults, several techniques exist for enhanced full-scan circuits, based on either structural ATPG tools [32], [112] or function based tools using Binary Decision Diagrams (BDDs) [10], [74], [13] and Boolean-SAT [19], [117] implementations. Delay testing for combinational circuits or for the enhanced full-scanned sequential circuit versions [31], involves the application of a test vector pair at the circuits primary inputs, $V = \{v_1, v_2\}$. This is a necessary step for creating and propagating transitions in the circuit under investigation. The first vector, $v_1$, initializes the circuit under test to the desired values while the second vector, $v_2$, propagates the desired transition to the primary outputs of the circuit. The values observed at the primary outputs are then compared to the already known output values of a faulty-free circuit. Any arbitrary vector pairs can be applied since the two vectors $v_1$ and $v_2$ are uncorrelated. Hence for sequential circuit's $v_1$ and $v_2$ do not need to be functional vectors. Fig. 2.5 shows the application of the vector pair to the circuit. An extra hold latch is used per each scan flip-flop. The first pattern can be scanned in and applied to the functional logic and the second pattern can then be scanned

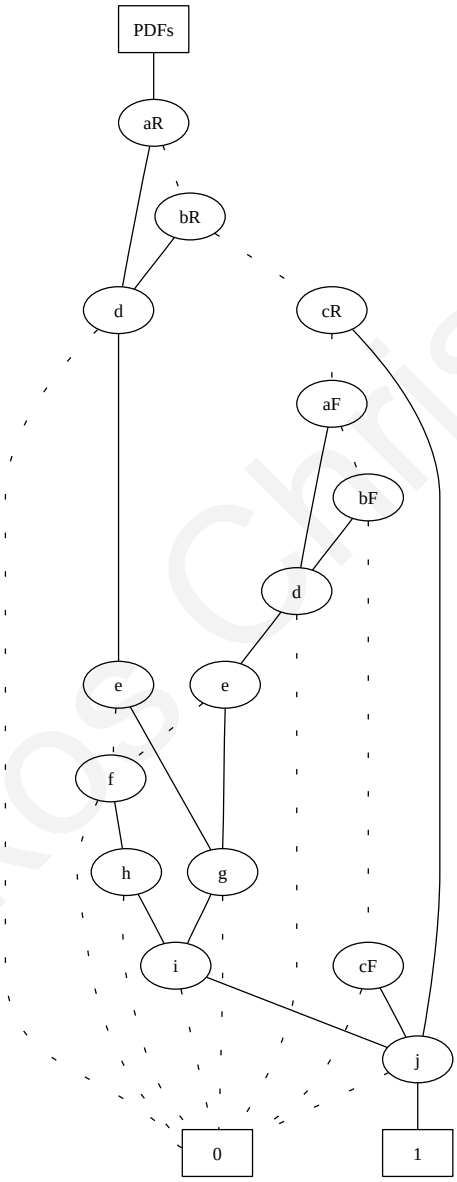



Figure 2.4: ZBDD with all PDFs of circuit $C_1$ of Fig. 2.3.

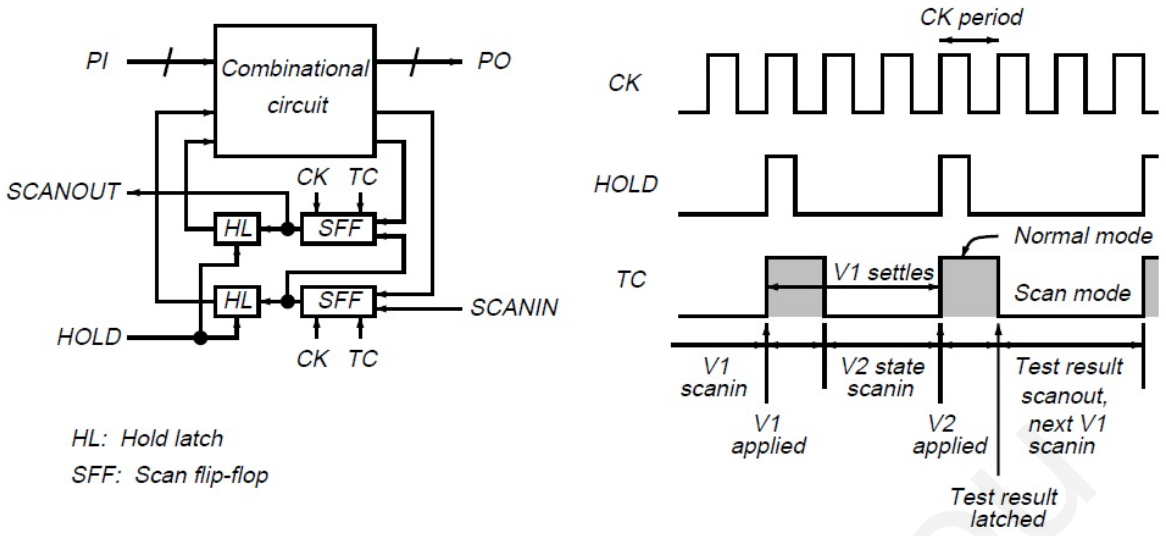



Figure 2.5: Enhanced-Scan delay test application to sequential circuit (Adopted from [16])

in, while the first pattern is still being applied to the functional logic. The second pattern can then be applied to the functional logic.

Two main approaches are distinguished in the case where sequential circuits have limited scan capability namely Launch On Capture (LOC- [97], [2]) and Launch On Shift (LOS - [98]). For the LOS delay test application, also known as skewed-load delay test (launch-on-last-shift), vector $v_1$ is arbitrary while vector $v_2$ is derived by a 1-bit shift of $v_1$. The first pattern, $v_1$, is scanned in and the second pattern, $v_1$, is obtained from $v_1$, with the application of a single scan clock cycle. Thus $v_2$ vector is restricted since it must be a shifted version of $v_1$. If the scan path architecture is based on flip-flops, the scan shift is significantly lower speed compared to the system clock. The scan enable signal should be able to switch values very quickly and as soon as $V_2$ is at the flip-flop outputs, the response of the circuit must be captured within a time equal to the system clock cycle time. The LOS waveform can be seen in Fig. 2.6. For the LOC test application methodology also called broadside or double capture or even functional transition method, $v_1$ is an arbitrary vector while vector $v_2$ is derived from $v_1$ through the circuit function. The first pattern, $v_1$, is scanned through the scan path. Pattern $v_2$ is then loaded from the functional logic outputs. Any arbitrary $v_1$ can be used but $v_2$ is restricted based on the functional logic. The waveform of LOC can be seen in Fig. 2.7.

Circuits such as a microprocessor (used in Chapter 5) can not have the luxury of scan capability because this would imply a huge overhead. Delay fault testing in circuits with no scan capabilities require more than two vectors and two methodologies can be applied

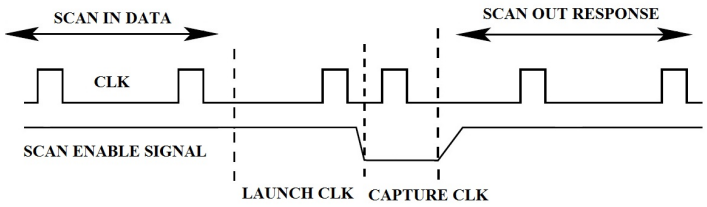



Figure 2.6: Launch On Shift Test methodology

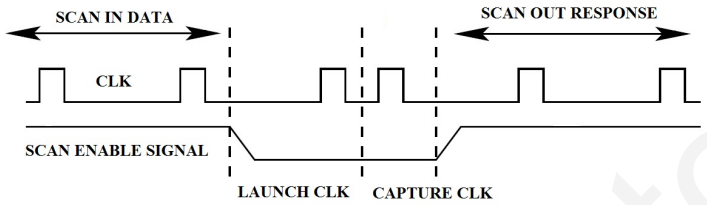



Figure 2.7: Launch On Capture Test methodology

[12], [68]. The basic idea is that the second vector $v_2$ should be justified by $v_1$ through the combinational function while $v_1$ will be generated by a set of vectors starting at some initial state (justification sequence). The state should be propagated at some primary outputs if the path destination is a flip flop. A functional test is a test that can be applied at speed, at the normal circuit operation. A path is functionally testable if there exist a functional test for that path. Software-based test generation tries to produce such test sequences, i.e; instruction sequences in the case of a microprocessor. Some work on software-based test generation has been done exploiting deterministic techniques [104], [59], [35], [58], [18]. If a microprocessor design is tested using it's functional vectors, i.e using instruction sequences [58], [18], [59] the input signals to the embedded block of the processor are derived by their functionality. Testing a microprocessor requires that a test program should be made to ensure that in a consecutive pair of clock cycles the excitement conditions of the targeted path delay fault is met and that the delay effects are propagated to the processor's output ports.

# Chapter 3

# ZBDDs for Implicit and Compact Path Delay Fault Test Generation

## 3.1 Introduction

The Path Delay Fault (PDF) model is one of the most popular models used for delay testing, since it is the most accurate model that can detect both lumped and distributed delay defects. For practical reasons, PDF testing usually considers functionally sensitizable PDFs [10], [11], [21], [38], [48], [62], [72], [74], [85], [90], [91], [99], [101], [102], [109], [114], [115] even though it has been shown in [105], that a subset of the multiple PDFs, called the primitive PDFs, needs and suffices to be tested. Furthermore, due to the large number of faults, which is exponential to the circuit size in the worst case, often only the critical paths are considered [85], [102], [114]. A critical path is one with large delay. Traditionally, the delay of a path has been calculated based on discrete-valued models, such as the fixed and bounded delay models. More recently, statistical models are also proposed for the selection of critical paths [114]. Even when the problem is restricted to critical PDF testing, its complexity remains prohibitive, mainly because:

1. the number of critical PDFs to be considered can still be very large and
2. often, only a small number of the critical PDFs are sensitizable.

For the above two reasons, the problem of deriving compact test sets for critical PDFs in an implicit manner is crucial, especially for path intensive circuits. The test generation problem remains a critical and a hot research topic for more that three decades now [92], [89].

This work considers the PDF classification of [21], and gives function-based formulations with appropriate representations and test generation algorithms, using ZBDDs [78], in a non-enumerative, compact and efficient PDF test generation. *Test Efficiency (TE)* is defined here as the *number of new (critical) PDFs detected by a test*. For completeness and clarity, it is first shown how such a function can be constructed for all the PDFs in a circuit, not just the critical ones. In this case, the problem of exact identification of sensitizable PDFs (also, that of unsensitizable PDFs) is solved non-enumeratively. This is a major problem in PDF ATPG, as demonstrated in [21], [38], [62], [72], [85], [101], [109], among many others. Consequently, it is shown how the proposed data structure can be constructed for any subset of PDFs, such as the potentially critical PDFs. In this case, the proposed PDF-sensitization function is restricted such that only sensitizable PDFs from a targeted set of PDFs are included.

The motivation is to investigate how the generated data structure, containing all the targeted sensitizable PDFs and their corresponding tests, can be efficiently manipulated to derive a compact test set. A challenging task here involves exploiting the properties of the structure to generate a test with high *TE*, without enumerating any of the paths or tests represented by the structure. A major contribution of this work involves deriving a test with maximal *TE* in linear time to the size of the data structure, using appropriate graph traversals. Additional tests with high *TE* are derived by removing already detected PDFs from the data structure and repeating the method to derive another test with maximal *TE*. Thus, each generated test guarantees to detect a large number of PDFs that have not been already detected.

The recently proposed function-based method of [85] stores all sensitizable PDFs in a circuit using ZBDDs, but functions are maintained only for each sensitizable segment of the PDFs, as BDDs [13] (a segment is defined between any consecutive fan-out stems). This method examines pairs of path segments that are on common unsensitizable paths, similarly to the structural techniques of [38], [62] and [101] [1], which identify either such pairs of segments or lines. The advantage of [85] over these structural techniques is on the representation of the PDFs (or segments), such that it gains in computation time. However, the test efficiency of this method is very low, almost a test per fault, since the data structures it uses cannot assist in compact test pattern generation. Low test efficiency also exists with the

[1] These methods estimated a lower bound on the number of unsensitizable PDFs, which can be used to indicate the completion of the ATPG process more accurately, however, they do not provide any specific guidance to the ATPG on path or test selection.

structural method of [109], which identifies testable and untestable PDFs using an implication graph to generate test sets with very high fault coverage.

Methods that explicitly target the generation of compact test sets for PDFs have been proposed in [115], [11], [47], [90], [99]. The test compaction procedures of [11], [47], [90], use the concept of primary and secondary target faults. Once a test is found for a primary fault, it is expanded to detect one or more secondary faults. The level of compaction in these techniques depends greatly on the selection order of the primary and secondary faults. [99] finds maximal sets of potentially compatible faults. Even though they may not target all faults explicitly, the above methods remain enumerative since they are based on the principle of first targeting a single fault and then attempting to find one or more faults that can be tested mutually with the original fault. The methods of [115] and [47], which focus on critical paths, report considerably higher test efficiencies that the other enumerative methods, however, they both remain restrictive due to their path enumerative nature.

Non-enumerative ATPG methods, such as [48], [74], [91], were proposed to overcome the problem of path enumeration. [48], [91] are structural-based methods relying on graph theoretic arguments. Still, each of their generated test detects a very small number of PDFs. The recent function-based (BDD-based) non-enumerative tool of [74] outperforms both [48], [91] in terms of *TE*. Although this method offers scalability (*TE* does not drop much as coverage increases), it is not complete since it may not be able to achieve 100% coverage since it generates a test that sensitizes a number of PDFs simultaneously, however, the PDFs detected may not always be new. Furthermore, [74] cannot handle unsensitizable paths implicitly (i.e., cannot avoid targeting such paths). More importantly, none of the existing non-enumerative methods can handle critical PDF test generation in an absolute non-enumerative manner.

The rest of the chapter is organized as follows. Section 4.2 gives necessary background and notation. Section 3.3 shows the proposed function formulation along with its representation can be derived, under each of the considered sensitization types. Section 3.4 extends the framework to consider only a subset of the PDFs, such as the potentially critical PDFs. The compact ATPG process is discussed in Section 3.5. Section 3.6 reports and discusses the obtained experimental results and Section 3.7 concludes the chapter.

| Expression | Description |
|---|---|
| $!(S,v)$ | Complement $v$ for all combinations in set $S$ |
| $S \cup Q$ | Set union |
| $S \cap Q$ | Set intersection |
| $S \backslash Q$ | Set difference |
| $\exists(S,v)$ | Existentially abstract variable $v$ from set $S$ |
| $\|S\|$ | Number of combinations in set $S$ |
| $\subset (S,v)$ | Subset of set $S$ such that $v = 1$ |

Table 3.1: Standard ZBDD Operators used

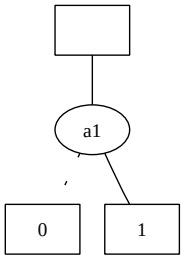
## 3.2 Preliminaries and Notation

Finding sensitizable PDFs requires considering the sensitization conditions at the path off-inputs, which can be derived by examining the functionality of the circuit's lines. Thus, sensitization conditions can be represented using functions, referred to as *test functions*. The work in [10], has shown BDD-based test functions per single PDF and [74] derived BDD-based test functions for sets of PDFs. A test function contains all tests that detect one (or a set) of PDFs. However, it does not contain any information on which PDF(s) is detected. Here, the goal is to find the sensitizable PDFs along with their tests and store them compactly in a common structure. The PDFs are represented using ZBDDs, for efficiency and compactness, as demonstrated in [84]. To incorporated the tests in the same data structure with the PDFs, we generate ZBDD-based ISOPs to represent the sensitization conditions (instead of BDDs, as in [74] and [10]. The resulting common structure is called an ISOPs/ZBDD because of the interpretation as far as variable appearances are concerned. Some variables participate in pairs and thus are interpreted according to the ZBDD-based ISOPs encoding whereas the remaining variables are interpreted as usual. Each node in this structure obeys standard ZBDD decomposition and reduction rules. Table 3.1 lists all the standard ZBDD operations used by the proposed methodology.

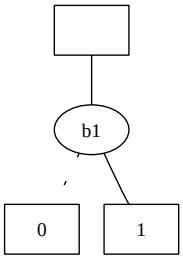
Next, we present basic notation used in this chapter and some more details on ZBDD representations of PDFs and derivation of ZBDD-based ISOPs from standard BDDs.

The proposed function is expressed over two sets of variables, the *test* variables which encode the test cubes and the *path* variables which encode the PDFs. Let $\mathscr{P}$ define the set of path variables, *PI* the set of primary inputs, and *L* the set of circuit lines other than primary inputs. There is one path variable per line in *L*, and two path variables per line in *PI* to represent the rising and falling transitions on the primary inputs. Hence $|\mathscr{P}| = |L| + 2 \times |PI|$. The reader id refereed to read Chapter 2, where a detailed example that demonstrates the

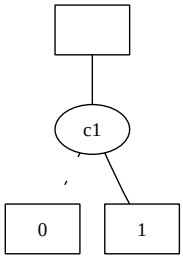
encoding used is presented.



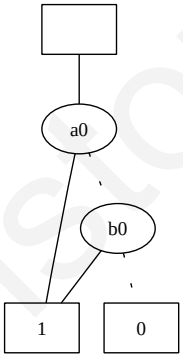

(a) $f_a^1() = a1\bar{a}0 = a$.




(b) $f_b^1() = b1\bar{b}0 = b$.




(c) $f_c^1() = c1\bar{c}0 = c$.




(d) $f_g^1() = \bar{a}1a0 + \bar{b}1b0 = \bar{a} + \bar{b}$.

Figure 3.1: Line functionalities using ISOP-based ZBDDs.

Let $\mathscr{T}^1 \subset \mathscr{T}$ be the set of test variables corresponding to vector $v^1$, i.e., $\{i_0^1, i_1^1\} \in \mathscr{T}^1$, $\forall i \in PI$. Similarly we define $\mathscr{T}^2 \subset \mathscr{T}$ for vector $v^2$. $f_l^1(\mathscr{T}^1)$ and $f_l^2(\mathscr{T}^2)$ are the functions realized at a circuit line $l$ expressed with respect to variables in $\mathscr{T}^1$ and $\mathscr{T}^2$, respectively. These functions are derived by generating the appropriate BDD per line and then using the procedure of [77] to derive their ZBDD-based ISOPs representation. Fig. 3.1 shows the line functionalities using ISOPs-based ZBDDs for various lines of circuit $C_1$ of Fig. 2.3, with respect to variables in $\mathscr{T}^1$ for the first vector $v^1$. For clarity, the variable notation used in Fig. 3.1 is a bit simplified from the one presented in the previous paragraph. Thus, $a1(b1,c1)$ is actually $a_1^1(b_1^1,c_1^1)$ and $a0(b0)$ is $a_0^1(b_0^1)$, i.e., the superscript 1 is implied in these figures (all the figures in this chapter follow this labelling, unless otherwise stated). Observe the ISOPs-based ZBDD of Fig. 3.1. Variable $a1$ appears in its normal form and the missing variable $a0$ is implied to appear in a complemented form. Thus, $f_a^1(\mathscr{T}^1) = a1 = a1\overline{a0} = a$ (where $a$ is the original variable in the BDD). A function $f_l^2(\mathscr{T}^2)$, for some line $l$, is identical to $f_l^1(\mathscr{T}^1)$ when every variable in $\mathscr{T}^2$ is substituted by its corresponding variable in $\mathscr{T}^1$. It is necessary to define different line functions (and hence, input variables) over the two time frames of $(v^1, v^2)$, since an input can assume different values over the two time frames of a

delay test.

We also define stability functions $S_l^0(\mathscr{T})$ and $S_l^1(\mathscr{T})$ per line $l$, to contain all solutions of $(v^1, v^2)$ that bring a stable-at-0 and stable-at-1 value at $l$, respectively. Stability functions were introduced in [10] for single-input change tests (also used in [74]) and later generalized in [54] for multi-input change tests. The later method is used to derive multi-input stability functions for all circuit lines in BDD format which are then converted to ZBDD-based ISOPs.

## 3.3 ISOP/ZBDD Graph for Sensitizable PDFs

This Section discusses the desired function formulation, using a polynomial number, to the circuit size, of standard ZBDD operations. Both circuit paths and PDF tests are captured implicitly (no path or segment or test enumeration is performed), which gives a strictly non-enumerative solution to the problem of exact identification of (un)sensitizable PDFs. The focus of this Section is on path sensitization, how to capture the various sensitization conditions [21] in a single function, along with the sensitized paths. Given correct sensitization conditions, restricting appropriately to only critical paths is consequently considered in Section 3.4.

Assume a circuit line $l$ with set of immediate predecessor lines denoted by $FI(l)$. Functions $G_l^R(\mathscr{T} \cup \mathscr{P})$ and $G_l^F(\mathscr{T} \cup \mathscr{P})$ denote the desired sensitization functions at some line $l$. We show how functions $G_l^R()$ and $G_l^F()$ are formulated at $l$, based on functions $G_i^R()$ and $G_i^F()$, $i \in FI(l)$. Let $ncv_g, cv_g \in \{0,1\}$ denote the non-controlling and controlling value of gate $g$, respectively.

### 3.3.1 Non-Robust PDF Sensitization

Under the non-robust sensitization criterion, path off-inputs at some gate $g$ must be set to $x \rightarrow ncv_g$ under $(v^1, v^2)$, irrespective of the type of transition propagated on the path on-input. This implies that only input values for $v^2$ need to be examined (values in $v^1$ are implied based on the values of $v^2$). Hence, only test variables in $\mathscr{T}^2 \subset \mathscr{T}$ are used.

Let $i$ be a primary input. We start by defining functions $G_i^R()$ and $G_i^F()$ at the primary inputs in $PI$. Each $i \in PI$ is associated with two path variables, $iR$ and $iF$, and two test variables, $i_0^2$ and $i_1^2$. The functionality of $i$, based on the ZBDD-based ISOPs representation, is given by $f_i^2() = \overline{i_0^2} \cdot i_1^2$ and its complement is $\overline{f_i^2()} = i_0^2 \cdot \overline{i_1^2}$. (The complementation operation is not the standard Boolean one. It is based on ZBDD-based ISOPs encoding.) To represent

a transition on $i$, the appropriate path variable ($iR$ or $iF$) and corresponding value at $i$ in $v^2$ (value 0 for falling and 1 for rising transition) need to be included. The necessary function formulations at a primary input $i$ are given below:

$$G_i^R\left(\mathscr{T}^2 \cup \mathscr{P}\right) \;=\; !\left(f_i^2(\mathscr{T}^2),\; iR\right),\;\; i \in PIs \tag{3.1}$$

$$G_i^F\left(\mathscr{T}^2 \cup \mathscr{P}\right) \;=\; !\left(\overline{f_i^2(\mathscr{T}^2)},\; iF\right),\;\; i \in PIs \tag{3.2}$$

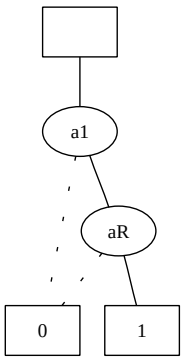
The change operator (!) appends the appropriate variable to each cube in the set, and gives $G_i^R() = \overline{i_0^2} \cdot i_1^2 \cdot iR$ and $G_i^F() = i_0^2 \cdot \overline{i_1^2} \cdot iF$. Observe how the desired information is encoded in these functions. For example, $G_i^R = \overline{i_0^2}.i_1^2.iR$ gives the rising segments up to line $i$ (which is just line $i$ in this case encoded as $iR$ in the cube) that is non-robustly sensitized by setting $i = 1$ ($\overline{i_0^2}.i_1^2 = i$) in $v^2$. The value of $i$ is implied in $v^1$ to the 0 value, since $iR$ encodes a rising transition. Consider the circuit $C_1$ of Fig. 2.3. The ISOPs/ZBDD graphs for the sensitization functions for some of $C_1$'s input lines are given in Fig. 3.2. For example, in Fig. 3.2(b) $G_a^F() = a0.aF$. Variable $a1$ has a 0 value and, therefore, is suppressed. Here, the falling segment up to a line $a$ ($aF$) is sensitized by setting $a$ to 0 in $v^2$ ($\overline{a1}.a0 = \overline{a}$). For $v^1$, $a = 1$ is implied by the existence of variable $aF$ in the cube.

Consider now an AND gate $g$ with output line $l$ and fanins $FI(l)$. PDF segments can be sensitized up to line $l$ through any of the lines in $FI(l)$. Thus, any line in $FI(l)$ can be considered as an on-input with the remaining lines being off-inputs that must settle to $ncv_g$. Consider line $y \in FI(l)$ to be an on-input. Function $\bigcap_{x \in FI(l), x \neq y} f_x^2()$ will give all test cubes that allow all remaining off-inputs in $FI(l)$ to settle to $ncv_g = 1$ (given by $f_x^2()$ in the expression). Let $tr \in \{R, F\}$ denote a Rising or a Falling transition. Function $!((G_y^{tr}() \cap (\bigcap_{x \in FI(l), x \neq y} f_x^2())),\; l)$ will give all non-robustly sensitizable PDFs up to line $l$, through line $y$, along with the sensitization cubes. Every line in *FI(l)* is a possible on-input, thus function $G_l^{tr}()$ given by:

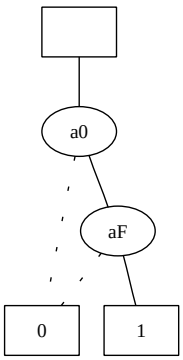
$$G_l^{tr}() \;=\; !\left(\left(\bigcup_{y \in FI(l)} \left(\, G_y^{tr}() \cap \left(\bigcap_{x \in FI(l), x \neq y} f_x^2()\right)\right)\right),\; l\right) \tag{3.3}$$

For an OR gate $g$ the off-inputs in $FI(l)$ need to settle to $ncv_g = 0$. Thus, it suffices to replace $f_x^2()$ with $\overline{f_x^2()}$ in Eq. 3.3, as given below:

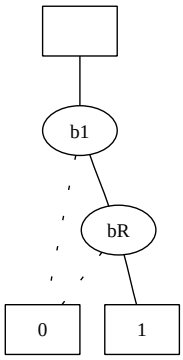
$$G_l^{tr}() \;=\; !\left(\left(\bigcup_{y \in FI(l)} \left(\, G_y^{tr}() \cap \left(\bigcap_{x \in FI(l), x \neq y} \overline{f_x^2()}\right)\right)\right),\; l\right) \tag{3.4}$$



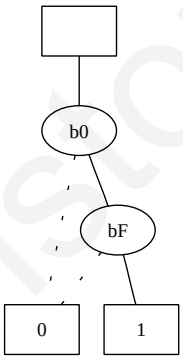

(a) $G_a^R() = a1.aR.$




(b) $G_a^F() = a0.aF.$




(c) $G_b^R() = b1.bR.$




(d) $G_b^F() = b0.bF.$

Figure 3.2: Non-Robust Sensitization functions for primary inputs of circuit $C_1$ of Fig. 2.3.

The $G_l^R()$ and $G_l^F()$ functions for the output line $l$ of a NOT gate with input line $y$ are given below.

$$G_l^R() \;=\; !\left(G_y^F(),\, l\right) \tag{3.5}$$

$$G_l^F() \;=\; !\left(G_y^R(),\, l\right) \tag{3.6}$$

An illustrating example is presented next. Consider the circuit of Fig. 2.3, which has 5 physical paths and, thus, 10 PDFs. There are 3 primary inputs and a total of 10 lines in the circuit. As already mentioned in Section 4.2, there are 13 path variables denoted by $\mathscr{P}$. Since non-robust sensitization is considered, $\mathscr{T} = \mathscr{T}^2$. For clarity, variable $i_0^2(i_1^2)$ is denoted by $i0(i1)$ in the figures. In this example, there are 6 test variables in $\mathscr{T}$ ={*a1, a0, b1, b0, c1, c0*}. Fig. 3.3 shows the falling(Fig. 3.3(a)) and rising(Fig. 3.3(b)) sensitizable PDF segments up to the internal line $g$ of the circuit. All sensitizable segments up to $g$, that arrive at $g$ with a falling transition, are shown in Fig. 3.3(a). Only two such segments exists, $aR-d-e-g$ and $bR-d-e-g$. The ISOPs/ZBDD graph in Fig. 3.3(a) contains 2 paths from the root node to the terminal-1 node, $\{a1.b1.aR.d.e.g, a1.b1.bR.d.e.g\}$, one per existing

sensitized segment. Furthermore, the graph contains all sensitization cubes for each of these two segments, given by $a1b1 = a1\overline{a0}b1\overline{b0} = ab = 11$ for $v^2$. For $v^1$, $a = 0$ for segment $aR.d.e.g$ and $b = 0$ for $bR.d.e.g$ (implied by the $aR$ and $bR$ variables, respectively). Thus, $(v^1, v^2) = (a^1b^1c^1, a^2b^2c^2)$ = (0$xx$,11$x$) is the complete set of tests for $aR.d.e.g$. Similarly, $(v^1, v^2) =$ ($x$0$x$, 11$x$) for $bR.d.e.g$.

The ISOPs/ZBDD for the sensitization functions for complete PDFs, derived only after the output line $j$ is processed, are given in Fig. 3.4. All sensitizable PDFs, arriving at the output line $j$ with a falling transition, are contained in the graph of Fig. 3.4(a) (3.4(b)). Five (one) such PDFs exists for $C_1$ and exactly 5 (1) paths from the root node of $G_j^F()$ ($G_j^R()$) to the terminal-1 node exist. The circuit contains 4 non-robustly unsensitizable PDFs ($aF.d.e.g.i.j$, $bF.d.e.g.i.j$, $aR.d.f.h.i.j$, $bR.d.f.h.i.j$). All of these PDFs arrive with a rising transition to one of the input lines of the AND gate with output line $i$. The fact that they are unsensitizable is recognized when forming $G_i^F()$, which equals to the constant function 0 implying that no PDF segment can be sensitized with a rising transition on line $i$. This occurs since none of the necessary sensitization conditions imposed by the function formulation is satisfied and, implicitly, all PDF segments with a rising transition at line $i$ are dropped (removed) from the corresponding ISOPs/ZBDD. In the case where only a subset of segments is identified as unsensitizable, the function formulation ensures that only the unsensitizable segments are dropped, as it happens in $G_j^R()$ of Fig. 3.4(b).

### 3.3.2 Other PDF Sensitization Types

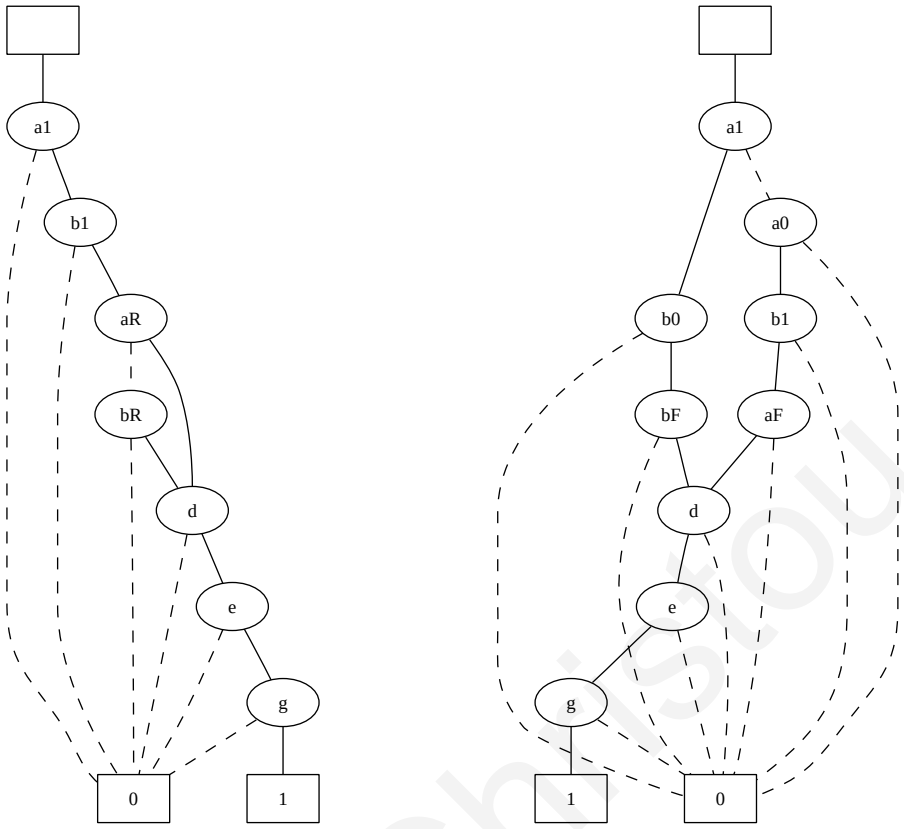
For robust and functional sensitization, certain input values in both $v^1$ and $v^2$ vectors need to be explicitly fixed [21]. Thus, the set of $\mathscr{T} = \mathscr{T}^1 \cup \mathscr{T}^2$ test variables is used. For a primary input $i$, the $G_i^R()$ and $G_i^F()$ functions for either robust or functionally sensitizable PDFs are given below:

$$G_i^R(\mathscr{T} \cup \mathscr{P}) \;=\; !\left(\overline{f_i^1()} \cap f_i^2(),\, iR\right),\;\; i \in PIs \tag{3.7}$$

$$G_i^F(\mathscr{T} \cup \mathscr{P}) \;=\; !\left(f_i^1() \cap \overline{f_i^2()},\, iF\right),\;\; i \in PIs \tag{3.8}$$

In this case, the sensitization cubes for a rising transition on input $i$ are given in $\overline{f_i^1}.f_i^2 = i_0^1.\overline{i_1^1}.i_0^2.\overline{i_1^2} = \overline{i^1}.i^2$, which denotes that $i = 0$ in $v^1$ and $i = 1$ in $v^2$. Similarly, for a falling transition on $i$, $f_i^1.\overline{f_i^2}$ gives $i = 1$ in $v^1$ and $i = 0$ in $v^2$.

When a $cv_g \rightarrow ncv_g$ transition is propagated on a PDF on-input, the off-input sensitization criteria are identical for all types of sensitization [21]. Thus, Eq. 3.3 and 3.4 also hold for




(a) $G_g^F()$. (b) $G_g^R()$.

Figure 3.3: Sensitization functions for line $g$ of circuit $C_1$ of Fig. 2.3.

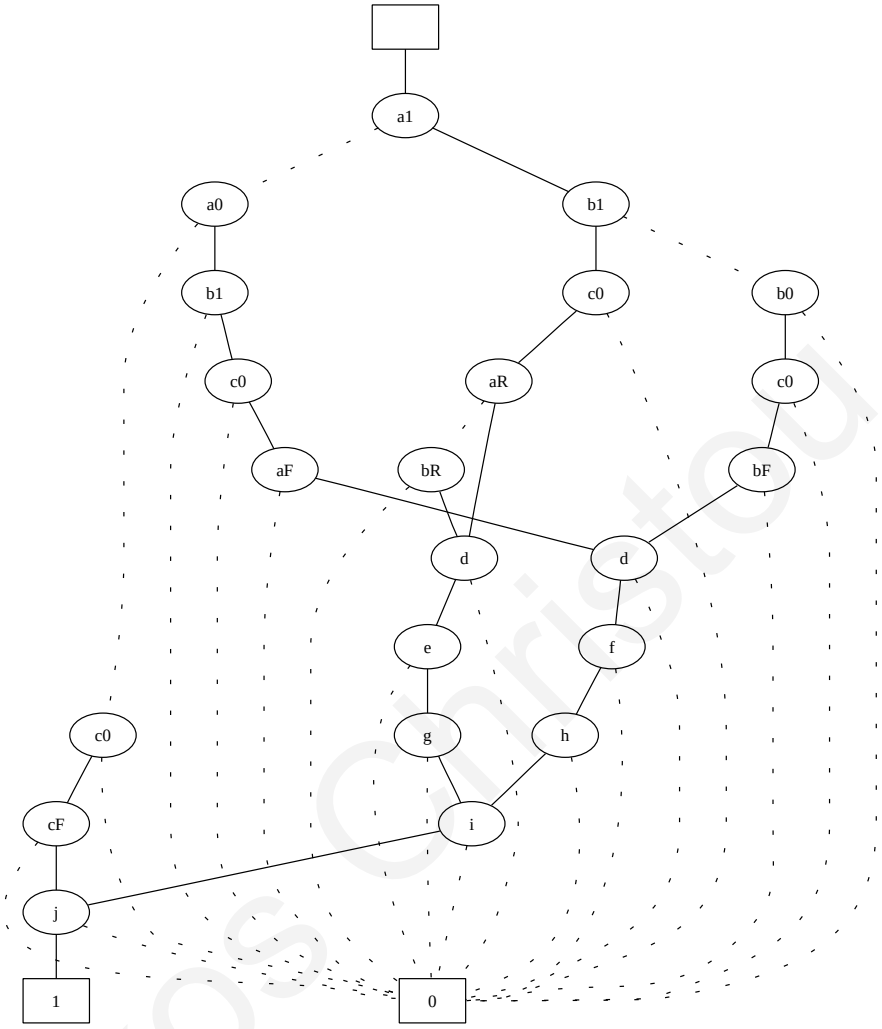
robust and functional sensitization. When the propagating on-input transition is $ncv_g \rightarrow cv_g$, robust sensitization requires all off-inputs to be stable at the non-controlling value, while functional sensitization allows any value other than stable at the controlling value at the off-inputs. Stability functions are used in order to express the necessary off-input constraints. For robust sensitization, function $G_l^{tr}()$ for any non-inverting gate $g$ with output line $l$ under $tr = ncv_g \rightarrow cv_g$, is given by:

$$G_l^{tr}() \;=\; !\left(\left(\bigcup_{y\in FI(l)}\left(\, G_y^{tr}() \cap \left(\bigcap_{x\in FI(l),x\neq y} S_x^{ncv_g}()\right)\right)\right), \, l\right) \tag{3.9}$$

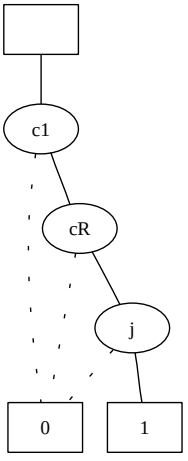
In Eq. (3.9), $S_x^{ncv_g}()$ denotes the stable at non-controlling value function at line $x$, which is the $S_x^0()$ function for an OR gate or the $S_x^1()$ function for an AND gate. Similarly to Eq. (3.9), function $G_l^{tr}()$ for functional sensitization is defined below, for $tr = ncv_g \rightarrow cv_g$:

$$G_l^{tr}() \;=\; !\left(\left(\bigcup_{y\in FI(l)}\left(\, G_y^{tr}() \cap \left(\bigcap_{x\in FI(l),x\neq y} \overline{S_x^{cv_g}}()\right)\right)\right), \, l\right) \tag{3.10}$$

Function $S_x^{cv_g}()$ denotes the stable at controlling value function at line $x$, which is the $S_x^1()$




(a) Sensitization function for falling line $j$, $G_j^F()$.




(b) Sensitization function for rising line $j$, $G_j^R()$.

Figure 3.4: Sensitization functions for line $j$ of circuit $C_1$ of Fig. 2.3.

function if $g$ is an OR gate or the $S_x^0()$ function if $g$ is an AND gate.

### 3.3.3 ISOPs/ZBDD for All Sensitizable PDFs

**Data**: Circuit $C$, Sensitization Type $ST$

**Result**: ISOPs/ZBDD for $G_{circuit}()$

% Gate Types (GT) = {BUFF, NOT, AND, NAND, OR, NOR};

% Sensitization Types (ST) = {R, NR, FS};

Declare set of test variables $\mathscr{T} = \mathscr{T}^1 \cup \mathscr{T}^2$;

$\mathscr{T}_i^2 = \{i^2 \mid \forall i \in PI\}$;

**if** *ST == (R or FS)* **then**

  $\mathscr{T}_i^1 = \{i^1 \mid \forall i \in PI\}$;

**else**

  $\mathscr{T}^1 = \emptyset$;

**end**

Declare set of path variables $\mathscr{P}$, $\mathscr{P} = \{iR, iF, l \mid \forall i \in PI \text{ and } \forall l \in L\}$;

Build BDDs for line functionalities $f_l^1(\mathscr{T}^1), f_l^2(\mathscr{T}^2) \forall l \in L \cup PI$;

**if** *ST == (R or FS)* **then**

  Build BDDs for stability functions $S_l^0(\mathscr{T}), S_l^1(\mathscr{T}), \forall l \in L \cup PI$;

**end**

Convert all BDDs to ISOP-based ZBDDs;

% Traverse $C$ in topological manner;

**foreach** *line* $l \in L \cup PI$ **do**

  Call Sub-Sensitization Algorithm ;

**end**

$G_{circuit}(\mathscr{T} \cup \mathscr{P}) = \bigcup_{O \in PO} \left(G_O^R() \cup G_O^F()\right)$;

**Algorithm 3.1:** The Sensitization Algorithm

Let $PO$ define the set of all primary outputs. After all circuit lines are processed, the union of all primary output line functions, $\bigcup_{l \in PO} G_l^{tr}()$, gives the function that contains all sensitizable PDFs and their corresponding test cubes. Let this function be denoted by $G_{circuit}()$. The basic steps of the proposed algorithm are listed in the pseudocode of Algorithm 3.1. The input parameters are the circuit under consideration, $C$, and the desired sensitization type, $ST = \{R, NR, FS\}$ for Robust, Non-Robust and Functional Sensitization, respectively.

```
1  g = gate that drives line l;
2  FI(l) = fanin list of g;
3  if (l ∈ PI) then
4      if (ST==NR) then
5          use Eq. 3.1 and 3.2 for G_l^R() and G_l^F();
6      else if ST == (R or FS) then
7          use Eq. 3.7 and 3.8 for G_l^R() and G_l^F();
8      end
9  else if GT(g) == BUFF then
10     G_l^R() = G_j^R() and G_l^F() = G_j^F(), j ∈ FI(l);
11 end
12 else if GT(g) == NOT then
13     G_l^R() = G_j^F() and G_l^F() = G_j^R(), j ∈ FI(l);
14 end
15 else if GT(g) ∈ {AND, NAND, OR,NOR} then
16     if (cv_g → ncv_g) or (ST==NR) then
17         if GT(g) == (AND or NAND) then
18             use Eq. 3.3 for G_l^tr(), tr ∈ {R, F};
19         end
20         if GT(g)== (OR or NOR) then
21             use Eq. 3.4 for G_l^tr(), tr ∈ {R, F};
22         end
23     else if (ncv_g → cv_g) and (ST==R) then
24         use Eq. 3.9 for G_l^tr(), tr ∈ {R, F};
25     end
26     else if (ncv_g → cv_g) and (ST==FS) then
27         use Eq. 3.10 for G_l^tr(), tr ∈ {R, F};
28     end
29     if g == INV (inverting gate) then
30         G_temp() = G_l^R(), G_l^R() = G_l^F() and G_l^F() = G_temp();
31     end
32 end
```

**Algorithm 3.2:** The Sub-Sensitization Algorithm

The set of internal lines is denoted by $L$. Algorithm 3.2, *Sub-Sensitization Algorithm* , is a sub-algorithm of the sensitization Algorithm 3.1.

The algorithm for constructing the ISOPs/ZBDD traverses the circuit in a topological order, starting from the primary inputs. The first 10 steps of Algorithm 3.1 are some necessary initialization steps and declarations. Step 11-14 constructs the BDD line functionalities or stability functionalities for all primary inputs, in linear time. Step 15, converts all BDDs to ISOP/ZBDDs. This is a standard ZBDD operation which in the worst case is polynomial. Lines 16-19 traverse the circuit and when a circuit line $l$ is visited, its corresponding sensitization functions are generated (Algorithm 3.2), based on the previously generated sensitization functions of the line's immediate predecessor lines. The number of standard ZBDD operations per line $l$ is $(n-1)^2$, where $n$ is the number of immediate predecessor lines of $l$ (gate fanin), and thus polynomial. Thus, the complexity of the Sub-Algorithm 3.2 is polynomial in the worst case, since it is executed in a liner number, $(n-1)^2$, of standard ZBDD operations. The last step of Algorithm 3.1, 20, takes the standard ZBDD union operation of polynomial complexity at the worst case, of all the the sensitization functions generated on all the primary outputs. All operations used are of polynomial complexity and the number of operations are linear per circuit line. Thus, the total number of standard ZBDD operations used in Algorithm 3.1 is $|Lines| * (n-1)^2$. Thus, the overall complexity of the algorithm is in the worst case polynomial.

Lines 1-15 of Algorithm 3.1 list necessary preprocessing steps to define all Boolean variables and construct line functionalities as BDDs, which are then converted to ISOPs-based ZBDDs (line 15) using the method of [77]. If the examined sensitization type is either R or FS, then the necessary stability functions are also created, first as BDDs (as in [54], [74]) and then converted to an ISOPs-based ZBDD format. The ISOPs/ZBDD graphs ($G_l^R$ and $G_l^F$) per line $l$ are constructed based on the type of sensitization (ST) considered and the type of the gate driving $l$, as outlined in Algorithm 3.2. The pseudocode refers to Eq. 3.1 - 3.10, for each appropriate case, which give the exact ZBDD operations performed.

To find the ZBDD that contains only all sensitizable PDFs, denoted by $P_{circuit}()$, suffices to Existentially abstract all variables in $\mathscr{T}$ from $G_{circuit}()$:

$$P_{circuit}(\mathscr{P}) \; = \; \underset{\forall v \in \mathscr{T}}{\exists} (G_{circuit}(),\; v)$$

$|P_{circuit}()|$ will give the exact number of sensitizable PDFs in the circuit. The ZBDD with all unsensitizable PDFs is derived by taking the set difference of the ZBDD representing all

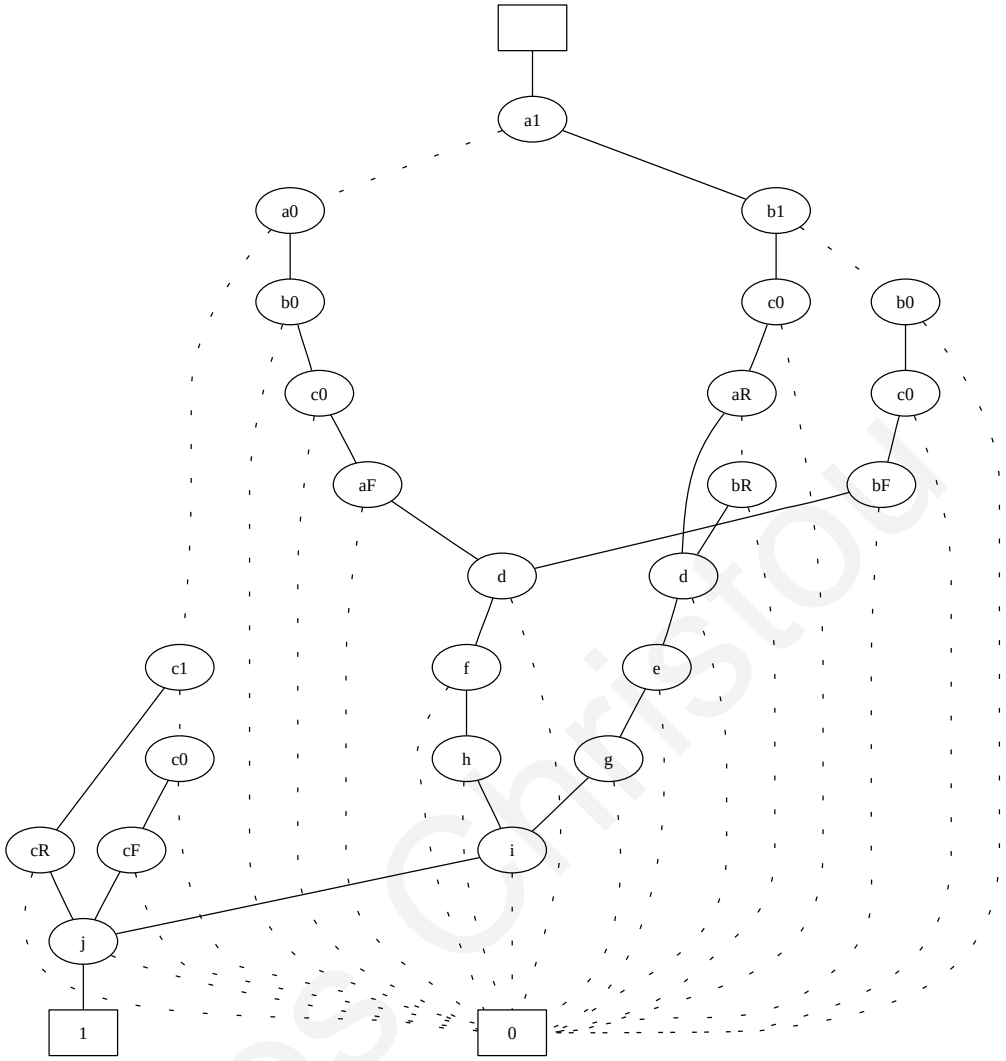



Figure 3.5: Function $G_{circuit}()$ for circuit $C$ of Fig. 2.3.

PDFs with $P_{circuit}()$.

The function that contains only all tests, denoted by $T_{circuit}()$, can also be derived by existentially abstracting all variables in $\mathscr{P}$ from $G_{circuit}()$.

Fig. 3.5 shows the ISOPs/ZBDD graph for $G_{circuit} = G_j^F() \cup G_j^R()$ of Fig. 2.3, for the non-robust case. Observe the paths from the root to the terminal-1 node, implying that there are 6 sensitizable PDFs that reach output $j$, 5 with a falling transition and 1 with a rising transition at $j$. Fig. 3.6 shows the *ZBDD* that contains only the sensitizable PDFs, after all test variables are existentially abstracted. The number of sensitizable PDFs can be calculated using the cardinality operator ($|P_{circuit}()|$), which is linear to the size of the graph.

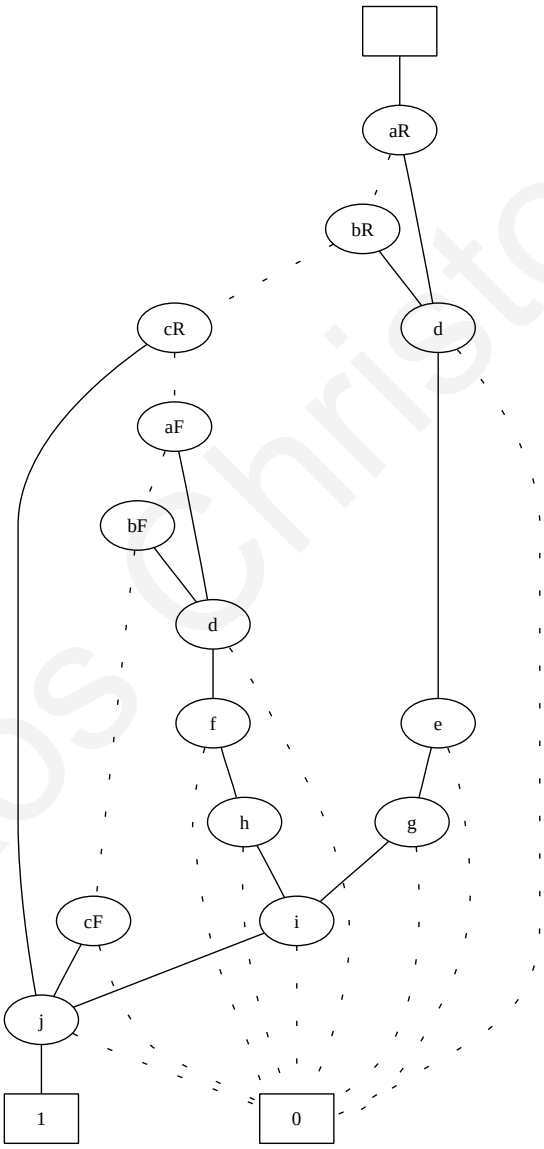



Figure 3.6: $P_{circuit}()$ for circuit $C$ of Fig. 2.3.

## 3.4 Finding Critical Sensitizable PDFs

The previous Section presented how the sensitizable PDFs and their tests can be derived and stored in an ISOPs/ZBDD graph. To consider critical PDF test generation, it is necessary to generate an ISOPs/ZBDD which contains only the critical faults. We show how the sensitization function of Section 3.3 can be restricted to only contain critical PDFs. This is achieved in a non-enumerative manner by appropriate manipulation of the ISOPs/ZBDD per circuit line, while taking under consideration a selected set of potentially critical PDFs represent by a ZBDD. A potentially critical PDF is not necessarily sensitizable.

Any previously proposed method for critical path selection, under various delay models, can be considered. Consequently, the selected paths can be encoded efficiently in ZBDD format. Depending on the path selection methodology used the encoding can be done in an enumerative or non-enumerative manner. In [85] for example, it is shown how a ZBDD containing all potentially critical PDFs, under the bounded delay model, can be derived without any path enumeration. The same method applies under the other traditional gate delay models, such as the fixed and min-max delay models. The remaining of this Section focuses on re-formulating the ISOPs/ZBDD sensitization function to only include critical PDFs.

Let $\mathscr{Z}$ denote the ZBDD containing the set of potentially critical PDFs. When considering critical paths, three major issues need to be considered when forming the sensitization function:

*(i)* How to only consider potentially critical circuit lines. A potentially critical line is one that is at least on one potentially critical PDF. This is achieved by appropriate examination of the paths in $\mathscr{Z}$.

*(ii)* How to identify the unsensitizable PDFs in $\mathscr{Z}$ and exclude them from the generated sensitization function. This is done automatically since the sensitization function formulation of Section 3.3, considers the various sensitization criteria and discards unsensitizable segments implicitly.

*(iii)* How to avoid introducing segments/PDFs that are not included in $\mathscr{Z}$. This problem appears when processing fanout stem lines that drive several branches, and it is common in methods that process paths in an non-enumerative manner [84], [48], [91].

We first address issue *(i)* and show the formulation of the critical PDF sensitization function for fanout-free circuits. Consequently, we discuss issue *(iii)* and present the solution. Issue *(ii)* has already been addressed in the function formulation of Section 3.3. Without any

loss of generality, we describe how to derive the ISOPs/ZBDD for critical non-robustly sensitizable PDFs. The formulation extensions for the robust and functional sensitization case can be derived in a similar fashion as in Section 3.3. The underlying operations necessary for the critical path extension are independent to the type of the PDF sensitization considered.

### 3.4.1 Fan-out free circuits

Let $G_l^{tr}()$, $tr = \{R, F\}$, denotes a rising or falling sensitization function containing all *critical* PDF segments up to line $l$, along with all their corresponding sensitization cubes.

When at some line $l$, it is necessary to determine whether $l$ is a potentially critical line. If not, $G_l^{tr}() = \emptyset$. To determine if $l$ is potentially critical, it suffices to find the subset of $\mathscr{Z}$ such that $l = 1$, given by $\mathscr{Z}_l = \subset (\mathscr{Z}, l)$ (see Table 3.1 for the standard ZBDD subset operation). If $\mathscr{Z}_l = \emptyset$ then $l$ is not critical. Otherwise, $\mathscr{Z}_l$ will contain all potentially critical PDFs through line $l$. According to the ZBDD subset operation, the variable of $l$ will not appear in $\mathscr{Z}_l$, since it is cofactored. The next step is to determine which of the immediate predecessor lines of $l$, given in *FI(l)*, are potentially critical. This is also determined using the subset operation. A line $y \in FI(l)$ is potentially critical if $\mathscr{Z}_y = \subset (\mathscr{Z}, y) \neq \emptyset$. Let $CI(l) \subseteq FI(l)$ denote the set of all immediate potentially critical predecessors of $l$.

Function $G_l^{tr}()$ is formulated with respect to the critical sensitization functions of its predecessor lines in *CI(l)*. The main idea here is to be able to find all potentially critical PDF segments up to line $l$ and through some line $y \in CI(l)$. Once this is derived, the sensitization conditions of the underlying sensitization type can be incorporated for the off-inputs, in order to drop any unsensitizable segments from the function. Thus, for the case of fanout-free circuits, the sensitization function for all critical PDF segments up to the output line $l$ of an AND gate, is given by:

$$G_l^{tr}() \;=\; ! \left( \left( \bigcup_{y \in CI(l)} \left( G_y^{tr}() \cap \left( \bigcap_{x \in FI(l), x \neq y} f_x^2() \right) \right) \right), l \right) \tag{3.11}$$

This formulation is identical to that of Eq. 3.3 in Section 3.3, with the exception that only critical predecessor lines of $l$ (*CI(l)*) are considered. The off-inputs sensitization conditions are enforced on all immediate predecessor in *FI(l)*.

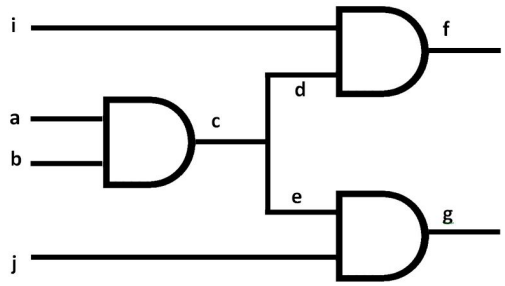



Figure 3.7: Example circuits $C_2$.

### 3.4.2 Treating Fan-Out Stems

When considering circuits with fanout sections, the formulation of Eq. 3.11 can introduce non-critical PDFs in $G_l^{tr}()$. We discuss why this occurs with the help of an example. Consider circuit $C_2$ in Fig. 3.7. It contains 12 PDFs. Let's consider only the rising PDFs. Let $\mathscr{Z}()$ contain 2 potentially critical PDFs, $\mathscr{Z}() = aR.c.d.f + bR.c.e.g$. For the primary inputs $G_a^R() = f_a^2().aR$ and $G_b^R() = f_b^2().bR$. At line $c$, both segments $aR-c$ and $bR-c$ are included, since they are both critical. Thus, $G_c^R() = f_a^2().f_b^2().aR.c + f_a^2().f_b^2().bR.c$. Up to this point, all functions contain the correct information. [2] Consider lines $f$ and $g$. Observe that only a subset of $G_c^R()$ is needed when computing the functions for these two lines, since $G_f^R()$ should only include path $aR-c-f$ and $G_g^R()$ should only contain path $bR-c-g$. However, based on the formulation of Eq. 3.11, $G_f^R() = f_a^2().f_b^2().f_i^2().aR.c.f + f_a^2().f_b^2().f_i^2().bR.c.f$, which includes the non-critical path $bR-c-f$. Similarly, $G_g^R()$ will include the non-critical path $aR-c-g$.

This problem is alleviated by appropriate manipulations on $\mathscr{Z}()$. Consider again a line $l$ and corresponding $\mathscr{Z}_l = \subset (\mathscr{Z}, l)$ which contains all potentially critical paths through line $l$. Removing from $\mathscr{Z}_l()$ all variables corresponding to lines not driving $l$, gives all potentially critical PDF segments up to $l$. This is done using the existential abstraction operator, as given below. The set of all lines not driving line $l$ is denoted by *ND(l)*, and it can be derived via a single circuit traversal.

$$\mathscr{Z}_{l,seg}() \quad = \quad \underset{v \in ND(l)}{\exists} (\mathscr{Z}_l, v)$$

Let $G_y^{tr}()$, $y \in CI(l)$, be the function with all critical PDF segments up to $y$. When computing the critical sensitization function for line $l$, it is necessary to only consider those critical

[2] Branch lines and single input gates have been explicitly represented mainly for illustration and clarification purposes. We do not use them in our implementation.

segments in $G_y^{tr}()$, and corresponding tests, that also pass through line $l$. This is achieved by computing the intersection of $G_y^{tr}()$ with $\mathscr{Z}_{l,seg}()$. $\mathscr{Z}_{l,seg}()$ must be extended with all test variables, otherwise the outcome of the intersection will be the empty set. This occurs because $G_y^{tr}()$ also contains the test cubes per critical segment. In order to maintain the correctness of the test set, all test variables are incorporated in $\mathscr{Z}_{l,seg}()$ at their don't care state, as shown below:

$$\mathscr{Z}_{l,seg}^{X}() \quad = \quad !(!(\mathscr{Z}_{l,seg}(), \overline{v_0}), \overline{v_1}), \forall v \in \mathscr{T}$$

The intersection will give the ISOPs/ZBDD with all critical segments up to line $l$ through line $y$, along with the sensitizatizing tests up to line $y$. Consequently, the sensitization function for all critical segments up to the output line $l$ of an AND gate, is given by:

$$G_l^{tr}() =! \left( \left( \bigcup_{y \in CI(l)} ((\mathscr{Z}_{l,seg}^{X}() \cap G_y^{tr}()) \cap ( \bigcap_{x \in FI(l), x \neq y} f_x^2())) \right), l \right) \tag{3.12}$$

The functions for other gate types and sensitization criteria are derived in a similar manner as in Eq. 3.12. Continuing with the example of Fig. 3.7, $\mathscr{Z}_{f,seg}^{X}() = aR.c$ (branch variable $d$ can be ignored). $G_c^R() \cap \mathscr{Z}_{f,seg}^{X}() = aR.c$ removes segment $bR.c$ and, thus, $G_c^R() = f_a^2().f_b^2().f_i^2().aR.c.f$ is now correct.

The following example illustrates the new operations introduced in this subsection. Circuit $C_3$ of Fig. 3.8 contains 12 PDFs. Assume that all paths that pass through three or more gates are considered critical. In this case, $\mathscr{Z}()$ contains 8 potentially critical PDFs, as shown in Fig. 3.9. These are $\{bR-f-j-l-m, bF-f-j-l-m, cR-j-l-m, cF-j-l-m, bR-g-h-k-l-m, bF-g-h-k-l-m, dR-k-l-m, dF-k-l-m\}$. The potentially 4 critical PDFs through line $k$ are given in $\mathscr{Z}_k()$ in Fig. 3.10. Variable $k$ does not appear in Fig. 3.10, since it is cofactored by the subset operation $\mathscr{Z}_k() = \subset (\mathscr{Z}, k)$. The ZBDD with all PDF segments up to $k$ (excluding $k$) is given in $\mathscr{Z}_{k,seg}()$ in Fig. 3.11. The ISOPs/ZBDD for all critical PDFs is shown in Fig. 3.13. It contains only 4 critical PDFs $\{bR-f-j-l-m, cR-j-l-m, cF-j-l-m, bR-g-h-k-l-m\}$. The remaining 4 PDFs in $\mathscr{Z}()$ are implicitly identified as non-sensitizable and are not included. Fig. 3.12 shows the ISOPs/ZBDD for all sensitizable PDFs in $C_3$ which is actually a superset of Fig. 3.13.

The critical PDF sensitization algorithm follows the same basic steps outlined in Algorithm 3.1. In addition, it accepts or creates the ZBDD of the potentially critical PDFs $\mathscr{Z}()$,

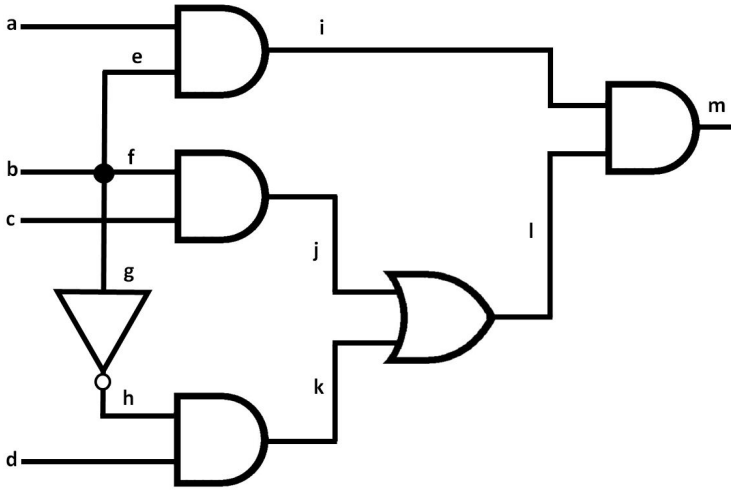



Figure 3.8: Example circuits $C_3$.

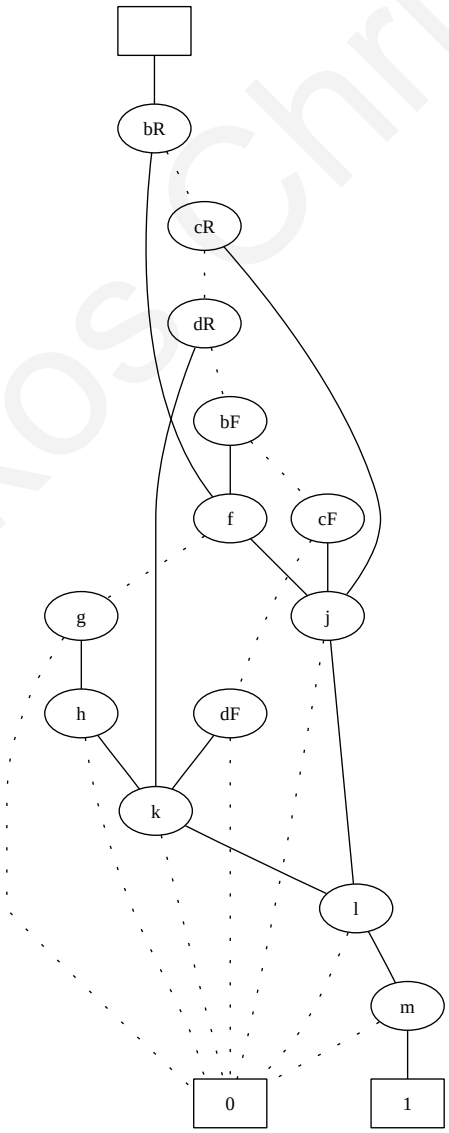



Figure 3.9: $C_r$ for circuit $C_3$.

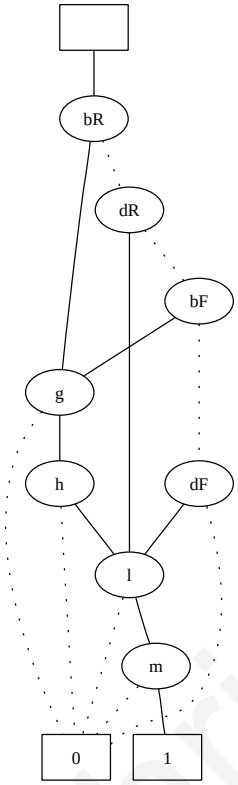



Figure 3.10: Subset Operation on k of $C_3$.

and, before forming the critical sensitization function for a line $l$, it computes $\mathscr{Z}^X_{l,seg}()$. Eq. 3.12 replaces Eq. 3.3 in Algorithm 3.1. All other Eq. are modified appropriately in a similar manner as Eq. 3.12.

## 3.5 Generation of Tests with Maximal Test Efficiency

This Section discuses the generation of a compact test set by generating tests with high test efficiency. It is shown how a test with maximum test efficiency, under the given ordering of variables in the ISOPs/ZBDD (i.e changing the variable ordering can lead to different TE), is derived. Consequently, additional tests with high test efficiency are derived by appropriately removing the detected PDFs and repeating the process. Thus, each generated test detects a large number of PDFs that have not been detected by already generated tests. No unsensitizable PDFs are targeted in this process.

The challenging task in finding a compact test set is finding a test with high test efficiency. Therefore, the first part of this Section is focused on this task, while the second part describes how additional tests with the same property can be derived.

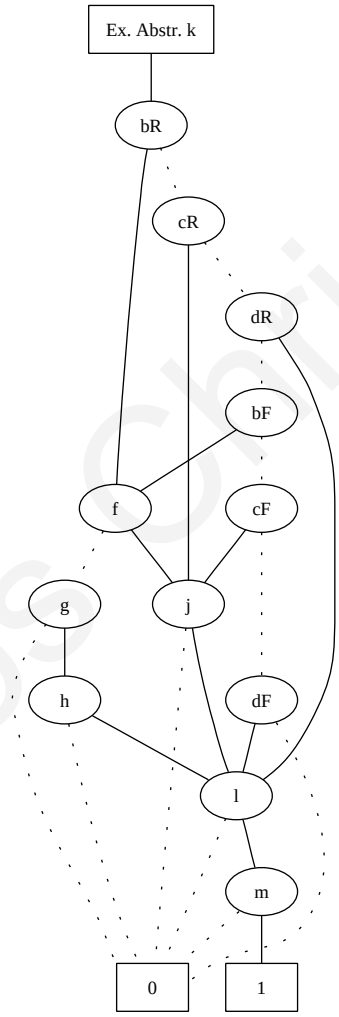



Figure 3.11: Existentially Abstract Operation on k of $C_3$.

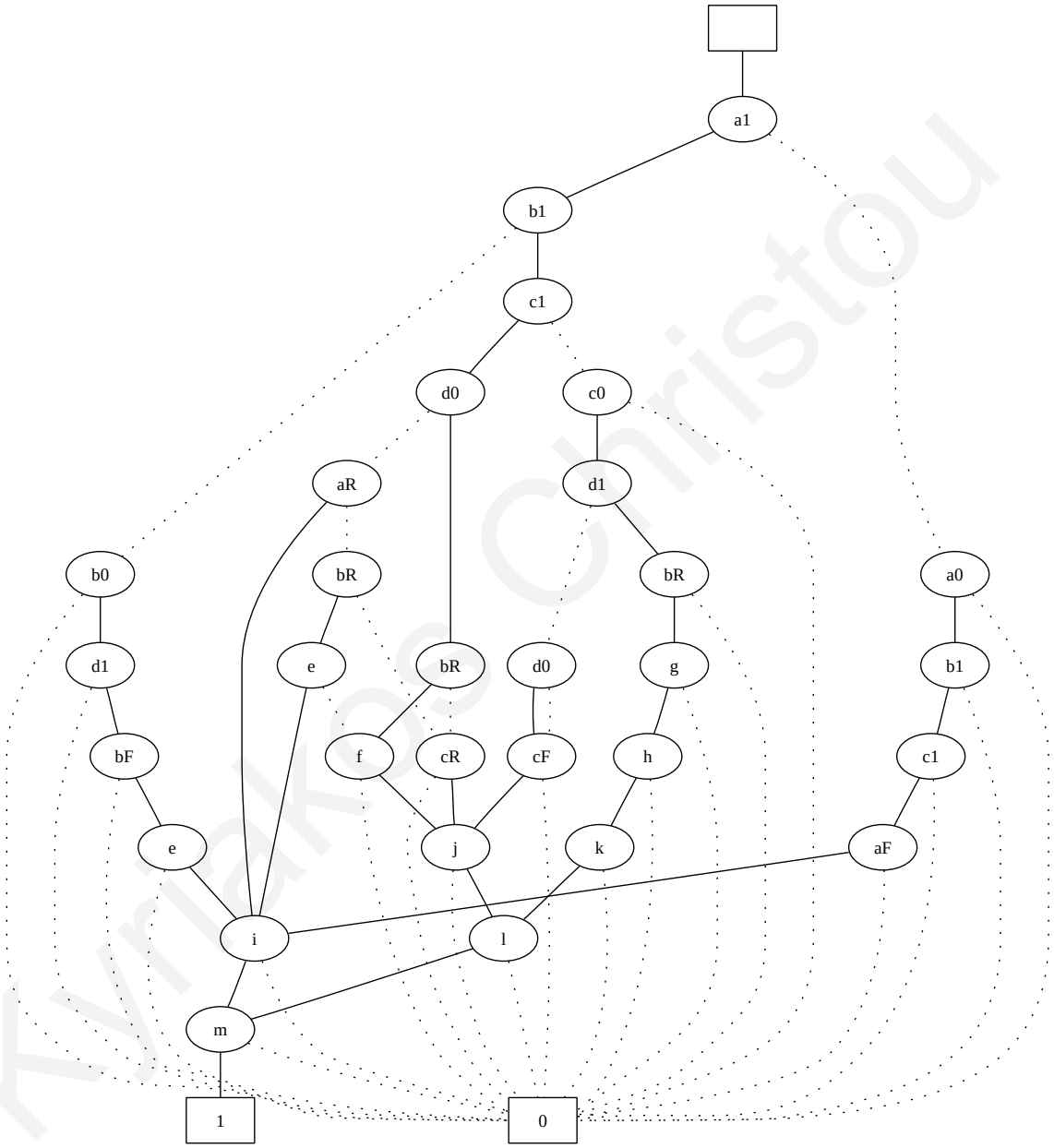



Figure 3.12: ISOPs/ZBDD for all sensitizable PDFs in $C_3$ of Fig. 3.8.

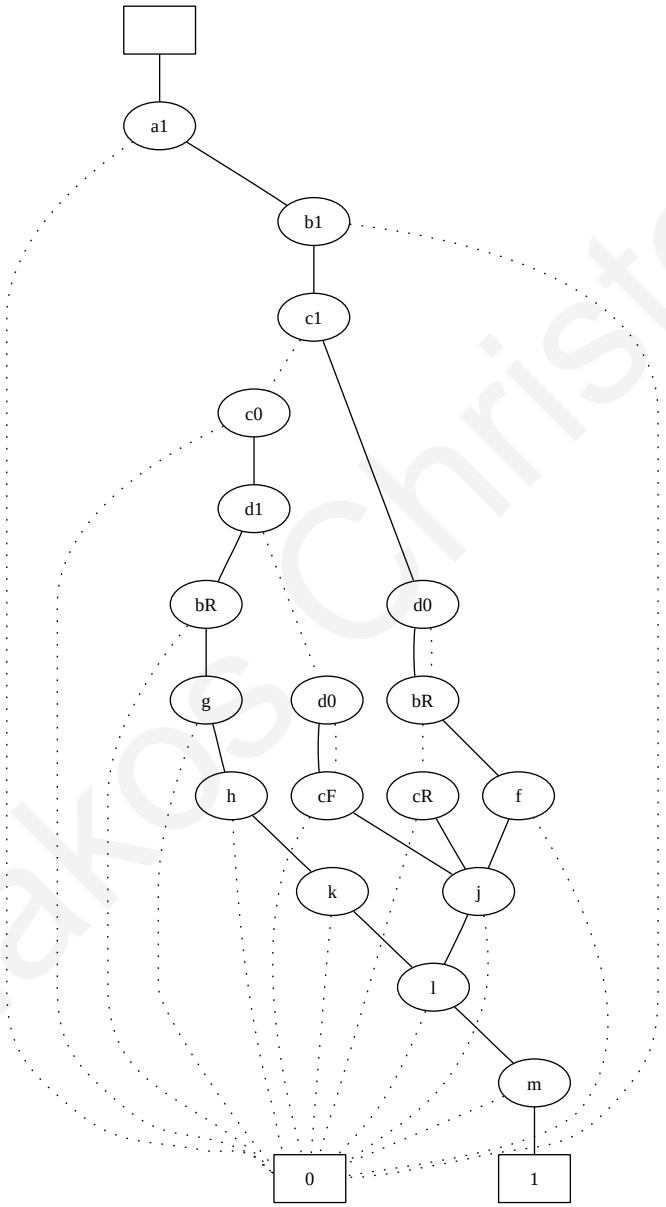



Figure 3.13: ISOPs/ ZBDD for critical PDFs in $C_3$ of Fig. 3.8.

### 3.5.1 Generation of the $\mathscr{T}$-graph

The basic idea is centered on finding one test in the ISOPs/ZBDD graph that shares the largest number of paths from the root to the terminal-1 node. This amounts to finding a set of test nodes in the graph that cover the maximum number of such paths.

The ISOPs/ZBDD nodes are classified as test nodes ($\mathscr{T}$-node) if they correspond to variables in $\mathscr{T}$ or path nodes ($\mathscr{P}$-node) if they correspond to variables in $\mathscr{P}$. Observe that, in decision diagrams such as the one considered here, several nodes in the graph can correspond to the same variable. Let the ISOPs/ZBDD graph be denoted by $G$, and $N^{\mathscr{T}}$ denote all $\mathscr{T}$-nodes in $G$. A graph that contains all $\mathscr{T}$-nodes in the original ISOP/ZBDD graph, plus the terminal-1 node, is derived. In this graph, referred to as the $\mathscr{T}$-graph, an edge from a $\mathscr{T}$-node $v$ to a $\mathscr{T}$-node (or terminal-1 node) $u$ exists if $u$ is the first $\mathscr{T}$-node reachable from $v$ through some path in $G$. This information for all $\mathscr{T}$-nodes can be found via a single traversal of $G$. Furthermore, a weight $w(v \rightarrow u)$ is defined per edge, to be the number of paths from $v$ to $u$ in $G$. The weights can also be calculated via a graph traversal. Without any loss of generality, it is assumed that the root of $G$ is a $\mathscr{T}$-node (this always holds with variable reordering). The $\mathscr{T}$-graph has a single source, the root of $G$, and a single destination, the terminal-1 node. A post-processing step is necessary after all edges and weights have been derived to remove $\mathscr{T}$-nodes (and their incoming edges), other than the terminal-1 node, in the $\mathscr{T}$-graph that have no successors. Such nodes do not have any paths to the terminal-1 node in $G$, which implies that they are not in the function on-set. The basic steps of the algorithm that builds the $\mathscr{T}$-graph are listed in Algorithm 3.3. $V$ is the set of nodes and $E$ the set of edges of the generated $\mathscr{T}$-graph.

Algorithm 3.3, generates the $\mathscr{T}$-graph via a traversal on a DAG, the ISOP/ZBDD structure. Starting from the root node, steps 4-14, traverse the input ISOP/ZBDD and visits all nodes of the data structure while constructing the $\mathscr{T}$-graph, which contains only the test nodes of the input ISOP/ZBDD. Thus, the overall complexity for generating such a graph is of linear complexity to the size of the input ISOPs/ZBDD graph.

As an example, consider circuit $C_3$ of Fig. 3.8. The ISOPs/ZBDD for all (critical) PDFs in $C_3$ was given in Fig. 3.12 (Fig. 3.13). The $\mathscr{T}$-graph for each of these case is shown in Fig. 3.14 and Fig. 3.15, respectively. All $\mathscr{T}$-nodes and the terminal-1 node from the ISOPs/ZBDDs appear in the $\mathscr{T}$-graphs (ignore the numbers in the square brackets in each of the nodes of $\mathscr{T}$-graph at this point). Edges can be of three types. A solid (dotted, dashed) edge leaving some node $v$ implies that $v = 1$ ($v = 0$,$v = X$) in the test cube.

**Data**: $G(\mathscr{T} \cup P)$ ISOPs/ZBDD graph

**Result**: $\mathscr{T}$-graph$(V,E)$

Find node set $N^{\mathscr{T}} = \{v, \mid v \in \mathscr{T}\text{-node of } G(\mathscr{T} \cup P)\}$ ;

Vertex set $V = \{N^{\mathscr{T}} \cup$ (terminal-1 node)$\}$;

% Traverse $G$ in topological order, starting from the root node;

**foreach** *node* $v \in N^{\mathscr{T}}$ **do**

    $s(v) =$ list of all immediate successors of $v$ in $N^{\mathscr{T}}$;

    **foreach** $u \in s(v)$ **do**

        Add edge $(v,u)$ in $E$, if there exists a path from $v$ to $u$;

        $w(v,u) =$ number of paths from $v$ to $u$;

    **end**

    **foreach** *node* $u \in N^{\mathscr{T}}$ **do**

        **if** $u \neq$ *terminal-1 node AND* $s(u) == \emptyset$ **then**

            $N^{\mathscr{T}} = N^{\mathscr{T}} - u$;

        **end**

    **end**

**end**

**Algorithm 3.3:** $\mathscr{T}$-Graph_Create

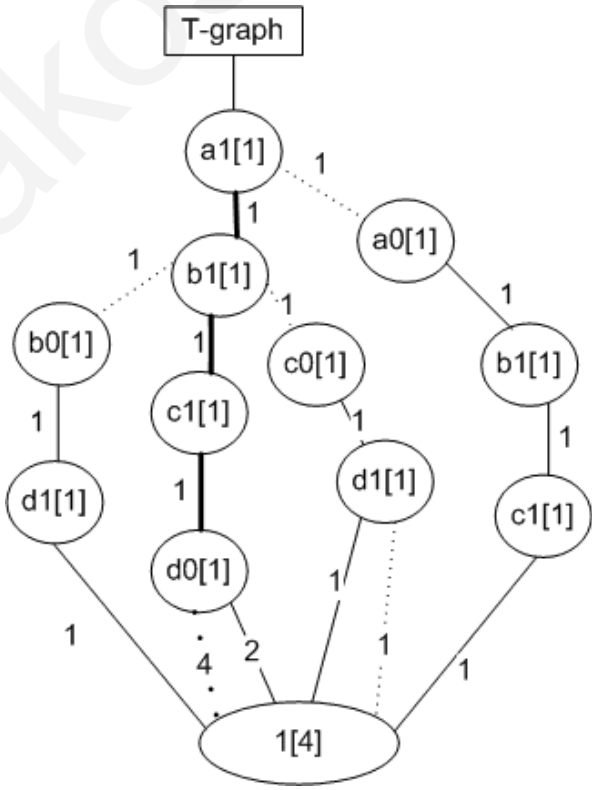



Figure 3.14: T-graph for the ISOPs/ZBDD of Fig. 3.12.

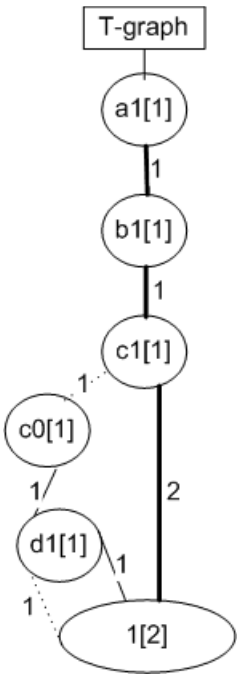



Figure 3.15: $\mathscr{T}$-graph for the ISOPs/ZBDD of Fig. 3.13.

The test that detects the maximum number of PDFs, under the variable ordering of the ISOPs/ZBDD graph, is the path in $\mathscr{T}$-graph that terminates to the terminal-1 node with the maximum weight. In general, let a path $P$ be denoted as a collection of consecutive nodes $e_1 \rightarrow e_2 \rightarrow \ldots \rightarrow e_n$. The weight of $P$, denoted by $W(P)$ is the product of the weights on its edges. Thus,

$$W(e_n) = \prod_{i=1\ldots n} w(e_i)$$

A topological traversal of the $\mathscr{T}$-graph suffices to calculate the maximum weight among all paths up to each node, which leads to the calculation of the maximum weight of a path in the graph. The root node $r$ is always initialized to $W(r) = 1$. When at some node $v$, with immediate predecessor nodes in $FI(v)$, then $W(v)$ is given by:

$$W(v) = \max_{i \in FI(v)} \{W(i) \times w(i \rightarrow v)\}$$

The maximum weighted path can be easily found by backtracking appropriately from the terminal-1 node to the immediate predecessors, until the root node is reached. The complexity cost is linear since it just linear traversals on the $\mathscr{T}$-graph.

Fig. 3.14 and Fig. 3.15 show the $W(v)$ value per node in the $\mathscr{T}$-graph in square brackets inside each node. The maximum weighted path in the $\mathscr{T}$-graph of Fig. 3.14 is $a1 - b1 - c1 - d0 - 1$ with weight 4 and, thus, the test cube $a \cdot b \cdot c \cdot \overline{d}$ is the highest quality test (it can

sensitize 4 PDFs). The maximum weighted path in the $\mathscr{T}$-graph of Fig. 3.15 is $a1-b1-c1-1$, with weight 2.

For this example, where non-robust sensitization was considered, the derived test determines the value of vector $v^2$ for the PDF test $(v^1, v^2)$. A traversal on the ISOP/ZBDD can determine the PDFs detected by the test. Then, $v^1$ is determined by the primary input path variables on the detected PDFs, by a linear number (to the number of primary inputs) subset operations. Continuing with the example of Fig. 3.14, $v^2 = a \cdot b \cdot c \cdot \overline{d}$ (=$a1 \cdot b1 \cdot c1 \cdot d0$) leads to PDFs $aR-i-m$, $bR-e-i-m$, $bR-f-j-l-m$ and $cR-j-l-m$ and, hence, $v^1 = \overline{a} \cdot \overline{b} \cdot \overline{c} \cdot x$ ($d$ is a don't care). This process is not necessary for robust or functional sensitization, since the test will contain the values for both $v^1$ and $v^2$ vectors.

### 3.5.2 Deriving additional tests

We now discuss how additional tests with maximal TE can be derived. The idea here is to be able to remove the PDFs detected by the maximum weighted test extracted by the $\mathscr{T}$-graph from the ISOPs/ZBDD, and repeat the process to find another test with high test efficiency.

Assume that the first test, $t$, has been generated. Let the original ISOPs/ZBDD be denoted by $G()$. A single traversal from the root to the terminal-1 node of $G()$, restricted by the values of the variables in $t$, will give the ZBDD that contains all PDFs detected by $t$, denoted by $G_t()$. Then, $G_t()$ is extended to an ISOPs/ZBDD, denoted by $G'()$, by insisting that all test variables in the circuit appear as don't cares, which is encoded as $\overline{v_0} \cdot \overline{v_1}$ for a variable $v$ in ISOPs-based ZBDD form. This is achieved by a linear number, to the primary inputs, of change (!) operators, as shown below:

$$G'() = !(\ !(G_t(), \overline{v_0}),\ \overline{v_1}), \forall v \in \mathscr{T}$$

It suffices to remove $G'()$ from $G()$, using the set difference operation $G() \backslash G'()$, to remove all PDFs detected by $t$ from $G()$. Consequently, the next test generated will be maximal and guarantees to detect only new PDFs. The process just described can be repeated until a fixed number of tests is derived or the desired fault coverage is achieved. The basic steps of the proposed ATPG algorithm are listed in Algorithm 3.4. $T_{cov}$ and $T_{th}$ denote user defined parameters that can be used to terminate the ATPG process when a desired coverage or number of tests has been reached. Alternately the process can run until 100% coverage is achieved. The algorithm accepts as input an ISOPs/ZBDD for the targeted PDFs, denoted by $G()$. $\mathscr{T}$ denotes the set of test variables.

**Data**: $G(), T_{cov}, T_{th}$

**Result**: Compact test set $T$

**repeat**

  $\mathscr{T}$-graph-Create($G(), \mathscr{T}$-graph$(V,E)$);

  **foreach** *node* $u \in V$ **do**

    FI($u$) = immediate predecessor nodes of $u$;

    % calculate maximum path weight up to $u$;

    $W(u) = \max_{i \in FI(u)} \{w(i) \times w(i \rightarrow u)\}, i \in FI(u)$;

  **end**

  $t =$ path with maximum weight in $\mathscr{T}$-graph;

  Derive $G_t()$ from $G()$;

  $G'() = !(\ !(G_t(), \overline{v_0}),\ \overline{v_1}), \forall v \in \mathscr{T}$;

  $G() = G() \backslash G'()$;

**until** *Coverage*==100% *OR Coverage*==$T_{cov}$ *OR Coverage*==$T_{th}$;

**Algorithm 3.4:** Compact-ATPG Algorithm

The ATPG process is linear to the size of the ISOPs/ZBDD graph, per generated test. Each iteration is faster than the previous one since the size of the ISOPs/ZBDD graph decreases. The coverage obtained during each operation is trivially calculated without employing fault simulation. If $G() == \emptyset$, then a 100% coverage is reached since no more undetectable PDFs remain in the ISOPs/ZBDD. The achieved coverage is calculated by dividing the number of PDFs detected by the total number of PDFs in $G()$. This requires finding the cardinality of $G()$, which is a standard ZBDD operation that amounts to a linear traversal on $G()$. The coverage calculation is exact, in contrast to all existing practical ATPG methods, since the exact number of sensitizable PDFs is known.

The overall complexity of Algorithm 3.4, depends on the number of iterations executed. Each step has a linear complexity, and as discussed before each iteration is faster than the previous one. In the worst case where the 100% coverage is needed, and assuming that test efficiency is a test per fault then, in this case the algorithm stops executing and takes a random test per PDF. Experimental results indicate that Algorithm 3.4 is much faster than a test per fault approach and produces highly compact tests.

## 3.6 Experimental Results

The proposed method was implemented in C language on top of the decision diagram package of [107] (for all BDD and ZBDD related operations), and run on a 1GHz SunBlade 1500 workstation. We report results for all *ISCAS'*85, except for C6288, and the enhanced full-scanned *ISCAS'*89 circuits. C6288 cannot be represented directly using BDDs, however, such circuits could be represented in another appropriate canonical form, such as partitioned-BDDs [80], which can then be converted to ZBDD-based ISOPs. The ZBDD representation of all $\sim 10^{20}$ PDFs of C6288 has been shown to be very efficient in [84]. The initial variable ordering in the BDDs was the one provided along with the tool of [107]. No variable reordering, which can be used to reduce the size of the decision diagrams, was activated in any of the reported experiments. We present results for the non-robust sensitization criterion.

Table 3.2 and 3.3 reports the resource requirements, for time and space, for the generated ISOPs/ZBDDs. In both tables 3.2 and 3.3 Column 2 lists the total number of PDFs per circuit. Column 3 gives the exact number of sensitizable PDFs per circuit, obtained after generating the ISOPs/ZBDD graph for the sensitization function presented in Section 3.3. No fault was aborted.

Examining Table 3.2 the maximum number of PDFs detected by a single test, denoted by MaxTE, is listed in Column 5, obtained by the ATPG algorithm of Section 3.5. Time (in CPU seconds) and memory (in MBytes) needed to construct the sensitization function and generate the test of MaxTE are given in Columns 6 and 7, respectively. Both requirements are very small for the small circuits. For the larger circuits, memory requirements are increased due to the large number of PDFs in the circuits and, more importantly, the enormous number of tests that exists under the non-robust sensitization criterion. For certain circuits, the single traversal approach to examine all circuit paths, required a large amount of memory to build the sensitization functions. In such cases, the circuit was divided into partitions such that no two partitions contained the same PDFs. The approach was applied on each of these partitions. These circuits are indicated by a * in Column 1 of Table 3.2. Partitions were derived either by considering all paths terminating to the same primary output or, in certain cases, all paths starting at the same primary input and terminating at the same primary output. The total number of sensitizable PDFs reported is the sum of the sensitizable PDFs in each partition. Since the partitions are non-overlapping, they can be examined independently and, thus, can all be processed in parallel. Time and memory reported for each of these circuits is the maximum required among all partitions. TE can be impacted when partitions are

considered, since only a subset of the circuit paths are considered at one time.

Considering both Tables 3.3 and 3.3, column 4 lists the number of critical PDFs, obtained after generating the ISOPs/ZBDD graph for critical PDFs (Section 3.4). The proposed method can consider various delay models and critical path selection methods. In our experimentations, we used the bounded delay model. Delay ranges for each gate were obtained from the TSMC 0.18 *micron* technology files using the corner values for the nMoS and pMOS transistors. The delay threshold was set as *90%* of the circuit delay, obtained using static timing analysis. A single topological traversal was used to implicitly identify all *potentially* critical PDFs. The numbers listed in Column 4 are those identified by our approach as critical. The MaxTE, time and space required to derive the critical PDFs and the test with the maximum test efficiency are given in Columns 5, 6, and 7, respectively of Table 3.3. For critical PDFs, a single sensitization function was derived for every circuit (no partitions were considered). This explains why the time required is sometimes larger for critical PDFs than the one required for all sensitizable PDFs. However, considering the entire circuit at once contributes to a considerable increase in MaxTE in most of the larger circuits, as it can be seen by comparing the results reported in Column 5 in Tables 3.2 and 3.3.

The overwhelming majority of the CPU time spent and the memory used, in both cases (all sensitizable PDFs and critical PDFs), is attributed to the ISOPs/ZBDD generation and not to the ATPG task, since the latter is archived by only a linear traversal of the ISOPs/ZBDD. This is very important, since it allows for the generation of complete test sets in linear time to the size of the ISOPs/ZBDD. The only factor that impacts the ATPG time is the number of generated tests (since an ISOPs/ZBDD traversal per test is necessary) but, this is also optimized since the tool's goal is to maximize the test efficiency and hence, the generated test set is inherently compact.

Figures 3.16, 3.17, 3.18, 3.19 demonstrate that the same trend exist for all benchmark circuit examined. Each of these figures consists of three sub-figures. The first sub-figure shows number of $PDFs$ per test, the second sub-figure shows the total number of detected $PDFs$ per test and the third and last sub-figure reports the average Test Efficiency, *AvgTE*, per test. Even when considering large circuits someone can see that all the circuits follow the same trend for all the circuits in all the three examined cases.

Table 3.4 reports results that demonstrate the scalability of the approach for critical PDF test generation. The average test efficiency for test sets with different number of tests is given. The average test efficiency, denoted by *AvgTE*, listed in Columns 4, 6, 8, 10, 12 and 13, is the ratio of all detected PDFs (given in Columns 3, 5, 7, 9 and 11) over the total number

Table 3.2: Resource requirements for the ISOPs/ZBDD and MaxTE for All PDFs

| Circuit Name | Total PDFs | Sens. PDFs | Critical PDFs | All PDFs | | |
|---|---|---|---|---|---|---|
| | | | | MaxTE | Time(secs) | Mem(MBs) |
| s208.1 | 284 | 284 | 12 | 10 | 0.05 | 0.53 |
| s386 | 414 | 414 | 78 | 28 | 0.07 | 0.66 |
| s298 | 462 | 364 | 45 | 14 | 0.03 | 0.29 |
| s344 | 710 | 654 | 50 | 10 | 0.04 | 0.60 |
| s349 | 730 | 656 | 50 | 10 | 0.07 | 0.80 |
| s510 | 738 | 738 | 14 | 31 | 0.18 | 5.20 |
| s382 | 800 | 734 | 94 | 19 | 0.08 | 0.62 |
| s526n | 820 | 718 | 45 | 14 | 0.06 | 0.57 |
| s420.1 | 948 | 948 | 20 | 18 | 2.19 | 28.31 |
| s820 | 984 | 984 | 84 | 34 | 0.38 | 6.56 |
| s832 | 1012 | 996 | 84 | 36 | 0.41 | 6.68 |
| s444 | 1070 | 813 | 72 | 19 | 0.11 | 4.81 |
| s1488 | 1924 | 1916 | 6 | 50 | 0.77 | 16.82 |
| s1494 | 1952 | 1927 | 4 | 50 | 0.77 | 16.40 |
| s953n | 2312 | 2312 | 54 | 56 | 0.87 | 16.79 |
| s641 | 3488 | 2270 | 142 | 21 | 1.29 | 21.48 |
| s1196 | 6196 | 3759 | 104 | 81 | 3.92 | 30.81 |
| s1238 | 7118 | 3684 | 90 | 96 | 7.32 | 38.81 |
| C880 | 17284 | 16652 | 3768 | 94 | 70.50 | 43.59 |
| s3271 | 38388 | 19292 | 2421 | 87 | 12.94 | 95.27 |
| s713 | 43624 | 4922 | 613 | 185 | 2.41 | 24.71 |
| s1423 | 89452 | 45198 | 890 | 100 | 144.60 | 547.57 |
| C2670* | 1359920 | 130626 | 56982 | 407 | 24.22 | 55.91 |
| C7552* | 1452988 | 277244 | 21680 | 280 | 254.70 | 92.65 |
| C1908* | 1458114 | 355168 | 67584 | 117 | 1150.80 | 196.58 |
| s38584.1* | 2161446 | 334927 | 20444 | 603 | 1034.91 | 670.54 |
| C5315* | 2682610 | 342117 | 64032 | 593 | 159.13 | 72.31 |
| s13207* | 2690738 | 476145 | 64681 | 271 | 367.22 | 233.70 |
| C1355* | 8346432 | 1110304 | 1008800 | 345 | 539.18 | 252.31 |

* denotes partitioning-based processing on primary output or primary input/primary output basis for All PDFs (Columns 5-7)

Table 3.3: Resource requirements for the ISOPs/ZBDD and MaxTE for Critical PDFs

| Circuit Name | Total PDFs | Sens. PDFs | Critical PDFs | Critical PDFs | | |
|---|---|---|---|---|---|---|
| | | | | MaxTE | Time(secs) | Mem(MBs) |
| s208.1 | 284 | 284 | 12 | 3 | 0.06 | 0.22 |
| s386 | 414 | 414 | 78 | 5 | 0.12 | 0.38 |
| s298 | 462 | 364 | 45 | 10 | 0.11 | 0.30 |
| s344 | 710 | 654 | 50 | 10 | 0.22 | 0.30 |
| s349 | 730 | 656 | 50 | 10 | 0.23 | 0.31 |
| s510 | 738 | 738 | 14 | 3 | 0.18 | 0.28 |
| s382 | 800 | 734 | 94 | 6 | 0.46 | 0.39 |
| s526n | 820 | 718 | 45 | 10 | 0.18 | 0.25 |
| s420.1 | 948 | 948 | 20 | 4 | 0.39 | 0.38 |
| s820 | 984 | 984 | 84 | 8 | 0.79 | 0.48 |
| s832 | 1012 | 996 | 84 | 8 | 0.78 | 0.42 |
| s444 | 1070 | 813 | 72 | 12 | 0.36 | 4.25 |
| s1488 | 1924 | 1916 | 6 | 2 | 0.21 | 0.09 |
| s1494 | 1952 | 1927 | 4 | 2 | 0.11 | 0.08 |
| s953n | 2312 | 2312 | 54 | 12 | 0.77 | 0.24 |
| s641 | 3488 | 2270 | 142 | 21 | 3.68 | 1.09 |
| s1196 | 6196 | 3759 | 104 | 26 | 3.85 | 0.79 |
| s1238 | 7118 | 3684 | 90 | 17 | 2.98 | 1.03 |
| C880 | 17284 | 16652 | 3768 | 72 | 63.08 | 39.05 |
| s3271 | 38388 | 19292 | 2421 | 46 | 18.49 | 40.87 |
| s713 | 43624 | 4922 | 613 | 160 | 6.05 | 3.02 |
| s1423 | 89452 | 45198 | 890 | 48 | 12.44 | 6.50 |
| C2670 | 1359920 | 130626 | 56982 | 600 | 20.17 | 51.39 |
| C7552 | 1452988 | 277244 | 21680 | 512 | 197.41 | 81.03 |
| C1908 | 1458114 | 355168 | 67584 | 768 | 1053.80 | 180.32 |
| s38584.1 | 2161446 | 334927 | 20444 | 470 | 975.15 | 597.03 |
| C5315 | 2682610 | 342117 | 64032 | 864 | 147.35 | 68.45 |
| s13207 | 2690738 | 476145 | 64681 | 792 | 1105.81 | 679.31 |
| C1355 | 8346432 | 1110304 | 1008800 | 1152 | 489.81 | 231.13 |

Table 3.4: AvgTE for different test sets

| Circuit | Critical PDFs | Max # tests=10 | | Max # tests=20 | Max # tests=30 | | Max # tests=40 | | Max # tests=50 | | 100% Cov. | AvTE |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Ded | AvgTE | Ded | AvgTE | Ded | AvgTE | Ded | AvgTE | Ded | AvgTE | |
| s208.1 | 12 | 12 | 1.50 | - | - | - | - | - | - | - | - | 1.50 |
| s386 | 78 | 34 | 3.40 | 49 | 2.45 | 59 | 1.97 | 69 | 1.73 | 78 | 1.59 | 1.59 |
| s298 | 45 | 41 | 4.10 | 45 | 3.21 | - | - | - | - | - | - | 3.21 |
| s344 | 50 | 36 | 3.60 | 48 | 2.40 | 50 | 2.27 | - | - | - | - | 2.40 |
| s349 | 50 | 36 | 3.60 | 48 | 2.40 | 50 | 2.27 | - | - | - | - | 2.40 |
| s510 | 14 | 14 | 1.55 | - | - | - | - | - | - | - | - | 1.55 |
| s382 | 94 | 39 | 3.90 | 59 | 2.95 | 74 | 2.47 | 84 | 2.10 | 94 | 1.88 | 1.88 |
| s526n | 45 | 45 | 4.10 | - | 3.21 | - | - | - | - | - | - | 3.21 |
| s420.1 | 20 | 20 | 2.22 | - | - | - | - | - | - | - | - | 2.22 |
| s820 | 84 | 39 | 3.90 | 57 | 2.85 | 67 | 2.23 | 77 | 1.93 | 84 | 1.79 | 1.79 |
| s832 | 84 | 39 | 3.90 | 57 | 2.85 | 67 | 2.23 | 77 | 1.93 | 84 | 1.79 | 1.79 |
| s444 | 72 | 40 | 4.00 | 58 | 2.90 | 68 | 2.27 | 72 | 2.12 | - | - | 2.12 |
| s1488 | 6 | 6 | 1.20 | - | - | - | - | - | - | - | - | 1.20 |
| s1494 | 4 | 4 | 1.33 | - | - | - | - | - | - | - | - | 1.33 |
| s953n | 54 | 54 | 6.00 | - | - | - | - | - | - | - | - | 6.00 |
| s641 | 142 | 99 | 9.90 | 121 | 6.05 | 131 | 4.37 | 141 | 3.53 | 142 | 3.46 | 3.53 |
| s1196 | 104 | 67 | 6.70 | 88 | 4.40 | 103 | 3.43 | 104 | 3.35 | - | - | 3.35 |
| s1238 | 90 | 77 | 7.70 | 90 | 5.29 | - | - | - | - | - | - | 5.29 |
| C880 | 3768 | 504 | 50.40 | 821 | 41.05 | 1069 | 35.63 | 1285 | 32.13 | 1472 | 29.44 | 9.78 |
| s3271 | 2421 | 393 | 39.30 | 709 | 35.45 | 949 | 31.63 | 1149 | 28.73 | 1319 | 26.38 | 13.37 |
| s713 | 613 | 524 | 52.40 | 575 | 28.75 | 591 | 19.70 | 601 | 15.03 | 611 | 12.22 | 23.24 |
| s1423 | 890 | 304 | 30.40 | 464 | 23.20 | 572 | 19.07 | 652 | 16.30 | 716 | 14.32 | 13.21 |
| C2670 | 56982 | 4440 | 444.00 | 7428 | 371.40 | 10116 | 337.20 | 12132 | 303.30 | 13932 | 278.64 | 77.24 |
| C7552 | 21680 | 3378 | 337.80 | 5842 | 292.10 | 7558 | 251.93 | 8840 | 221.00 | 9838 | 196.76 | 56.96 |
| C1908 | 67584 | 8160 | 816.00 | 13847 | 692.35 | 18403 | 613.43 | 22897 | 572.42 | 25607 | 512.14 | 85.37 |
| s38584.1 | 20444 | 3154 | 315.40 | 5424 | 271.20 | 7052 | 235.06 | 7909 | 197.72 | 8766 | 175.32 | 51.29 |
| C5315 | 64032 | 7368 | 736.80 | 13208 | 660.40 | 17358 | 578.60 | 20844 | 521.10 | 24016 | 480.32 | 95.87 |
| s13207 | 64681 | 6980 | 698.00 | 12866 | 643.30 | 15945 | 531.50 | 19610 | 490.25 | 23627 | 472.54 | 79.54 |
| C1355 | 1008800 | 11520 | 1152.00 | 23040 | 1152.00 | 34560 | 1152.00 | 41472 | 1036.80 | 47232 | 944.64 | 127.51 |

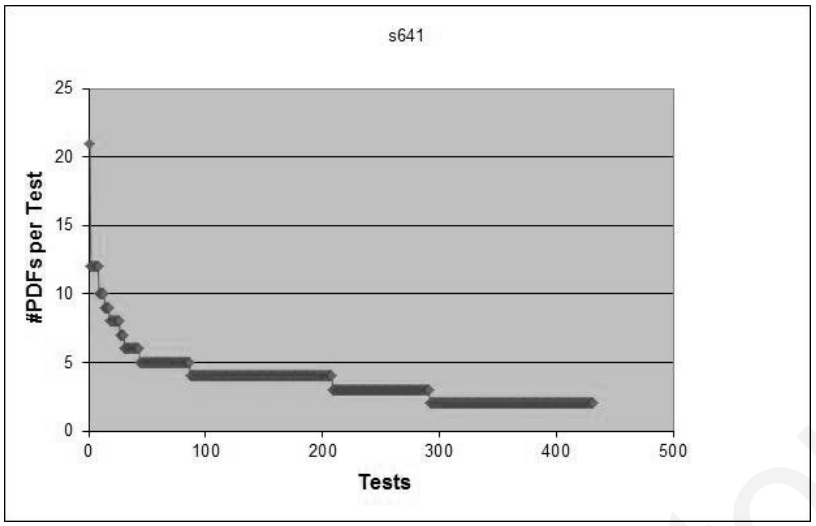
- denotes that 100% critical fault coverage is already achieved

of generated tests. For example, Column 3 (Ded) lists the number of critical PDFs detected and Column 4 lists their *AvgTE*, when up to the 10 tests are generated. Dividing Ded by *AvgTE* gives the exact number of tests generated in each case. We observe that the *AvgTE* is maintained in many occasions, i.e., additional tests with large TE are derived. These are very encouraging results since they demonstrate that very compact test sets can be derived by the proposed approach. Column 13 lists the *AvgTE* when all the critical PDFs listed in Column 2, are detected. The drop in *AvgTE* in this case is attributed to the fact that, after many PDFs are detected and removed from the ISOPs/ZBDD, the remaining critical PDFs are less likely to be tested by common tests. This is inherent to the circuit structure and not to the proposed method that considers all possible tests per PDF.

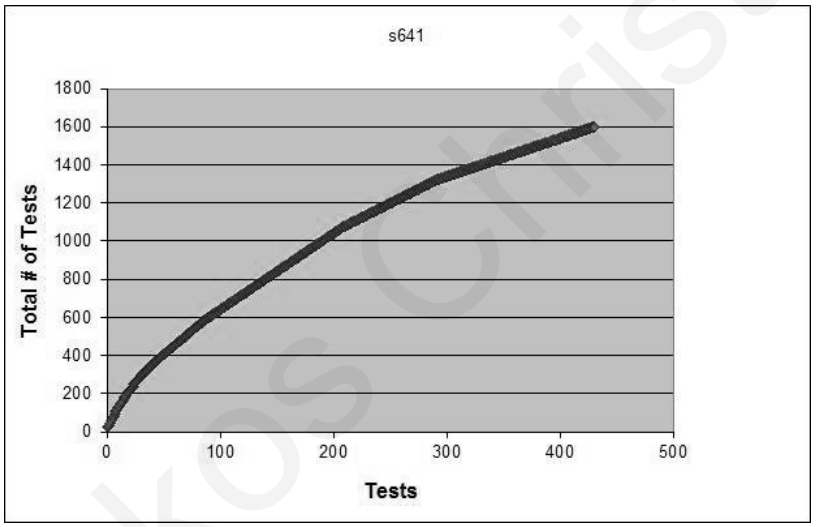
To our knowledge, no previous method that examines the proposed problem exists and, therefore, no comparison with previous work is possible. All existing methods concentrate either on the compactness constraint of the problem, but do not consider critical PDFs (such as [11], [47], [48], [74], [91], [90], [99]), or on critical PDF selection and test generation but do not consider the compactness issue (such as [85], [102], [114]). The methods of the later case enumerate the critical PDFs (even in the case of the recent method of [85], unsensitizable PDFs are identified non-enumeratevely but ATPG examines the critical PDFs explicitly) and, thus, the AvgTE of their generated test sets is bounded to be close to 1.

## 3.7 Conclusions

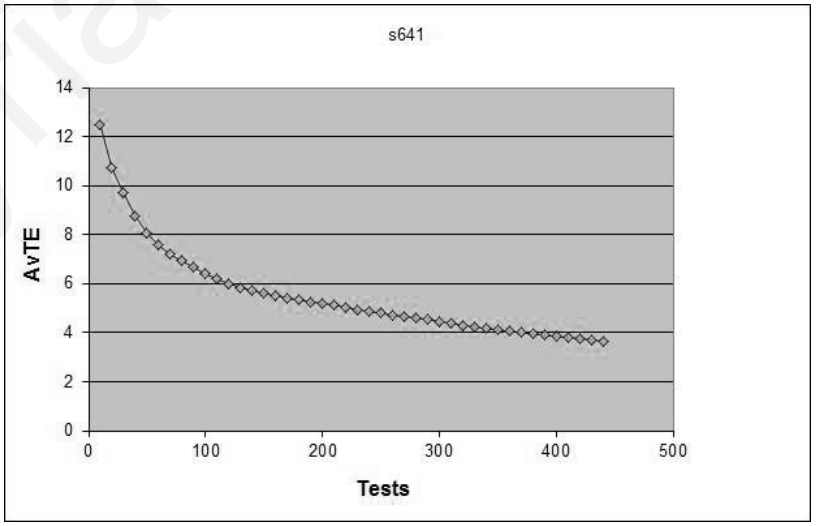
This work presents a test generation methodology for PDF tests with high test efficiency for critical PDFs. It is shown how tests with maximal test efficiency can be derived by linear manipulations of a canonical decision diagram that represents PDFs along with all their associated test patterns compactly. The method can apply to all circuit paths or to sets of circuit paths, such as the set of critical paths. The experimental results clearly demonstrate the practicality of the method and its superiority over existing methods in terms of high test efficiency for critical PDFs. Besides compact test generation, the proposed decision diagram can be important to a variety of other delay test-related and timing analysis problems.




(a) Number of PDFs per Test.

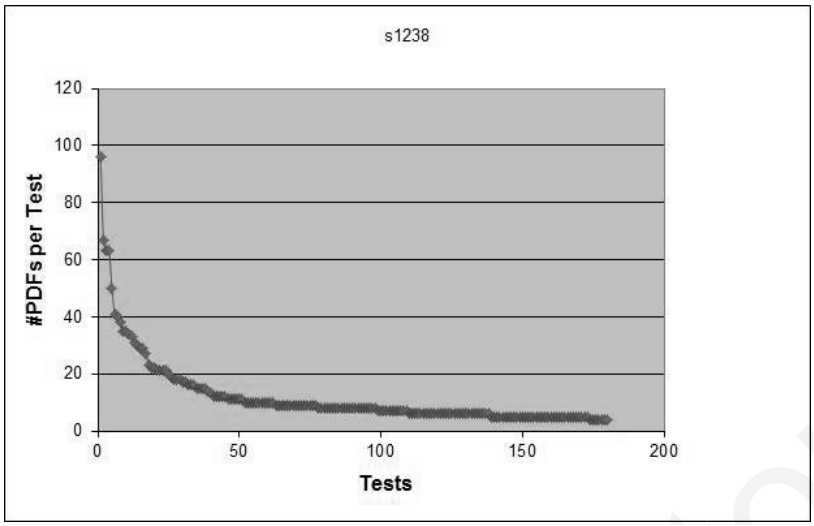



(b) TotalNumber of Tests.

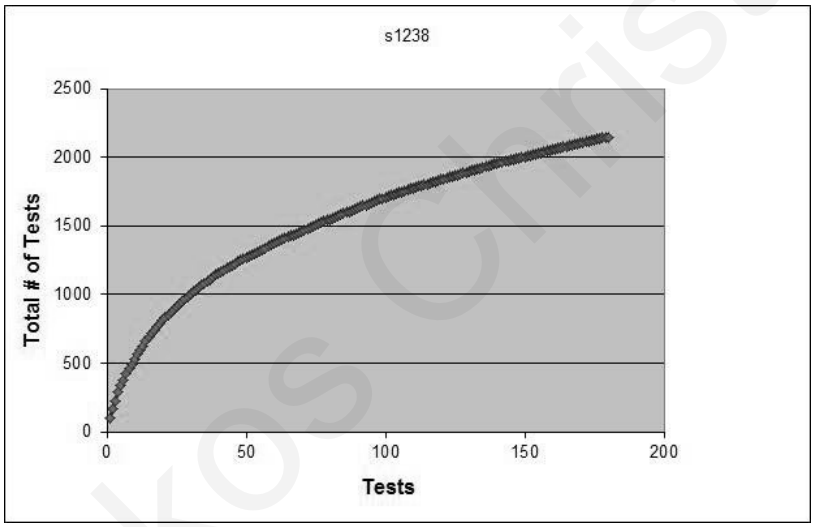



(c) Average Test Efficiency (AvTE).

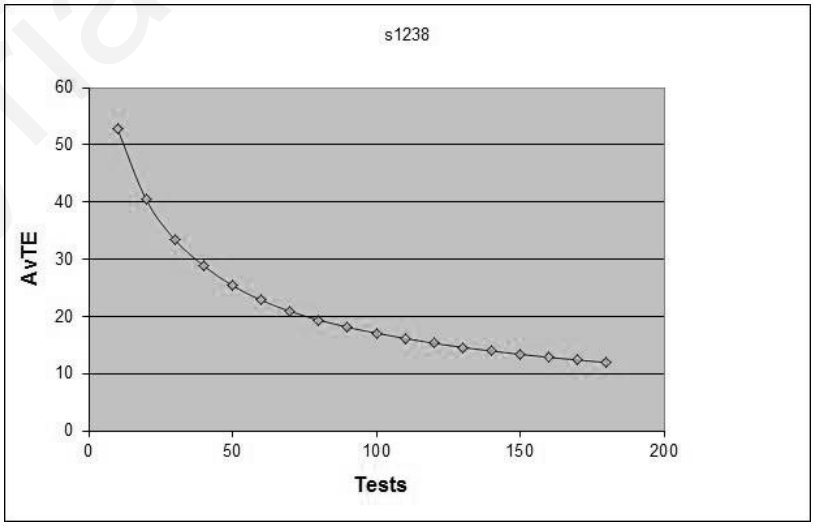
Figure 3.16: Statistical Information for Circuit S641.




(a) Number of PDFs per Test.

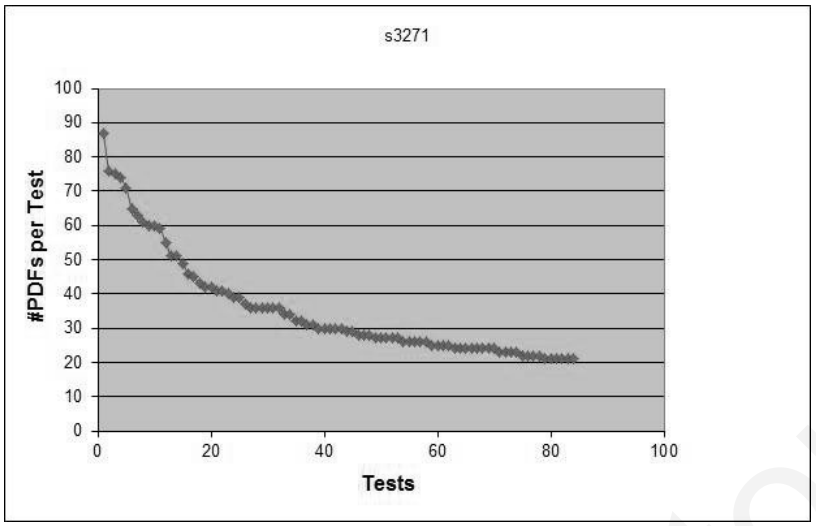



(b) TotalNumber of Tests.

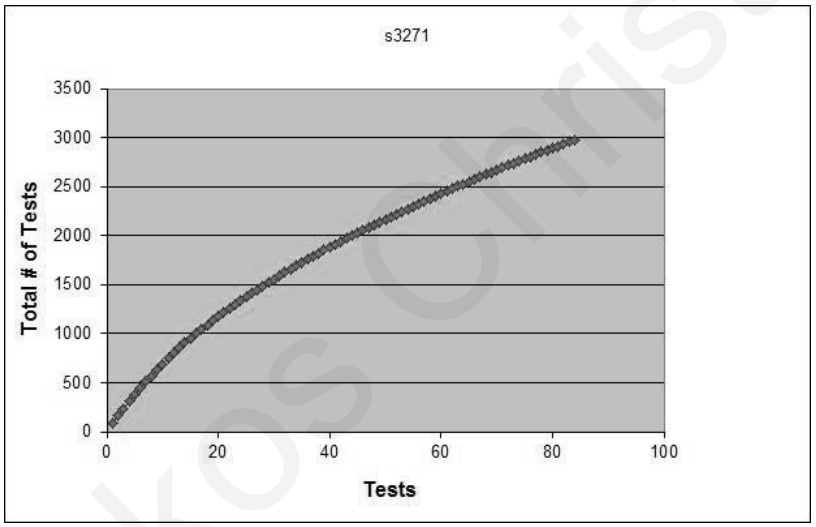



(c) Average Test Efficiency (AvTE).

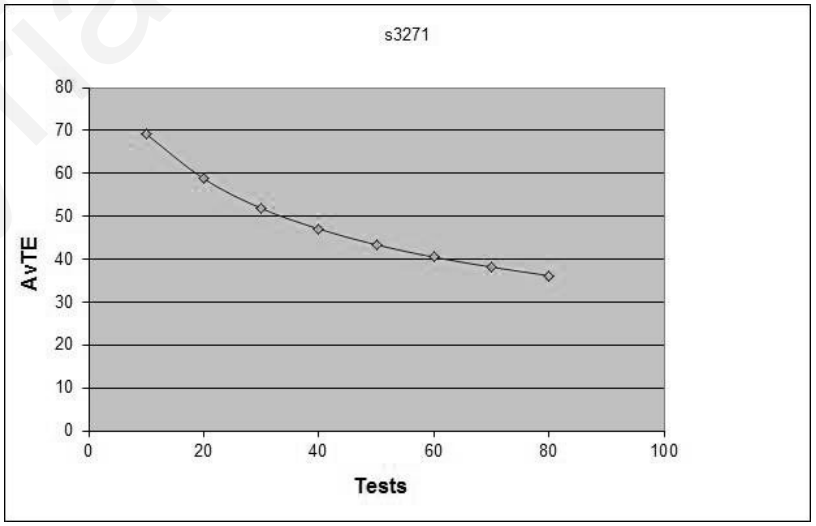
Figure 3.17: Statistical Information for Circuit S1238.




(a) Number of PDFs per Test.

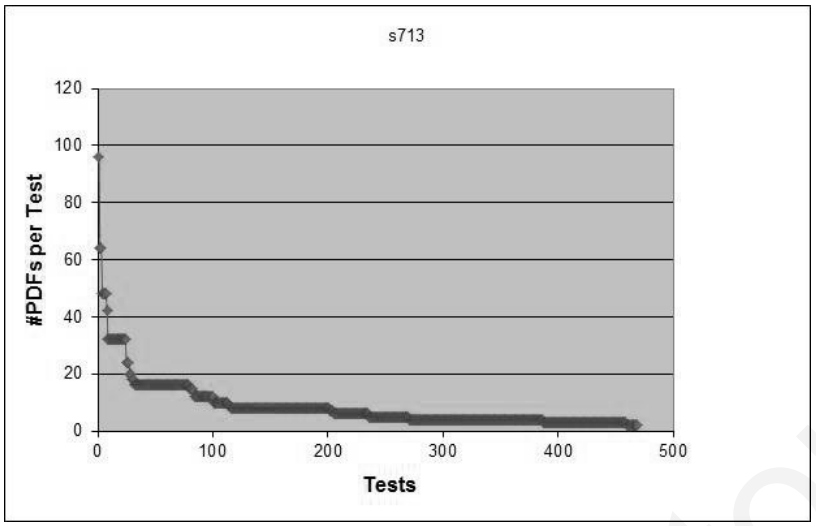



(b) TotalNumber of Tests.

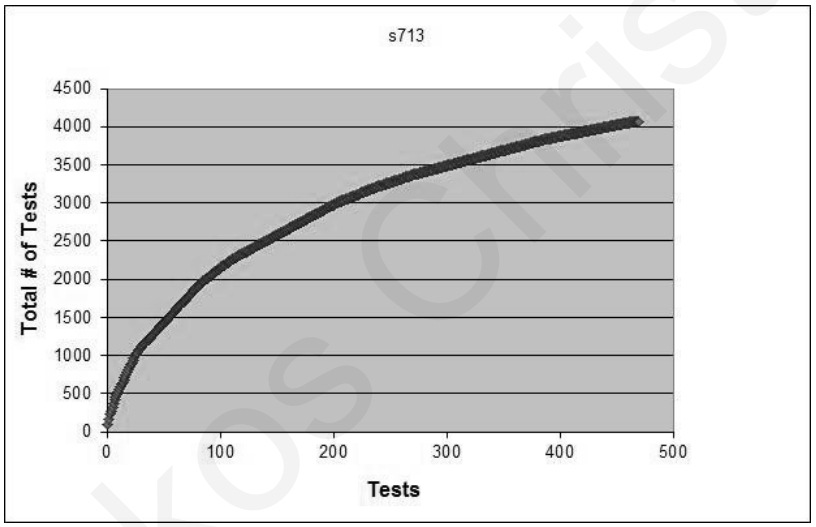



(c) Average Test Efficiency (AvTE).

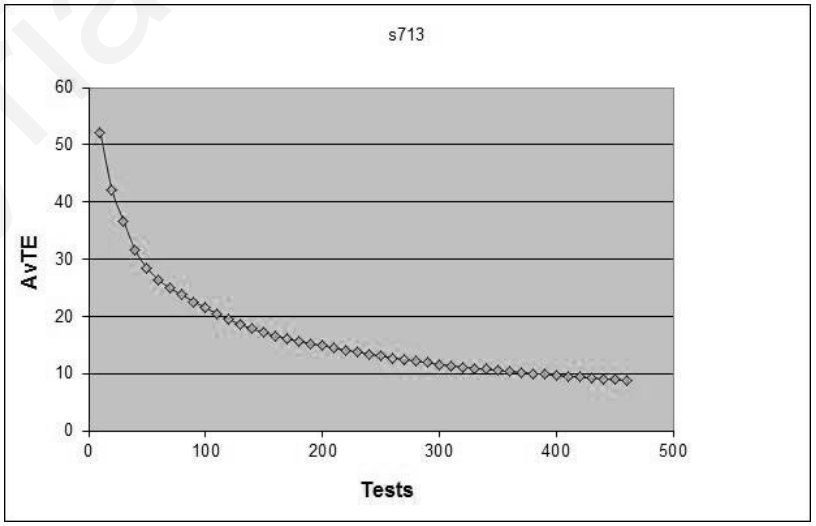
Figure 3.18: Statistical Information for Circuit S3271.




(a) Number of PDFs per Test.




(b) TotalNumber of Tests.




(c) Average Test Efficiency (AvTE).

Figure 3.19: Statistical Information for Circuit S713.

# Chapter 4

# Using of Decision Diagrams for Identifying Critical Primitive Path Delay Faults

## 4.1 Introduction

The Path Delay Fault (*PDF*) model has been long considered as the most accurate one among the various delay fault models, due to its ability to detect both lumped as well as small distributed delay defects [106]. The *PDF* set consists of single as well as multiple faults (a fault that is not singly detectable may be detected when tested as part of a multiple fault). A lot of effort has been devoted in the past on the quality of the generated tests for *PDFs*. Single *PDFs* are tested using robust tests, if these exist, otherwise non-robust tests are used. Multiple faults (also referred to as functionally sensitized faults in the literature) are tested using non-robust tests [21]. The number of functionally sensitized faults is usually intractable, as any number of paths can contribute to the formation of a multiple path. However, a significant number of functionally sensitizable *PDFs* does not have to be tested to ensure the temporal correctness of a circuit. It has been previously shown e.g., ( [51, 52, 56, 105]) that functionally sensitizable *PDFs* that can affect the performance of a circuit, must also be primitive faults. A multiple *PDF* is classified as a multiple primitive fault if the multiple *PDF* is static sensitizable and no proper subset of this multiple *PDF* is static sensitizable. The *primitive PDF* model, which is a refinement of the traditional *PDF* model, limits significantly the number of multiple faults needed to be tested (a non-primitive fault is always covered by a corresponding primitive one) and, besides in delay testing, it has also been

shown to apply in timing verification and timing analysis.

A major problem faced when considering any of these two path-dependent fault models is their enormous number of faults (paths), which in the worst case grows exponentially to the number of lines in the circuit. Restricting the problem to the critical path set, reduces considerably the number of faults considered and, hence, makes the problem more practical. A critical path is one with large delay, enough to affect the timing correctness of the circuit if it contains a delay defect. Various models have been used to derive the critical path set, including both deterministic, such as the fixed and bounded delay models, as well as statistical models [64, 65, 102, 113, 114].

This work considers the problem of finding the testable critical primitive *PDF* set in combinational or enhanced fully-scanned sequential circuits. Even though the problem of identifying testable critical *PDFs* has been extensively addressed in the literature, see [25, 64–66, 85, 114] among many others, none of these methods considers primitive *PDFs* explicitly. This work is the first to define and identify critical primitive *PDFs*, integrating both aspects of criticality and primitivity in a common framework. A major challenge in such a problem is the large number of paths needed to be considered in order to identify primitive faults, even when the problem is restricted to a small number of critical faults. It is important to note that the work here is independent from the critical *PDF* selection phase; any critical *PDF* set selection algorithm and delay model can be applied to derive the set of single critical *PDFs* which is used as input to our methodology.

The proposed methodology utilizes function-based formulations with appropriate data structures (Zero-Suppressed Binary Decision Diagrams - ZBDDs) for implicit and compact representation of paths such that the targeted set of faults is identified in a non-enumerative manner (no path, path-segment, or fault is ever enumerated). ZBDDs have been previously proposed for the simpler version of this problem for traditional (critical) *PDFs* [25, 85] however, none of these methods can be trivially extended to primitive faults since the standard ZBDD operators they utilize cannot handle multiple faults. This work presents new operators, polynomial to the size of the ZBDD, for efficient and non-enumerative manipulation of multiple faults which are necessary for identifying all critical primitive faults. The major contributions of this work are summarized below:

(i) testable critical primitive *PDFs* are defined,

(ii) the targeted faults are identified efficiently, avoiding enumeration of faults or paths which can be prohibitive for large circuits,

(iii) any delay model for identifying the potentially testable critical (single) *PDF* set can

be considered,

(iv) the generated data structure that represents the targeted faults also contains ATPG data (all tests per identified testable fault) and, hence, the proposed method can be easily incorporated in a very efficient ATPG framework, since the necessary tests are already generated (ATPG is not the main focus of this work, however, we do report some indicative ATPG results), and

(v) the reported experimental results show that only a small number of multiple primitive *PDFs* is testable (when compared to the set of single primitive *PDFs*), implying that a small number of additional tests suffices to guarantee the circuit's timing correctness under the multiple fault criterion.

The rest of this chapter is organized as follows: Section 4.2 presents necessary definitions and preliminaries concepts. Section 3 describes the proposed algorithm for identifying critical primitive *PDFs*; Subsection 4.3.1 presents the overall approach, subsection 4.3.2 concentrates on multiple critical *PDFs* and subsection 4.3.3 gives the detailed function-based formulation based on standard ZBDD operators and the newly proposed operators to handle multiple paths. Subsection 4.3.4 explains how the primitivity property is ensured in the proposed framework and, finally, subsection 4.3.5 presents in detail the new operators, along with some illustrative examples. Section 4.4 presents and discusses the obtained experimental results, and section 4.5 concludes the chapter.

## 4.2 Preliminaries and Notation

### 4.2.1 *PDF* Classification

Under the *PDF* model, a fault is a sequence of Falling ($F/\downarrow$) or Rising ($R/\uparrow$) transitions along a physical path from a primary input (*pi*) to a primary output (*po*), tested by a pair of input vectors ($v_1$, $v_2$).

The set of *PDFs* can be partitioned into two disjointed subsets, the *PDFs* that can be sensitized under certain sensitization conditions, and the *PDFs* that cannot be sensitized under any condition [21]. *PDFs* that cannot be sensitized under any condition have no effect on the timing of the circuit and are known as functionally unsensitizable, or functionally redundant, *PDFs*.

The set of functionally sesnsitazable *PDFs* includes both single and multiple *PDFs*. Single *PDFs* are those that can be tested with non-robust (or robust) tests. Multiple *PDFs*

require co-sensitization as they can only be detected if multiple faults exists [21]. According to [52], all single *PDFs* are primitive. However, only a small subset of the multiple *PDFs* are primitive and need to be tested. Let $\pi_i$ denote a primitive *PDF* of cardinality *i*. Hence, $\pi_1$ denotes a single primitive *PDF*. A primitive *PDF* of cardinality *i* ($\pi_i$) consists of exactly *i* single *PDFs* that terminate at the same *po*, none of which is part of any other primitive *PDF* of cardinality less than *i* [52]. Let $\Pi_i$ denote the set of all primitive *PDFs* of cardinality *i*. Then $\Pi_1$ is the single primitive *PDF* set, $\Pi_2$ is the set of all primitive *PDFs* of cardinality 2, and so on. We define $\Pi$ to be the set of all primitive *PDFs*, $\Pi = \Pi_1 \cup \Pi_2 \cup \cdots \Pi_n$, where *n* is the maximum cardinality index. Hence, the primitive *PDF* set can be categorized into the Single *PDF* (*SPDF*) set, which is $\Pi_1$, and the Multiple *PDF* (*MPDF*) set, which is $\Pi - \Pi_1$.

### 4.2.2 Critical Primitive *PDF*

Let $\mathscr{C}$ denote the set of the potentially testable critical single *PDFs*. $\mathscr{C}$ contains the single *PDFs* that are part of the critical set and may or may not be testable under the primitive model. $\Pi^{\mathscr{C}}$ denotes the set of all critical primitive *PDFs*, $\Pi_1^{\mathscr{C}}$ the set of all critical *SPDFs*, and $\Pi^{\mathscr{C}} - \Pi_1^{\mathscr{C}}$ the set of all critical primitive *MPDFs*. Furthermore we define $S_i^{\mathscr{C}}$ to be the set of single *PDFs* that have contributed in $\Pi_i^{\mathscr{C}}$. $S_1^{\mathscr{C}}$ is exactly the $\Pi_1^{\mathscr{C}}$ set.

The critical primitive *PDF* set $\Pi^{\mathscr{C}}$ has the following characteristics:

(i) $\Pi_1^{\mathscr{C}} \subseteq \mathscr{C}$, where $\mathscr{C} \backslash \Pi_1^{\mathscr{C}}$ is the set of single non-robustly untestable *PDFs*,

(ii) Any multiple critical primitive fault $\pi_i \in \Pi_i^{\mathscr{C}}$ consists only of single non-robustly untestable *PDFs* in $\mathscr{C} \backslash (S_1^{\mathscr{C}} \cup S_2^{\mathscr{C}} \cup \cdots \cup S_{i-1}^{\mathscr{C}})$.

For example, any fault $\pi_2 \in \Pi_2^{\mathscr{C}}$ will consist of 2 *PDFs* in $\mathscr{C}$, each of which is not sensitizable under the single fault criterion (and, hence, is contained in $\mathscr{C} \backslash \Pi_1^{\mathscr{C}}$). This ensures the primitivity property. Moreover, in order to maintain the criticality property, all single *PDFs* composing a multiple *PDF* $\pi_i \in \Pi_i^{\mathscr{C}}$ must be critical (belong to $\mathscr{C}$) since the fastest transition in a multiple fault is the one determining the overall delay of the fault.

As an example, consider the circuit of Fig. 4.2.2. Let the critical paths in $\mathscr{C} = \{\downarrow b.f.g.i, \downarrow b.f.h.i, \downarrow b.d.g.i\}$. Observe that none of these *PDFs* is singly sensitizable (non-robustly testable), hence, $\Pi_1^{\mathscr{C}} = \emptyset$. There are 3 possibly testable critical primitive *PDFs* of cardinality 2 (all the pairwise combinations in $\mathscr{C}$). Fig. 4.1 and 4.2 show 2 out of the 3 possible faults (shown in bold lines), which are both primitive (as no constituent single *PDF* belongs in $\Pi_1^{\mathscr{C}}$) and critical (as each constituent *PDF* belongs in $\mathscr{C}$). Observe that the sensitization criterion in multiple *PDFs* allows for a don't care bit in $v_1(x)$ and requires a non-controlling value for

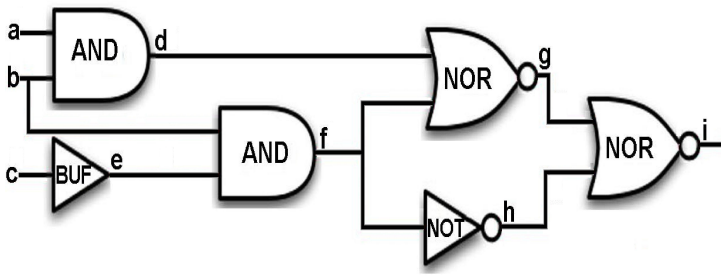



Figure 4.1: Example circuit $C$ with $\mathscr{C} = \{\downarrow b.f.g.i, \downarrow b.f.h.i, \downarrow b.d.g.i\}$

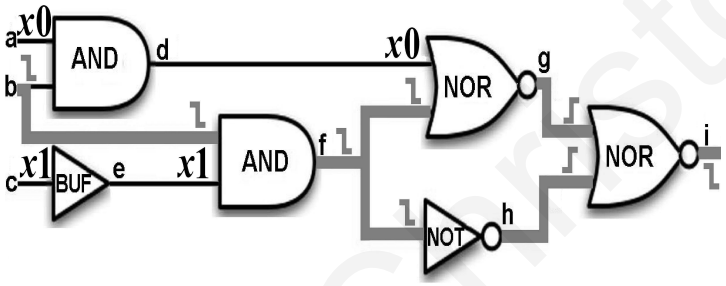



Figure 4.2: A primitive critical *PDF* of cardinality 2 ($\Pi_2^{\mathscr{C}}$)

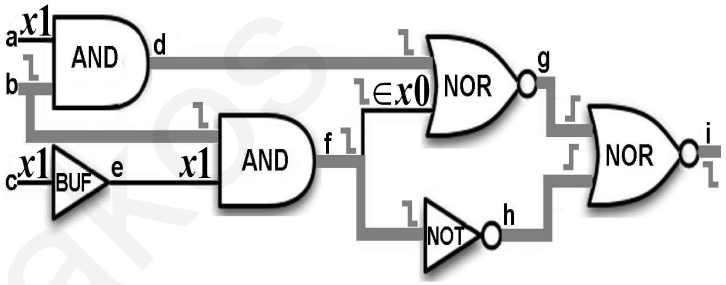



Figure 4.3: Another primitive critical *PDF* of cardinality 2 ($\Pi_2^{\mathscr{C}}$)

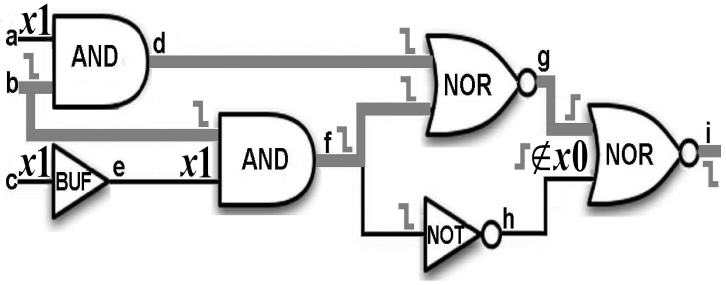



Figure 4.4: A functionally unsensitizable multiple *PDF* of cardinality 2

$v_2$ (0 for the NOR gate and 1 for the AND gate). Hence, the third possible fault, shown in Fig. 4.3, is redundant as line $h$ can not take any value in $x0$ ($\downarrow$ or stable at 0). As $S_2^{\mathscr{C}} = \mathscr{C}$, we know that no other critical primitive *PDF* of cardinality greater than 2 exists. Therefore, for this circuit and the given set of potentially testable critical *PDFs*, only 2=$|\Pi_1^{\mathscr{C}} \cup \Pi_2^{\mathscr{C}}|$ faults need to be tested.

### 4.2.3 Function Notation, ZBDDs and ISOP/ZBDD Graph

A *PDF* set can be encoded as a combinatorial set which in turn can be efficiently represented using ZBDDs, a variant of BDDs, where the absence of a variable is interpreted as a zero assignment. ZBDDs have been successfully used in representing a huge number of *PDFs* compactly in a non-enumerative fashion [84]. ZBDDs can also store boolean functions by introducing additional variables. For each variable in the function, two variables are used such that when both variables are suppressed, the original variable has a don't care value. Such a structure is called a ZBDD-based representation of an Irreduntant Sum-of-Products (ISOPs) [77]. The work in [25] has proposed a hybrid ISOP/ZBDD-based data structure capable of implicitly representing all sensitizable *PDFs* in a circuit, for various path sensitization types (robust, non-robust, functional sensitizable [21]), that requires only a polynomial number of standard ZBDD operations. This method also restricted the targeted *PDF* set to the critical paths. The methodology proposed in this chapter, in contrast to the work in [25], can consider primitive multiple faults and, hence, can drastically reduce the number of multiple faults needed to be tested in order to guarantee the circuit's timing correctness. Handling primitive faults with ZBDDs is a considerable challenge, not examined by previous work and requires new ZBDD operators, as it will be discussed in the next section.

The remaining of this subsection gives some preliminary concepts and notation for *PDF* representation using ZBDDs as well as the hybrid ISOP/ZBDD-based structure used in this work, via a simple example. Necessary notation and definitions will follow. The reader is refereed to [25, 84] for extended details.

Two sets of variables are defined, the *test* variables which encode the test cubes and the *path* variables which encode the *PDFs*. Let $\mathscr{P}$ define the set of path variables, $\mathscr{T}$ the set of test variables, *PI* the set of primary input lines, and *L* the set of internal circuit lines (other than fanout branches, including primary output lines). There are two path variables per line in $PI \cup L$, corresponding to *R* and *F* transitions. Hence $|\mathscr{P}| = 2 * |(PI \cup L)|$. For the test variables, we use 2 variables per primary input (as it will be explained a bit further), hence

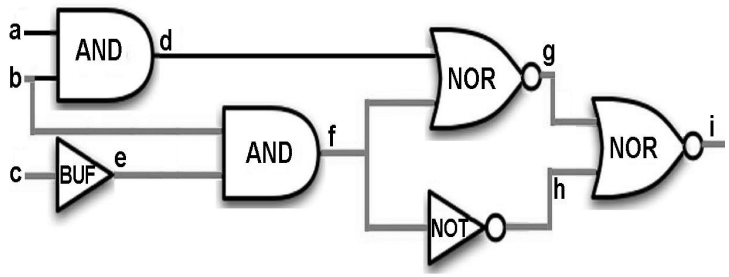



Figure 4.5: Example circuit $C$; critical paths shown in bold

$\mathscr{T} = 2 * |PI|$.

Consider the circuit of Fig. 4.5, which has 3 primary inputs, $PI = \{a, b, c\}$, and 6 lines in $L = \{d, e, f, g, h, i\}$. The ZBDD for all single $PDFs$ of $C$ requires 18 path variables, denoted by $\mathscr{P} = \{aR,aF,bR, \cdots ,iR,iF\}$, and is shown in Fig.4.6 with variable ordering $aR<aF<bR<bF< \cdots <iR<iF$. The variable ordering follows the topological order of the lines in the circuit. A $PDF$ is encoded by a combination over the variables of $\mathscr{P}$. Missing path variables assume a 0 value. Observe that there are exactly 12 routes from the root node to the terminal one node (**1**) that correspond to the 12 $PDFs$ of the circuit.

A ZBDD-based ISOPs representation is a compact and implicit cube set representation [77, 79, 96]. A ZBDD-based ISOPs involves defining two variables $i_0$ and $i_1$ per $i$ variable in the BDD, such that $i = i_1 \cdot \overline{i_0}$ and $\overline{i} = \overline{i_1} \cdot i_0$. Combination $\overline{i_1} \cdot \overline{i_0}$ implies that $i = x$ (don't care) and $i_1 \cdot i_0$ can not appear based on this encoding.As non-robust sensitization conditions are considered for primitive $PDFs$, [56, 105], it suffices to use only one variable per primary input in the BDD, the one corresponding to the sensitization condition of vector $v_2$. The value of $v_1$ is always don't care under the non-robust criterion, except from the primary input initiating the $PDF$ which must assume a $R$ or $F$ value. The later is encoded by the path variable in our representation, as it will become more clear in the next example (Fig. 4.7). As BDDs are transformed to ZBDD-based ISOPs in this work, the total number of test variables used is $|\mathscr{T}| = 2 * |PI|$. The ISOP/ZBDD-based tree of Fig. 4.7 represents all the singly testable $PDFs$ (subset of those of Fig. 4.6), along with all of the possible test cubes per $PDF$. Observe that the variables in $\mathscr{T} = \{a1,a0,b1,b0,c1,c0\}$, are placed on top of the variable ordering (the root of Fig. 4.7 starts with test variables, followed by the path variables). Every path from the root to the **1** node of Fig. 4.7 represents a single non-robustly testable $PDF$ along with some of its test cubes. For example, the path $a_1.b_1.\overline{c_1}.c_0.cF.eF.fF.hR.iF$ indicates that the falling $PDF$ on path $c.e.f.h.i$ can be tested by $(v_1,v_2)$=($xx1$, $110$). This is derived

as follows: the sub-combination $a_1.b_1.\overline{c_1}.c_0$ encodes values for $v_2$, such that $a = a_1.\overline{a_0} = 1$, $b = b_1.\overline{b_0}$=*1* and $c = \overline{c_1}.c_0$=*0*. Observe that as this is a ZBDD-based representation, any missing variables from the encoding (suppressed ZBDD nodes), such as $a_0$ and $b_0$, always assume the zero(0) value. Hence, $v_2 = 110$. For $v_1$, all variables (primary inputs) can take the don't care value , except from $c$ which must have a 1 value as indicated by variable $cF$ in the *PDF* encoding ($cF.eF.fF.hR.iF$). Hence, ($v_1$,$v_2$)=(*xx1, 110*). Observe that, for this particular case, $a_1.b_1.\overline{c_1}.c_0$ encodes the entire test space for the *PDF*, as $cF.eF.fF.hR.iF$ is not contained in any other path from the root to the terminal **1** node of the graph. The ZBDD in Fig. 4.8 contains only the singly testable *PDFs* of *C* (5 *PDFs*), with no test cube information, and is derived from Fig. 4.7 after existentially abstracting all test variables.

Standard ZBDD operations that are used by the proposed method can be seen in Table 4.1. Furthermore, three new, polynomial ZBDD operators are introduced in this work, which are used in combination with the standard ZBBD operators in order to derive the critical primitive path set in an implicit and non-enumerative manner. These new operators have a '*' in front of their description in Table 4.1. Namely these are New-Intersect, New-Product and New-Segment. New-Intersect performs intersection between the elements of two sets, starting from a given variable index and assuming that the second argument of the operator does not contain variables of smaller index than the specified one ($n$). A variable index $n$ refers to the level in the graph of the nodes of a particular variable. In decision diagrams (such as ZBDDs) a variable can have multiple nodes in the graph (see for example variable *bR* in Fig. 4.7). Furthermore, as these are ordered graphs, the nodes are levelized. The node at the root (corresponding to the first variable) has level 0, the nodes of the next variable in the order have level 1, and so on. Hence, for a variable order $a<b<c$ the corresponding variable indexes are *0<1<2*. Any node of variable $b$ ($c$) has index *1 (2)*. New-Product computes the binate product of 2 sets up to a specified variable index and then computes the unate product for the remaining elements in the sets. New-Segment is a single argument operation that returns the ZBDD segments, of the given set, starting from the specified index.

Multiple primitive *PDF* identification requires additional effort than that required for single primitive *PDF* identification. This derives from the multiple *PDF* representation as a single route in the ZBDD and the critical primitive conditions for the multiple *PDF* imposed. The need of new ZBDD operations arises from the logical representation used, both for the *PDF* and the functionality, and the conditions imposed by the problem examined. The exact methodology and a detailed description of the new ZBDD operators are given in the following section.

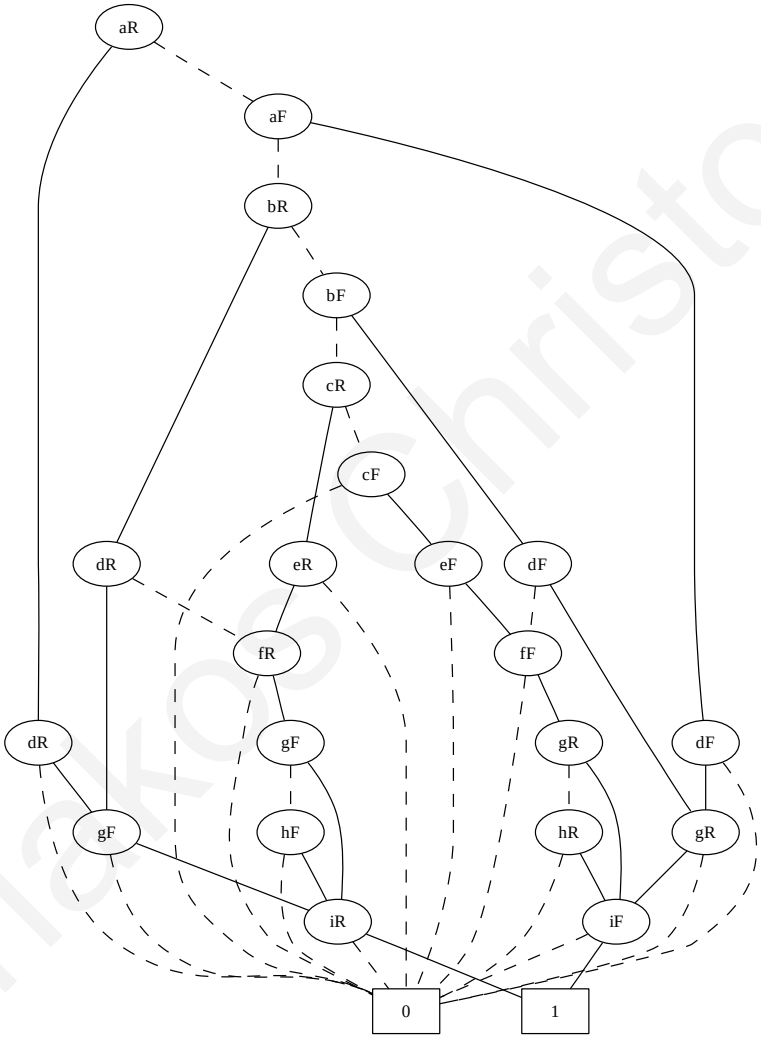



Figure 4.6: ZBDD for all single *PDFs* in Circuit *C*

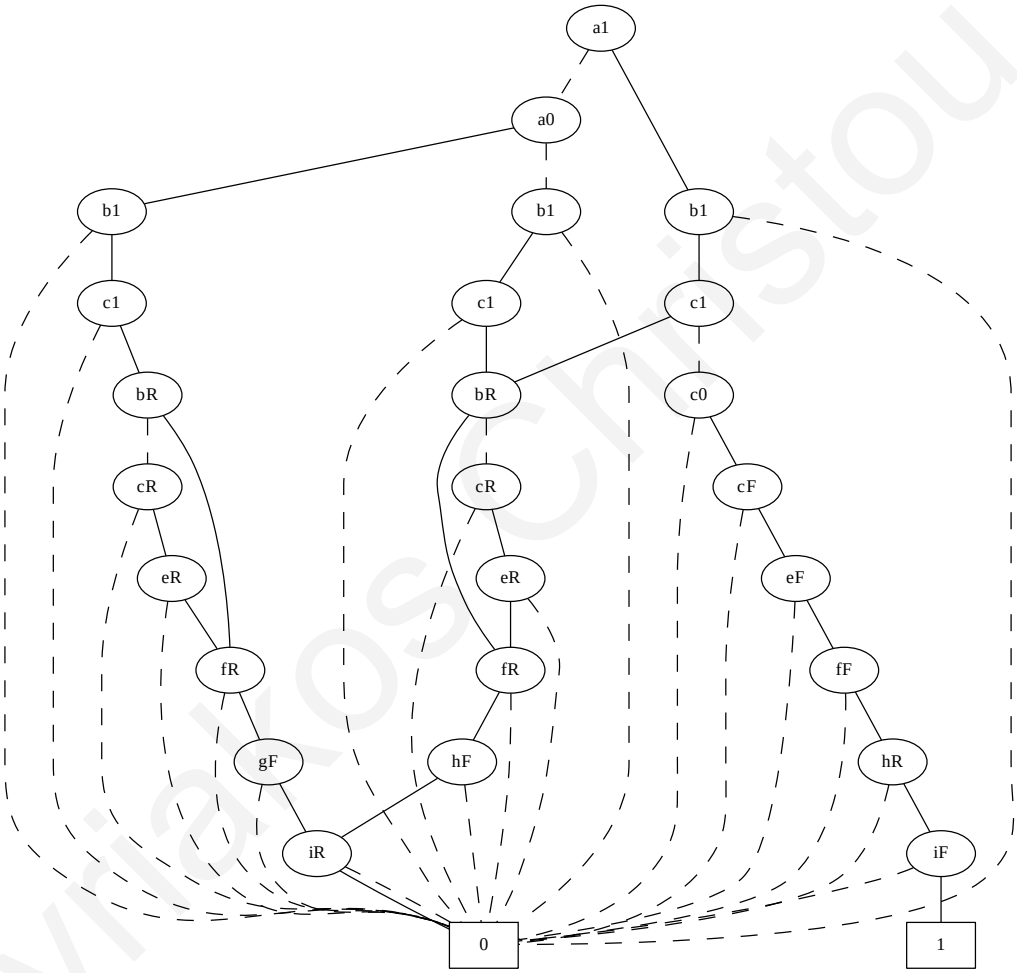



Figure 4.7: ISOP/ZBDD Graph for all single testable *PDFs* of Circuit *C*

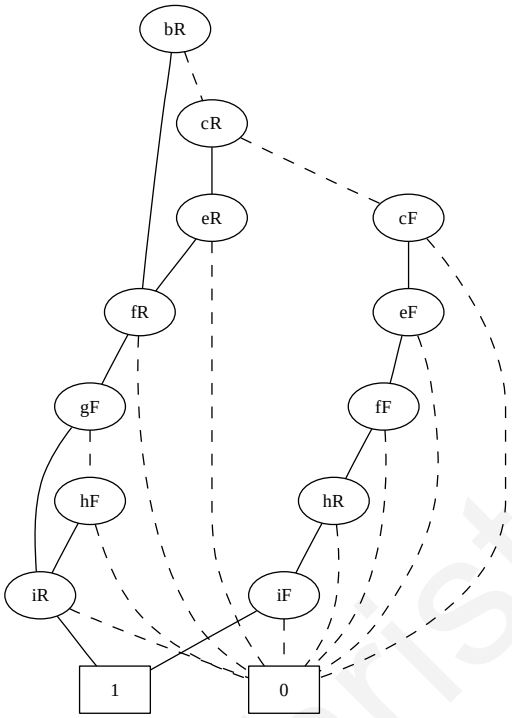



Figure 4.8: ZBDD for all single testable *PDFs* of Circuit *C*

Table 4.1: Standard/*NEW** ZBDD Operators used

| **Expression** | **Description** |
|---|---|
| $!(P,v)$ | Complement value of variable $v$ in $P$ |
| $P \cup Q$ | Union of $P$ and $Q$ |
| $P \cap Q$ | Intersection of $P$ and $Q$ |
| $P \times Q$ | Pairwise Intersection of $P$ and $Q$ |
| $P \bullet Q$ | Binate Product of $P$ and $Q$ |
| $P \backslash Q$ | Subtract $Q$ from $P$ ($P-Q$) |
| $\exists(P,v)$ | Existentially abstract variable $v$ from $P$ |
| $\|P\|$ | Cardinality of $P$ (number of paths to terminal **1**) |
| $P_{v_1}$ | Subset of $P$ such that variable $v=1$ |
| $P \cap_n Q$ | *Intersection of $P$ and $Q$ starting at variable index $n$ |
| $P \star_n Q$ | *Binate Product of $P$ and $Q$ until index $n$ and Unate Product of $P$ and $Q$ after index $n$ |
| $\wedge_n P$ | *Segments of $P$, starting from index $n$ |

## 4.3 Finding the Critical Primitive *PDF* Set

In this Section a methodology/algorithm for finding the critical primitive *PDF* set is given.

### 4.3.1 General Methodology

Let $\mathscr{C}$ denote the potentially testable critical *PDF* set. This set is an input to our methodology and, hence, can be derived using any of the state-of-the-art techniques for selecting critical paths. The set is encoded in ZBDD format using the method of [84].

The proposed methodology takes as input the netlist $C$ of the circuit under consideration and $\mathscr{C}$, and firstly applies the method of [25] to derive all the single non-robustly critical testable *PDFs*. This set is the single primitive set denoted by $\Pi_1^{\mathscr{C}}$. Then, it proceeds in an iterative manner to find multiple primitive critical *PDFs* of cardinality $i = 2, 3, ....$ The iterative process terminates when no more testable faults exist.

The basic steps of the method are given in Algorithm 4.1. The single critical primitive *PDF* set ($\Pi_1^{\mathscr{C}}$), is calculated in step 1. Consequently, step 2 removes all the singly testable faults from the critical path set $\mathscr{C}$, using a standard ZBDD set difference operation. Next, the algorithm proceeds in an iterative fashion to find the critical primitive *MPDF* set of cardinality $i$ ($1 < i$), per iteration (line 5). Every time a $\Pi_i^{\mathscr{C}}$ set is calculated, the set of potentially critical faults $\mathscr{C}$ is updated such that it only contains single *PDFs* that have not yet contributed in any of the primitive faults of cardinality equal or less than $i$. This is achieved by first finding the single *PDFs* in $\mathscr{C}$ that are a part of the multiple faults in $\Pi_i^{\mathscr{C}}$, denoted by $S_i^{\mathscr{C}}$ in Algorithm 4.1 (line 7), and then removing them from $\mathscr{C}$ (line 8). As a result, the critical primitive *MPDF* set $\Pi_{i+1}^{\mathscr{C}}$ calculated in the next iteration will maintain the primitivity property. The iterations terminate when the number of single *PDFs* that have remained in $\mathscr{C}$ is less than $i$, which means that no multiple critical primitive faults of cardinality $i$ exist. If $\mathscr{C}$ is the empty set at the end of the iterations it means that all the critical *PDFs* originally in $\mathscr{C}$ are testable under the primitive criterion. Otherwise, any *PDF* left in $\mathscr{C}$ is proven untestable (under the single and multiple fault criterion).

Next we give a simple example to outline the basic steps of the proposed methodology, followed by a more detailed description on how the multiple faults are derived in 4.3.2. Consider again the circuit in Fig. 4.5 and the input set $\mathscr{C}$ of potentially critical testable faults of Fig. 4.9 which contains 8 *PDFs* (indicated by the red/bold lines in Fig. 4.5). Only 5 out of the 8 faults in $\mathscr{C}$ are singly testable (see Fig. 4.8), as identified at step 1 of Algorithm 4.1.

**Data**: $C$ : Circuit;

$\mathscr{C}$ : Potentially testable critical single $PDFs$

**Result**: $\Pi^{\mathscr{C}}$: Critical Primitive $PDF$ Set

1 $\Pi_1^{\mathscr{C}} \leftarrow$ Single_Critical_Primitive_Set($C$,$\mathscr{C}$);

2 $\mathscr{C} \leftarrow \mathscr{C} \backslash \Pi_1^{\mathscr{C}}$;

3 $i \leftarrow 2$;

4 **while** *(*$|\mathscr{C}| \geq i$*)* **do**

5 $\quad \Pi_i^{\mathscr{C}} \leftarrow$ Multiple_Critical_Primitive_Set ($C$,$\mathscr{C}$,$i$);

6 $\quad \Pi^{\mathscr{C}} \leftarrow \Pi^{\mathscr{C}} \cup \Pi_i^{\mathscr{C}}$;

7 $\quad S_i^{\mathscr{C}} \leftarrow$ Extract_Contributing_Single_$PDFs$($\Pi_i^{\mathscr{C}}$,$\mathscr{C}$);

8 $\quad \mathscr{C} \leftarrow \mathscr{C} \backslash S_i^{\mathscr{C}}$;

9 $\quad i \leftarrow i+1$;

10 **end**

11 $\Pi^{\mathscr{C}} \leftarrow \Pi^{\mathscr{C}} \cup \Pi_1^{\mathscr{C}}$;

**Algorithm 4.1:** Compute Critical Primitive $PDF$ Set $\Pi^{\mathscr{C}}$

Hence, the updated $\mathscr{C}$ at step 2 will contain only the 3 singly untestable faults $\{$*bF.fF.gR.iF, bF.fF.hR.iF, cF.eF.fF.gR.iF*$\}$, as shown in Fig. 4.10. At this point, since $|\mathscr{C}| > i$ $(3 > 2)$, step 5 will be executed to derive a critical primitive $MPDF$ set of cardinality 2 ($\Pi_2^{\mathscr{C}}$), which as shown in Fig. 4.11 contains only one such multiple fault $\{$*bF.fF.gR.hR.iF*$\}$. Observe that this $MPDF$ consists of 2 out of 3 singly untestable critical $PDFs$ of Fig. 4.10, which are $\{$*bF.fF.gR.iF, bF.fF.hR.iF*$\}$. Step 7 will extract these 2 single paths from $\Pi_2^{\mathscr{C}}$ and step 8 will remove them from $\mathscr{C}$. As a result only 1 potentially testable critical single fault will remain in $\mathscr{C}$, (shown in Fig. 4.12) and, therefore, the algorithm will not enter into the next iteration. Hence for this example, $\Pi^{\mathscr{C}}$ contains 6 faults (5 $SPDFs$ and 1 $MPDF$). Moreover, fault $\{$*cF.eF.fF.gF.iF*$\}$ does not need to be tested according to the primitivity conditions.

Step 1 of the Algorithm 4.1, is of polynomial complexity as shown in Chapter 3. Step 2 is a standard ZBDD operation. The complexity of Algorithm 4.1 relies on the number of iterations used. At each iteration, Algorithm 4.1 tries to find a higher cardinality multiple PDF. The complexity of Algorithm 4.1, will be studied assuming that a reasonable number of executed iterations is 9 (in order to maintain reasonable execution times), that is examining multiple PDFs of cardinality 9, and that a reasonable number of fanins at a gate is 16 (considering the structure of modern circuits this a a fair assumption). All the steps inside

the while loop, expect steps 5 and 7 are standard ZBDD operations. Section 4.3.4, step 7, finds the composing single *PDFs* of a multiple *PDF* in a linear number of ZBDD operators of polynomial complexity to the input IZOP/ZBDD graph. Thus, the complexity of Algorithm 4.1 relies in step 5. This step computes combinations of PDFs and executes a linear number of polynomial ZBDD operations, as it will be seen in Algorithms 4.2 and 4.3. The number of possible combinations on a 16 fan-in gate is linear, $\binom{16}{1}+\binom{16}{1}+\cdots+\binom{16}{16}$. Thus, Algorithm 4.1 executes 9 circuit traversals with a linear number of polynomial operations on each visited line. These assumptions are fair if the complexity of the problem is taken into account. The number of multiple PDFs is double exponential in the worst case. If the number of PDFs is $w = |PDFs|$, then $w$ is exponential with respect the number of circuit lines. The total number of multiple PDFs considered is $\binom{w}{1}+\binom{w}{2}+\cdots+\binom{w}{w-1}+\binom{w}{w}$.

The necessity of steps 7 and 8 derives from the encoding that the *MPDFs* follow. As it can be seen through this example finding the composing single *PDFs* of a multiple *PDF* is not straightforward. Finding the contributing single *PDFs* is an important step since it has to do with the correctness of the methodology used. The steps for solving this issue are explained in detailed later. The primitivity property is preserved at steps 2, 7 and 8 of Algorithm 4.1, by removing the already detected *SPDFs* from $\mathscr{C}$. Step 2 of Algorithm 4.1 is a standard set difference ZBDD operation, since it involves removing already detected *SPDFs*. Step 7 and 8 involves *MPDFs* and is treated differently since $\mathscr{C}$ consists of *SPDFs*. Step 7 finds all the *SPDFs* that contributed to the *MPDF* set and then step 8, a standard set difference ZBDD operation, follows to remove all these identified *SPDFs* that are a part of the identified *MPDF* set, from $\mathscr{C}$. These two steps are a necessity for the algorithms correctness, before continuing searching for faults of higher cardinality. As it can be seen from the example, multiple faults are represented within a ZBDD as a single route from the root node to the terminal **1** node. Each such route denotes a multiple fault. In order to find all the different *SPDFs* that contributed to a *MPDF* fault, $S_i^{\mathscr{C}}$, a two-step operation is used, given by $S_i^{\mathscr{C}} = \left(\Pi_i^{\mathscr{C}} * \mathscr{C}\right) \cap \mathscr{C}$. The resulting set after steps 7-8, contains only *PDFs* that have not contributed to a multiple *PDF*.

### 4.3.2 Finding Multiple Critical Primitive faults

The most challenging step of Algorithm 4.1 is step 5, which is presented in this subsection in more detail. Algorithm 4.2 lists the basic steps for the identifications of all critical primitive *MPDFs* of some cardinality $i$, $i > 1$, given a set of potentially testable critical paths $\mathscr{C}$.

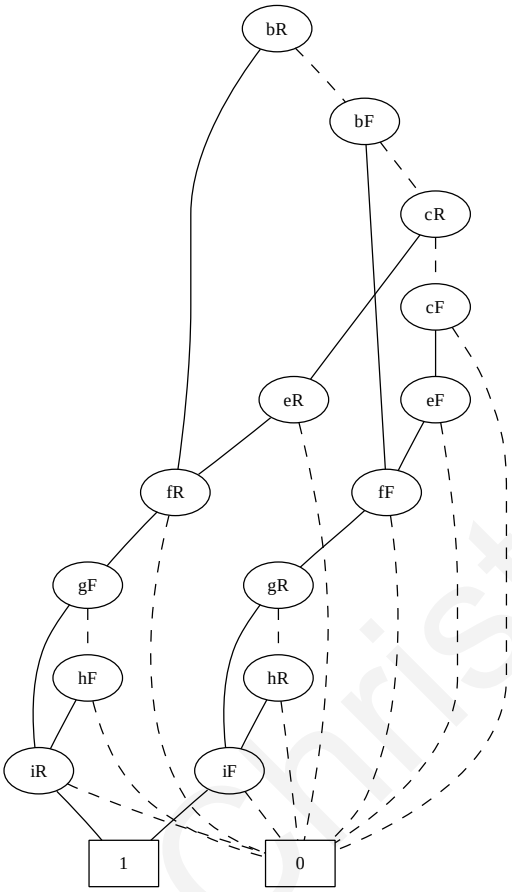



Figure 4.9: Potentially Testable Critical *PDFs* of circuit $C$ ($\mathscr{C}$)

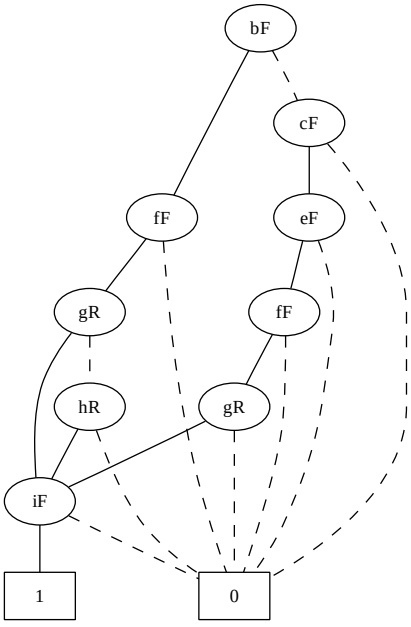



Figure 4.10: Singly Untestable Critical *PDFs* of circuit $C$ ($\mathscr{C}$)

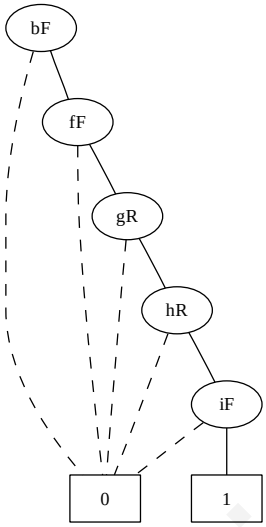



Figure 4.11: Critical Primitive *PDFs* of circuit $C$ ($\mathscr{C} - \Pi_1^{\mathscr{C}}$)

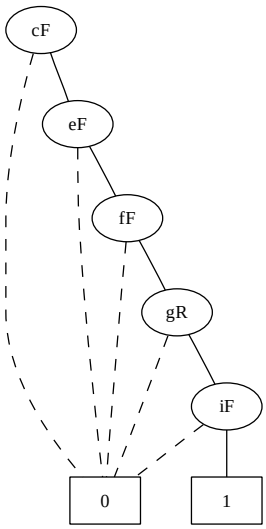



Figure 4.12: Untestable Set *MPDF* set ($\Pi_2^{\mathscr{C}}$)

**Data**: $C$ : Circuit;

$\mathscr{C}$ : Potentially testable critical single $PDFs$;

$i$ : Cardinality;

**Result**: $\Pi_i^{\mathscr{C}}$ : Multiple Critical Primitive $PDF$ Set;

**if** *(cardinality* $i == 2$*)* **then**

  %Traverse $C$ in backward topological;

  **foreach** *(*$l$ *line* $\in C$*)* **do**

    $SinglePO_l^{\mathscr{C}}() \leftarrow$ Single_Sensititized_Segment_PO(C,$\mathscr{C}$,$l$);

  **end**

**else if** *(cardinality* $i > 2$*)* **then**

  $SinglePO_l^{\mathscr{C}}() = SinglePO_l^{\mathscr{C}}() \cap_n (\wedge_m \mathscr{C})$, $(m = Index_l())$;

  %Traverse $C$ in forward topological;

  **foreach** *(*$l$ *line* $\in C$*)* **do**

    %Combination of cardinality $i$ at line $l$;

    **foreach** $(c_{i,l} \in comb_{l,i})$ **do**

      $\Pi_{i,l,c_{i,l}}^{\mathscr{C}} \leftarrow$ Multiple_Critical_Primitive $(C, \mathscr{C}, i, l, c_{i,l})$ ;

      $\Pi_i^{\mathscr{C}} \leftarrow \Pi_i^{\mathscr{C}} \cup \Pi_{i,l,c_{i,l}}^{\mathscr{C}}$;

    **end**

  **end**

**end**

**return** $\Pi_i^{\mathscr{C}}$;

**Algorithm 4.2:** Multiple Critical Primitive $PDF$ Set $\Pi_i^{\mathscr{C}}$

*MPDFs* consist of co-sensitized *SPDFs* that terminate to the same primary output and have the same transitions from their first common line (where the co-sensitized begins) until the primary output line. These are necessary conditions that have to be taken into consideration in order to guarantee the primitivity property. Hence, Algorithm 4.2 starts with a preprocessing phase (steps 1-7). Using a backward topological traversal it generates, per line $l$, the ISOP/ZBDD-based structure containing all singly testable *PDF* segments in $\mathscr{C}$, along with their sensitization cubes, from $l$ to all primary outputs. This information is used to combine different *PDF* segments (these segments are given in $SinglePO_l^{\mathscr{C}}()$), starting at some $PI(s)$ up to the line $l$ examined, that have a common segment to a specific primary output. This process needs only to be executed once (steps 1-5) in the process of finding all critical primitive *MPDFs* since the same data can be used by higher cardinality faults. Hence , steps 1-5 is only performed for $i = 2$ and the segments are globally saved. For $i > 2$, the only step that needs to be taken is to remove those single segments that are not contained in the critical set $\mathscr{C}$, after we have successfully removed the *PDFs* that have contributed to a *MPDF*. This is done in step 7, using two of the newly proposed *ZBDD* operators. Let $m$ denote the index of the *ZBDD* variable used at line $l$, given as $m = Index_l()$. Operator $\wedge_m\mathscr{C}$ returns the segments of the currently examined critical set,$\mathscr{C}$, from line $l$ to the *POs*. The second operation, $\cap_n$, updates set $SinglePO_l^{\mathscr{C}}()$ so as to only contain segments currently in $\mathscr{C}$ (i.e. it removes segments no longer in $\mathscr{C}$). This is necessary as $\mathscr{C}$ is updated after $\Pi_l^{\mathscr{C}}$ is calculated so as to be able to correctly identify $\Pi_{l+1}^{\mathscr{C}}$ (see step 8 of Algorithm 4.1).

After the preprocessing phase, Algorithm 4.2 proceeds in a topological order, from the primary inputs to the primary outputs to find all possible combinations that a *MPDF* of cardinality $i$ can be constructed, line 11-14. For example consider a line $l$, that has two fanins $FI_l() = \{a,b\}$. Examining cardinality $i = 2$, is easy to see that the only way to construct a double sensitized *PDF* is by combining single segments from the two fanins $a$ and $b$. Now setting the primitive cardinality $i = 3$ there are two possible ways that we can sensitize triple *MPDFs*. We can have an already double captured (but not double sensitizable) *MPDF* on fanin $a$ and a single captured segment on fanin $b$ (2+1) or the other way round (1+2). Going further and assuming $i = 4$ there are three possible ways to capture such faults namely 3+1, 1+3 and 2+2. Exploiting lines with larger number of fanins, increases the number of combinations that can form a *MPDF* of cardinality $i$.

Taking into consideration the discussion on the complexity of Algorithm 4.1 and examining Algorithm 4.2, this algorithm consists of a single circuit traversals either steps 1-5 or steps 6-15 are taken. In steps 1-5, a polynomial number of standard ZBDD operators is exe-

cuted (see Chapter 3). If steps 6-15 are executed then it has the same argument of complexity with Algorithm 4.1 with respect the number of combinations necessary, Algorithm 4.3. The complexity of both Algorithm 4.3 is discussed together with Algorithm 4.2. Algorithm 4.3 consists of a linear number of standard and new ZBDD operations of polynomial complexity, per examined combination. Thus, if the number of combinations needed is linear, then Algorithm 4.3 is of polynomial complexity and Algorithm 4.2 also of polynomial since Algorithm 4.3 is executed for all circuit lines.

The most critical step in finding the $\Pi^{\mathscr{C}}_{i,l,c_{i,l}}$ set is performed at step 12 and is given in detail in Algorithm 4.3. Set $\Pi^{\mathscr{C}}_{i,l,c_{i,l}}$ is derived through a linear number of ZBDD operators, including the newly proposed operators. Step 13 adds the newly captured $MPDFs$ to $\Pi^{\mathscr{C}}_i$ and step 17 returns the multiple critical primitive $PDF$ set, $\Pi^{\mathscr{C}}_i$, for cardinality $i$.

Algorithm 4.3 combines $PDFs$ not sensitized up to the examined functionality $i$, from $PIs$ ($\Pi^{\mathscr{C},x}_j$), to specific $POs$ ($SinglePO^{\mathscr{C}}_l()$). The methodology used can also run per $PO$ since primitive multiple sensitized $PDFs$ can be captured on one $PO$. Without any loss of generality the formulas given in Algorithm 4.3 assumes that a specific $PO$ is being examined. This makes notation used more simple for better comprehension.

At any step of the algorithm, if any computation results into the empty set ($\emptyset$), then the algorithm returns without capturing any $MPDF$ for that specific combination currently examined. Temporary $ZBDDs$ $z_1, z_2, z_3$, are used to hold intermediate results of the algorithm. Step 1 checks if there are any common single $PDF$ segments that the Algorithm 4.3 can combine from line $l$ at a primary output. Steps 2 and 3 computes the non-controlling functionality on the fanins of $l$ that do not participate in the specific combination. If the examined gate is a $NAND$/$AND$($NOR$/$OR$), then the $Type(g) = 1$ ($Type(g) = 0$). The type of the gate is actually the non-controlling value for a specific gate type for example the value 1(0) is a non-controlling value for gates $AND$($OR$), $NAND$($NOR$). Step 4 computes the product of the non-controlling functionality on the off inputs and the single segment from $l$ to the $POs$, along with its excitement functionality. This step is used as an intermediate result in steps 11-12 and is assigned to a temporary $ZBDD$ $z_1$. Up to this point all the single $PDF$ segment from $l$ along with the necessary excitement functionality from line $l$ has been computed. Step 5 checks if the output of the previous step 4, is the empty set to stop the Algorithm 4.3. Steps 7 through 19 try to match $PDFs$ from the composing elements of the combination examined. Each combination in the $c_{k,i,l} \in c_{i,l}$, consists of a set of elements that needs to be combined together in order to capture a multiple critical $PDF$. Number $k$ indicates the order of each

**Data**: $C$ : Circuit;

$\mathscr{C}$ : Potentially testable critical single $PDFs$;

$g$ : Circuit gate examined;

$l$ : Circuit line examined;

$c_{i,l}$ : Combination examined;

**Result**: $\Pi^{\mathscr{C}}_{i,l,c_{i,l}}$;

**if** $(SinglePO^{\mathscr{C}}_{l}() == \emptyset)$ **then return** $\Pi^{\mathscr{C}}_{i,l,c_{i,l}} \leftarrow \emptyset$;

**if** $(Type(g) = 1)$ **then** $ncf_{l,c_{i,l}}() \leftarrow \bullet_{y \in FI_l(), y \notin c^{x,j}_{i,l}, c^{x,j}_{i,l} \in c_{i,l}} f_y()$;

**else if** $(Type(g) = 0)$ **then** $ncf_{l,c_{i,l}}() \leftarrow \bullet_{y \in FI_l(), y \notin c^{x,j}_{i,l}, c^{x,j}_{i,l} \in c_{i,l}} \overline{f_y()}$;

$z_1 \leftarrow ncf_{l,c_{i,l}}() \bullet SinglePO^{\mathscr{C}}_{l}()$ ;

**if** $(z_1 = \emptyset)$ **then return** $\Pi^{\mathscr{C}}_{i,l,c_{i,l}} \leftarrow \emptyset$;

%For each element in $c_{i,l}$ such that $k \in \{1,2,..,|c_{i,l}|\}$;

**foreach** $c_{k,i,l} \in c_{i,l}$ **do**

  $z_3 \leftarrow \emptyset$;

  %For each composing element of combination $c_{k,i,l}$;

  **foreach** $c^{x,j}_{k,i,l} \in c_{k,i,l}$ **do**

    **if** $(j == 1)$ **then** $z_2 \leftarrow \left(z_1 \star_n \Pi^{\mathscr{C},x}_{1}\right) \cap_n \mathscr{C}$;

    **else** $z_2 \leftarrow \left(z_1 \star_n \Pi^{\mathscr{C},x}_{j}\right) \cap_n \left(\wedge_n SinglePO^{\mathscr{C}}_{l}()\right)$;

    **if** $(z_3 = \emptyset)$ **then** $z_3 \leftarrow z_2$;

    **else** $z_3 \leftarrow z_3 \star_n z_2$;

  **end**

  $z_3 \leftarrow z_3 \cap_n \left(\wedge_n SinglePO^{\mathscr{C}}_{l}()\right)$;

  **if** $(\Pi^{\mathscr{C}}_{i,l,c_{i,l}} = \emptyset)$ **then** $\Pi^{\mathscr{C}}_{i,l,c_{i,l}} \leftarrow z_3$;

  **else** $\Pi^{\mathscr{C}}_{i,l,c_{i,l}} \leftarrow \Pi^{\mathscr{C}}_{i,l,c_{i,l}} \cup z_3$;

**end**

**return** $\Pi^{\mathscr{C}}_{i,l,c_{i,l}}$;

**Algorithm 4.3:** Multiple Critical Primitive $PDF$ Set $\Pi^{\mathscr{C}}_{i,l,c_{i,l}}$

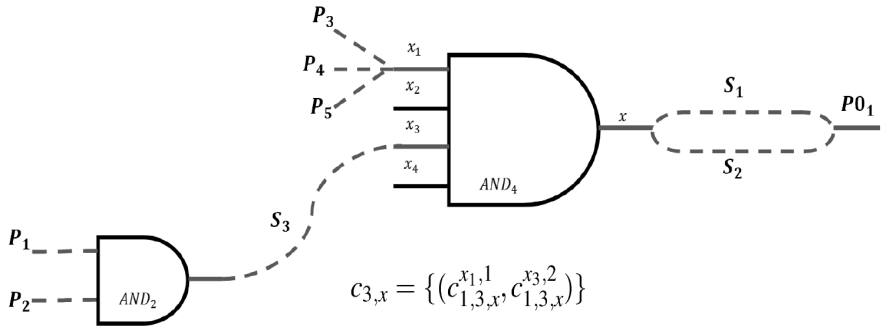



Figure 4.13: Example Circuit $F$

set in $c_{i,l}$. Basically $c_{i,l}$ is a set of sets of elements. Each element, $c^{x,j}_{k,i,l} \in c_{k,i,l}$, corresponds to a specific fanin line $x$ and a specific cardinality examined $j$. Adding all cardinality for each element in the set should equal $i$, that is the examined cardinality. Steps 7 loops among all possible combinations and step 10 loops among all elements of a specific combination. Step 8 is just an initialization of the temporary ZBDD variable $z_3$. If the cardinality, $j$, of the combination element $c^{x,j}_{n,i,l}$examined on fanin $x$ is 1, then step 11 is executed otherwise step 12 is taken. For better understanding of steps 11-16, which contain the newly introduced operators is best to see these steps through an example in the next paragraph. Steps 17-20 simply add the resulting ISOP/ZBDD with the newly captured critical primitive $PDFs$, steps 17-18, and return the output of Algorithm 4.3, step 20.

Consider the example circuit $F$ of Fig. 4.13. Following Algorithm 4.3 on this example circuit with the assumption that the current cardinality being examined is three, $i = 3$. Consider critical set $\mathscr{C} = \{P1.S3.S1,\ P2.S3.S1,\ P3.S1,\ P4.S1,\ P5.S2\}$, containing five elements, that is 5 single critical $PDFs$. The circuit contains an $AND_4$ gate, thus the gate type is 1, at line $x$ with 4 fanins $FI_x = \{x_1, x_2, x_3, x_4\}$. The current combination examined is $c_{3,x}$ and it consists of a single combination consisted by 2 elements namely, $c_{3,x} = \{(c^{x_1,1}_{1,3,x}, c^{x_3,2}_{1,3,x})\}$. This implies that Algorithm 4.3 will try to combine single $PDFs$ captured at $x_1$ fanin and doubly captured $PDFs$ at $x_2$ fanin. The requirement is that all the $PDFs$ considered singly or doubly captured have been invalidated at some point from the examined gate or later. That's why these $PDFs$ are not singly or doubly sensitized. Examining $x_1$ fanin of the $AND_4$ gate, three single segments with a total of 3 $PDFs$ have been detected namely $\{P3, P4, P5\}$. Observing the third fanin, $x_3$, a multiple fault of cardinality two, $\{P1.P2.S3.S1\}$, has been captured on an $AND_2$ gate before, but has been invalidated at some point between the examined $AND_4$ gate and the primary output.

Let us assume that the single segment $SinglePO^{\mathscr{C}}_x() = \{ncf_{S1}.S1, ncf_{S2}.S2\}$, step 1. The

*ncv* functionality on the site inputs has been computed at step 2 , $f_{x_2}().f_{x_4}()$, and step 4 computes the product of the non-controlling functionality on the off-site inputs previously computed and the single segment $SinglePO_x^{\mathscr{C}}()$ from step 1. This is stored in temporary *ZBDD* $z_1 = \{f_{x_2}().f_{x_4}().ncf_{S1}.S1, f_{x_2}().f_{x_4}().ncf_{S2}.S2\}$. Since $z_1$ is not empty, step 5, the algorithm proceeds with step 7. Since only one set of elements is in $c_{3,x}$ the algorithm proceeds on step 10, after initializing temporary ZBDD $z3$ at step 8.

Now for simplicity we abstract all functionality information from our example. Examining fanin $x_1$ step 11 is being executed. The first term of the equation, $z_1 \star_n \Pi_1^{\mathscr{C},x} =$, in step 11 is first computed. This term computes the product between the two ZBDD using binate product on the two ZBDDs until index $n$ and applying unate product for rest part of the ZBDDs. This is necessary since the problem examined is described with 2 variable kinds. We want the product operation on the functionality information of the ISOP/ZBDD to be performed in a binate way and the rest information that concerns *PDFs* to be performed in a unate fashion. Thus applying the 2 way product will produce $z_1 \star_n \Pi_1^{\mathscr{C},x} = \{S1, S2\} \star_n \{P3, P4, P5\}$ $= \{P3.S1,\ P4.S1,\ P5.S1,\ P3.S2,\ P4.S2,\ P5.S2\}$. As we can see from the produced result *PDFs* that are not part of the initial critical set have been produced for example $P4.S2$. In order to remove these false *PDFs* the second term of the equation is applied on the computed set of the first part that is $z_2 = \{P3.S1,\ P4.S1,\ P5.S1,\ P3.S2,\ P4.S2,\ P5.S2\} \cap_n \mathscr{C}$ $= \{P3.S1, P4.S1, P5.S2\}$. In this part of the equation an intersection between the resulted term from the first part and the critical set will remove these false *PDFs*. The intersection concerns only *PDF* variables that is with index $>= n$. Since we are investigating single *PDFs* we can apply the special intersection operation on the critical set which contains full *PDF* information.

For simplicity it is assumed that $P4.S1$ has not been included in $z_2$ because it has been removed by the functionality constrains(on the first part of the operation the special product). Thus $z_2 = \{P3.S1, P5.S2\}$. Step 11 is then executed since $z_3 = \emptyset$. The second half of the combination is examined , that is *PDFs* coming through fanin $x_3$. This combinations looks into the already captured doubly *PDFs* from $x_3$, thus line 12 of Algorithm 4.3 is executed. The first term of line 12 follows the same logic as in line 11, thus $z_2 = z_1 \star_n \Pi_{j>1}^{\mathscr{C},x}$ $= \{S1, S2\} \star_n \{P1.P2.S3.S1\} = \{P1.P2.S3.S1, P1.P2.S3.S1.S2\}$. The same phenomenon occurs as in line 11, that is false *PDFs* have been generated. Since now we can not apply the special intersection on the critical set since this is a multiple *PDF* and its representation differs. All the functional information are deleted from the single segment information on line $x$, with the operation $\wedge_n SinglePO_l^{\mathscr{C}}()$. Applying the special intersection operation will

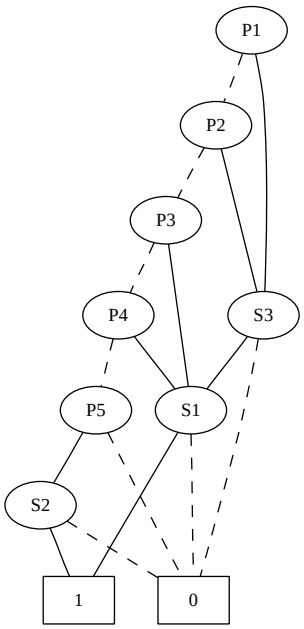



Figure 4.14: Critical ZBDD *PDFs* in Circuit $F$

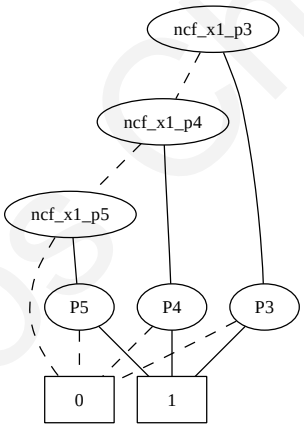



Figure 4.15: ISOP/ZBDD Graph for $x_1$ fanin of $F$

remove all those multiple captured *PDFs* that are not contained in the segments.

This results in removing the false *PDFs*, $z_2 = \{P1.P2.S3.S1\}$. Step 14 is then executed and this also introduces some false *PDFs*. Specifically $z_3 = \{P3.S1, P5.S2\} \star_n \{P1.P2.S3.S1\}$ and this results in $z_3 = \{P1.P2.S3.P3.S1, P1.P2.S3.P5.S1.S2\}$. All the combination have been covered thus step 16 is executed to clear all the false *PDFs*. The same arguments and explanation as in step 14 applies in this case too. Thus true multiple *PDF* $\{P1.P2.S3.P3.S1\}$ is captured and consists of three *PDFs* namely $\{P1.S3.S1, P2.S3.S1, P3.S1\}$. Step 17 is executed since $\Pi^{\mathscr{C}}_{i,l,c_{i,l}} = \emptyset$ and then step 20 returns the result.

Figure Fig. 4.14 shows the critical *PDFs* of circuit $F$, Fig. 4.15 the ISOP/ZBDD graph

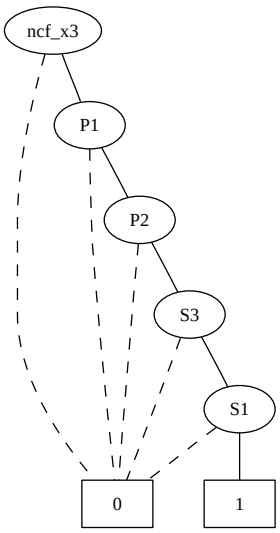



Figure 4.16: ISOP/ZBDD Graph for $x_3$ fanin of $F$

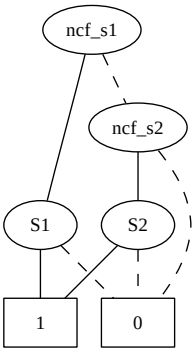



Figure 4.17: ISOP/ZBDD Graph for *SinglePO* of $x$

of fanin $x_1$ of circuit $F$ and Fig.4.16 shows the ISOP/ZBDD graph of fanin $x_3$ of circuit $F$. In 4.17 shows the *SinglePO* of line $x$ of sample circuit $F$. In these three figures, Fig.4.14 -4.17, ZBDD variables represent path segments, a sequence of variable. This does not imply that there is not variable sharing between the path segments. The abstraction has been made for better understanding/simplicity of Algorithm 4.3.

### 4.3.3 Necessary Function Formulation

Necessary function formulation are given below. Function $SinglePO_l^{\mathscr{C},tr}()$, is being used in Algorithms 4.2 and 4.3. At line $l$, $SinglePO_l^{\mathscr{C},tr}()$ denotes all the *PDF* segments from line $l$ until a primary output has been reached along with all necessary sensitization criteria at the *PDF* segments off-input and belong to the critical $\mathscr{C}$ set. Moreover $tr \in \{R,F\}$ denotes the transition type rising($R$) or falling($F$). These information are extracted from the circuit starting from the primary outputs until the primary inputs have been reached. These functions also are defined over a specific cardinality and a specific primary output. The definition of this function per primary output is not a necessary step as it can be defined over all the primary outputs, its only done for simplicity. The cardinality number is needed since the algorithm iteratively computes the primitive critical *PDF* set of a specific cardinality. In order to be correct we must remove all the segments captured at each iteration. This is achieved by the following operation:

$$SinglePO_l^{\mathscr{C},tr}() = SinglePO_l^{\mathscr{C},tr}() \cap_n (\wedge_m \mathscr{C}) \tag{4.1}$$

where $m = Index_l()$. This means that only segments belonging to the currently examined critical set, $(\wedge_m \mathscr{C})$, are considered. Algorithm 4.1 is a recursive operation on the cardinality number. At each iteration step a newly computed critical *PDF* set is being computed in order to maintain the primitivity correctness. Only those *PDF* that have not contributed to a lower cardinality *PDF* can contribute to the next iteration. Getting all the critical *PDF* segment from line $l$ $(\wedge_m \mathscr{C})$ and then intersecting with the constructed set when cardinality was two $i-1$ we remove all those *PDF* segment that have contributed on $i-1$ iterations. Maintaining a list at each line for every primary output makes it faster to find the primitive critical *PDF* set because multiple primitives critical *PDFs* combine at a gate for a certain primary output.

Let $o \in PO$. We start by defining functions $SinglePO_o^{\mathscr{C},R}()(SinglePO_o^{\mathscr{C},F}())$ at the *PO*. Each $o \in PO$ is associated with two path variables, $iR$ and $iF$. ZBDDs $z_1$ and $z_0$ denote the terminal one node and terminal zero node respectively. To represent a transition on

$o$, the appropriate path variable ($oR$ or $oF$) need to be included. The necessary function formulations at a primary output $o$ are given below:

$$SinglePO_o^{\mathscr{C},R}(T \cup \mathscr{P}) \;=\; !(z1,\, oR) \quad o \in POs \tag{4.2}$$

$$SinglePO_o^{\mathscr{C},F}(T \cup \mathscr{P}) \;=\; !(z1,\, oF) \quad o \in POs \tag{4.3}$$

Consider now an AND gate $g$ with output line $l$ and fanins $FI(l)$. *PDF* segments up to line $l$ along with all the sensitization information needs to traverse backwards through any of the lines in $FI(l)$. Any line in $FI(l)$ can be considered as an on-input with the remaining lines being off-inputs that must settle to $ncv_g$. Consider line $y \in FI(l)$ to be an on-input. Function $\bullet_{x \in FI(l), x \neq y} f_x()$ will give all test cubes that allow all remaining off-inputs in $FI(l)$ to settle to $ncv_g = 1$ (given by $f_x()$ in the expression). Function $!((SinglePO_l^{\mathscr{C},tr}() \bullet (\bullet_{x \in FI(l), x \neq y} f_x())),\, l)$ will give all non-robustly sensitizable *PDFs* up to line $y$, through line $l$, along with the sensitization cubes. Every line in *FI(l)* is a possible on-input not only on gate $g$ but also on other gates (to cover fanout stems), thus function $SinglePO_y^{\mathscr{C},tr}()$ given by:

$$SinglePO_y^{\mathscr{C},tr}() \;=\; \left(!\left(\left(\left(\, SinglePO_l^{\mathscr{C},tr}() \bullet (\bullet_{x \in FI(l), x \neq y} f_x())\, \right)\right), y^{tr}\right), \bigcap_n (\wedge_m \mathscr{C})\right) \bigcup \left(SinglePO_y^{\mathscr{C},tr}()\right) \tag{4.4}$$

For an OR gate $g$ the off-inputs in $FI(l)$ need to settle to $ncv_g = 0$. Thus, it suffices to replace $f_x()$ with $\overline{f_x()}$ in Eq. 4.3, as given below:

$$SinglePO_y^{\mathscr{C},tr}() \;=\; \left(!\left(\left(\left(\, SinglePO_l^{\mathscr{C},tr}() \bullet (\bullet_{x \in FI(l), x \neq y} \overline{f_x()})\, \right)\right), y^{tr}\right), \bigcap_n (\wedge_m \mathscr{C})\right) \bigcup \left(SinglePO_y^{\mathscr{C},tr}()\right) \tag{4.5}$$

The $SinglePO_y^{\mathscr{C},R}()$ and $SinglePO_y^{\mathscr{C},F}()$ functions for the output line $l$ of a NOT gate with input line $y$ are given below.

$$SinglePO_y^{\mathscr{C},R}() \;=\; !\left(SinglePO_l^{\mathscr{C},F}(),\, y^R\right) \bigcup SinglePO_y^{\mathscr{C},R}() \tag{4.6}$$

$$SinglePO_y^{\mathscr{C},F}() \;=\; !\left(SinglePO_l^{\mathscr{C},R}(),\, y^F\right) \;\bigcup\; SinglePO_y^{\mathscr{C},F}() \tag{4.7}$$

$SinglePO_y^{\mathscr{C}}()$ is computed iteratively on a reversed topological traversal manner until a primary input has been reached. Starting from the *PIs* until a *PO* has been reached the algorithm for finding a sensitized critical primitive *MPDF* tries to combine at a gate different fanins of that gate to form the *MPDF* (this step is done with $FuncP_k()$) and then tries to find common way to a single primary output through $SinglePO_y^{\mathscr{C}}()$. In these operations the transition is implied. That is we can only combine the rising transitions to a specific gates fanins and we look for a common way to the primary output with the same transition unless the gate is an inverting gate thus we look for the opposite transition.

Combining *MPDFs* can only occur on internal gates, thus not on primary input lines, and only on lines where the number of fanins is greater than 1. Thus single fanin gates, buffers and inverting gates, are also excluded, since no combination of paths can occur on these gate types. Moreover only *PDFs* of the same transition type can be captured together. That is on a line we can not combine a rising *PDF* from one fanin and a falling from another. The function formulation for capturing multiple *PDFs* of cardinality $i$ at circuit line $l$ and identifying the primitive critical *PDFs* at internal gates with greater than one fanin,gates of type *AND*, *NAND*, *OR* and *NOR*, is given below:

$$\Pi_{i,l}^{\mathscr{C},tr} = \bigcup_{c_{i,l}\in comb_{i,l}} \left( \left( \star_n \atop {c_{k,i,l}^{x,j}\in c_{k,i,l}} \left( \left( ncf_{l,c_{i,l}}() \bullet SinglePO_l^{\mathscr{C}}() \right) \star_n Z_1 \right) \cap_n \left( \wedge_n SinglePO_l^{\mathscr{C}}() \right) \right) \right) \tag{4.8}$$

Let $Z_1$ to be a temporary *ZBDD*. Let $x \in FI_l()$ and $j < i$ to denote the cardinality of the composing *PDFs* from fanin $x$ of line $l$. Then if $j > 1$ then $Z_1 \;=\; \Pi_{j>1}^{\mathscr{C},x}()$ else if $j = 1$ then $Z_1 \;=\; \Pi_1^{\mathscr{C},x}() \cap_n \mathscr{C}$. Computing the non-controlling functionality $ncf_{l,c_{l,i}}()$ depends on the type of the examined gate $g$. If the gate is *NAND* or *AND* then the non-controlling on the off inputs is 1 thus $ncf_{l,c_{l,i}}() \leftarrow \bullet_{y\in FI_l(),y\notin c_{k,i,l}^{x,j},c_{k,i,l}^{x,j}\in c_{k,i,l}} f_y()$. If the non-controlling functionality on the site inputs is 0, *OR* and *NOR* gate types, then $ncf_{l,c_{i,l}}() \leftarrow \bullet_{y\in FI_l(),y\notin c_{k,i,l}^{x,j},c_{k,i,l}^{x,j}\in c_{i,l}} \overline{f_y()}$.

Function $\Pi_j^{\mathscr{C},x}()$ denotes the already captured but not sensitized *PDFs* of cardinality $j < i$, where $i$ denotes the currently examined cardinality and $j$ a lower cardinality captured *PDF*. The main difference from the previous function formulation is that these *PDF* have been invalidated by the functionality on the single segment from $l$ to a *PO*. Thus the non-controlling functionality of the combination examined $ncf_{l,c_{i,l}}()$ is combined with the *ZBDD*

product operation of the single critical segments from $l$ to a $PO$ with no added functionality on the site inputs of the segment, $\wedge_m \mathscr{C}$, where $m = Index_l()$.

$$\Pi_i^{\mathscr{C},l,tr} = \bigcup_{c_{i,l} \in comb_{i,l}} \left( \left( \star_n \atop {c_{k,i,l}^{x,j} \in c_{k,i,l}} \left( \left( ncf_{l,c_{i,l}}() \bullet (\wedge_m \mathscr{C}) \right) \star_n Z_1 \right) \cap_n \left( \wedge_n SinglePO_l^{\mathscr{C}}() \right) \right) \right) \tag{4.9}$$

### 4.3.4 Maintaining the Primitivity Property-A Three Step Procedure

One of the most important issues in finding the primitive fault set is preserving the *Primitivity Property*. The Primitivity Property must be preserved throughout the whole procedure in order to have a *correct* primitive path fault set detected. In simple terms a *correct* primitive path fault set means that it does not contain smaller cardinality faults. For example if a *PDF* was a part of a primitive fault set of cardinality $i$, then it can no longer contribute and be a part a higher cardinality primitive path fault set.

The primitivity property is preserved at steps 2, 7 and 8 of Algorithm 4.1, by removing the already detected *PDFs* from the critical path set. Step 2 of Algorithm 4.1 is a standard set difference ZBDD operation. Step 7 and 8 involves multiple faults and is treated differently. These steps are a necessity for the algorithms correctness. Step 7 of Algorithm 4.1 detects the different single *PDFs* that form a multiple critical primitive fault and step 8 removes them from the critical set examined. These steps must be taken before continuing searching for faults of higher cardinality, than those currently examined. Observe that step 2 of Algorithm 4.1 involved only single *PDFs*.

Multiple faults are represented within a ZBDD as a single route from the root node to the terminal 1 node. Each such route denotes a multiple fault. In order to find all the different *PDFs* that contributed to that fault, a three-step operation is used. The multiple fault set ($\Pi_i^{\mathscr{C}}$) and the critical path set ($\mathscr{C}$) are used in these steps. First, at step 7, the Cross Product of the two set is computed, $\Pi_i^{\mathscr{C}} * \mathscr{C}$. Then the standard ZBDD intersection operator , $\cap$, is applied on the resulting set of the Cross Product and the critical path set. The resulting ZBDD $S_i^{\mathscr{C}} = \left(\Pi_i^{\mathscr{C}} * \mathscr{C}\right) \cap \mathscr{C}$ contains only the single *PDFs* that contributed in the multiple critical primitive *PDF* set. The third and final step, 8, is the set difference ZBDD operation between $S_i^{\mathscr{C}}$ and $\mathscr{C}$. The resulting set, $\mathscr{C}$ contains only *PDFs* that have not been merged and contributed to a multiple *PDF*. Thus $\mathscr{C}$ will be the set of critical *PDFs* not yet detected and that set will be used for finding higher cardinality primitive critical faults.

$$\mathscr{C} = \mathscr{C} \setminus \left( \left( \Pi_i^{\mathscr{C}} * \mathscr{C} \right) \cap \mathscr{C} \right) \tag{4.10}$$

The steps taken for maintaining the primitivity property is illustrated in detail for the circuit of Fig. 4.5. When the single testable *PDFs* have been found then these are removed from the initial set of potentially testable critical *PDFs*. For the circuit *C* of Fig. 4.5 the initial $\mathscr{C}$ set is composed of 8 *PDFs* and 5 of them are singly non-robustly testable. The set difference operator is applied to the initial *PDF* set and the set of singly testable *PDF* set and the resulting *PDFs*, 3, will be used for finding $\Pi_2^{\mathscr{C}}$ set. Fig. 4.10 defines the potentially testable critical path set of circuit *C* of Fig. 4.5 for finding set $\Pi_2^{\mathscr{C}}$. The ZBDD that contains $\Pi_2^{\mathscr{C}}$ set is showed on Fig. 4.11.

(1) Computing Cross Product:

$$\begin{aligned} \Pi_i^{\mathscr{C}} * \mathscr{C} &= \{bF.fF.gR.iF, bF.fF.hR.iF, cF.eF.fF.gR.iF\} * \{bF.fF.gR.hR.iF\} \\ &= \{bF.fF.gR.iF, bF.fF.hR.iF, fF.gR.iF\} \end{aligned}$$

(2) Computing Intersection:

$$\begin{aligned} \left( \Pi_i^{\mathscr{C}} * \mathscr{C} \right) \cap \mathscr{C} &= \{bF.fF.gR.iF, bF.fF.hR.iF, fF.gR.iF\} \cap \\ &\quad \{bF.fF.gR.iF, bF.fF.hR.iF, cF.eF.fF.gR.iF\} \\ &= \{bF.fF.gR.iF, bF.fF.hR.iF\} \end{aligned}$$

(3) Removing detected contributing single *PDFs*:

$$\begin{aligned} \mathscr{C} \setminus \left( \left( \Pi_i^{\mathscr{C}} * \mathscr{C} \right) \cap \mathscr{C} \right) &= \{bF.fF.gR.iF, bF.fF.hR.iF, cF.eF.fF.gR.iF\} \setminus \\ &\quad \{bF.fF.gR.iF, bF.fF.hR.iF\} \\ &= \{cF.eF.fF.gR.iF\} \end{aligned}$$

Thus the set $\mathscr{C} = \{cF.eF.fF.gR.iF\}$ is the potentially testable critical path set of circuit *C* that will be used to find $\Pi_3^{\mathscr{C}}$ and higher cardinality primitive critical *PDFs*. The three step procedure must be executed every time that a $\Pi_i^{\mathscr{C}}$ has been derived. This way it is ensured that the newly detected primitive critical *PDFs* have not been used in a lower cardinality primitive critical *PDFs* set. The resulting ZBDD of the third step can be seen in Fig. 4.12.

### 4.3.5 New *ZBBD* Operators for primitive *PDFs*

Three new, non-enumerative operators are introduced below. The need for these new ZBDD operators arises from the problem addressed in this work. Critical primitive paths are stored and manipulated in ZBDDs and existing ZBDD operators cannot support the problem formulation. These new operators do not increase the performance. All of the three newly proposed ZBDD operators are based on a depth first search and they are linear in terms the depth of the ZBDD. In the worst case scenario, any ZBDD operator may require to explore the ZBDD size, number of ZBDD nodes. As the primitive cardinality increases, the number of operations used, decreases. If these operators are performed at one element of the set at a time, then the time needed to compute it would depend on the size of the set. This is not practical for large sets. The new operators are based on the divide and conquer concept, as most of the operators of the cudd package. Terms are computed recursively, using hash-based cache thereby avoiding unnecessary execution. Thus the time needed for all of the three newly operators are linear to the ZBDD size. The complexity of the first two operators ($\cap_n, \star_n$) is in the worst case polynomial to the input ZBDDs and the complexity for the third operator ($\wedge_n$) is linear to the input ZBDD. All these operators have been used in the algorithm given finding the critical primitive PDF set.

Furthermore operators New-Intersect($P \cap_n Q$) and New-Product($P \star_n Q$) can not be constructed using existing ZBDD operators because these operators are specific to the addressed problem and handle binate and unate variables in a single structure, the ISOP/ZBDD structure. On the other hand operator Segment($\wedge_n P$) can be constructed by taking the union over the subset(0) and subset(1) over all variables with $index < n$ on ZBDD $Q = P_{v_1}$, for variable $v$ such that $Index(v) = n$. If $R$ is the resulting ZBDD then $R = \underset{v,Index(v)<n}{\forall} Q_{v_1} \cup Q_{v_0}$. In order to find $R$, $n$ union operations and $2 * n$ subset operations are needed. The Segment operation is at least twice faster since it explores the tree structure of the given ZBDD and uses the cache to store partial results thus avoiding extra operations to occur.

Another important observation is that the ZBDD operators, enable complex operations to be performed in an non-enumerative and implicit manner. This is really important in terms of memory and time requirements. From this point we assume that $P$ and $Q$ denote a ZBDD.

#### Operator New-Intersect $P \cap_n Q$

A new ZBDD operator is introduced ($\cap_n$) to perform intersection between the elements of two sets after a certain number of variables. This operator is needed since the methodology

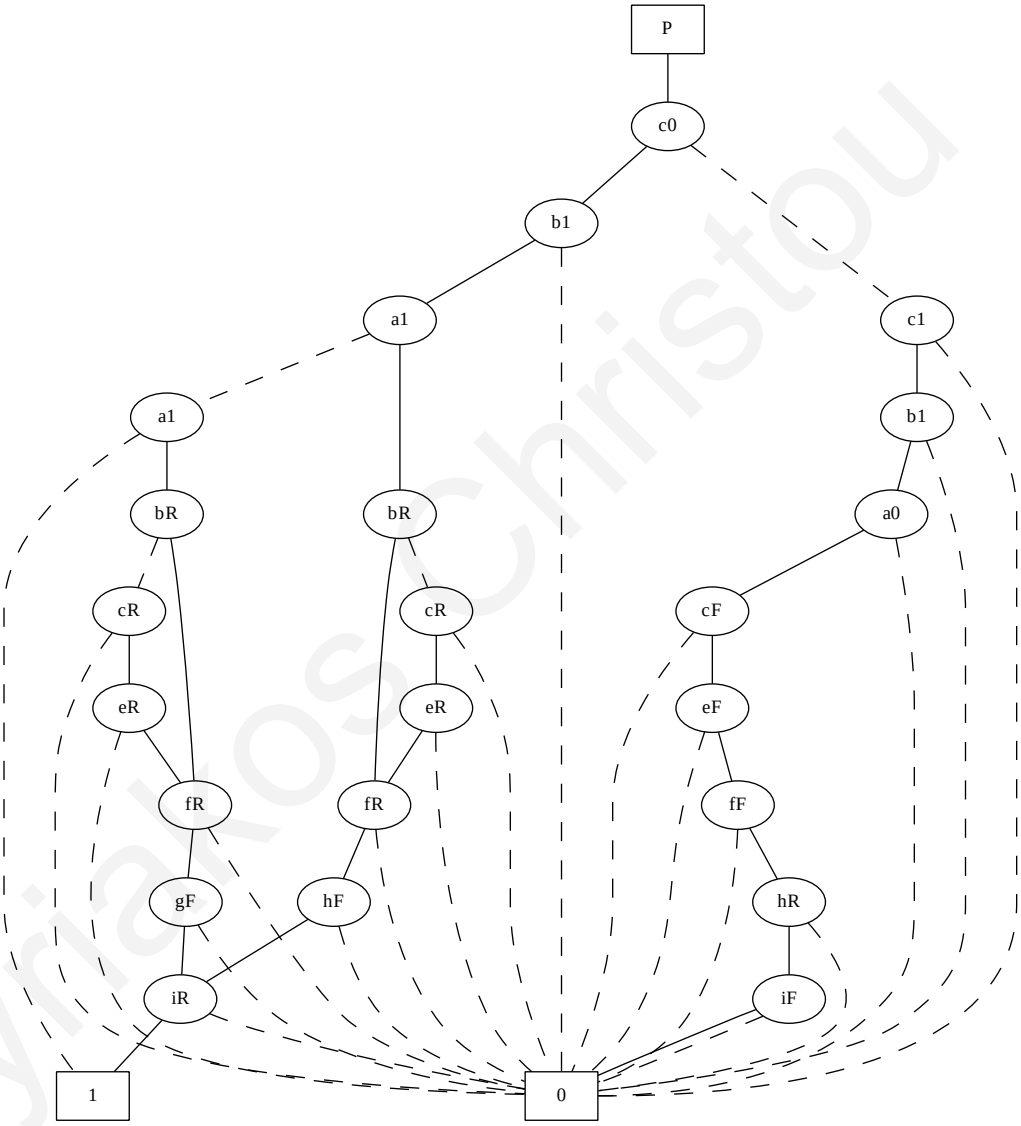



Figure 4.18: ISOP/ZBDD Graph for singly testable *PDFs* of $C$ ($P$)

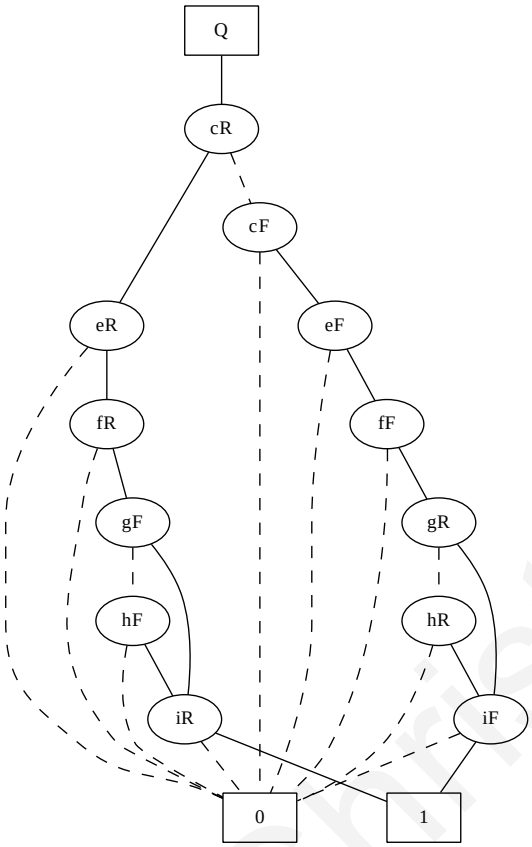



Figure 4.19: *PDFs* from PI *c* of Circuit *C* (*Q*)

used uses the ZBDD-based ISOP structure which consists of two kinds of variables, test and path variables. The input to the above operator is the $P$ set which contains both variable kinds and the $Q$ set only path variables. The number $n$ provided, denotes the first path variable found in order. This operation returns a ZBDD that contains only the paths that are common to both $P$,$Q$ sets, along with their functionality (which is only found on $Q$). Thus, this operator has the same complexity with the standard ZBDD intersection.

This is illustrated through the following example. Let $R$ denote the resulting set of the new-Intersection operation, $R = P \cap_n Q$. Let $P$ be the set that contains all the single testable *PDFs* of circuit of Fig. 4.5 along with their functionality. Fig. 4.18 denotes the ZBDD-based ISOP structure for $P$. There are exactly 5 *PDFs* contained in this structure. Let $Q$ be all the *PDFs* that begin from primary input $c$ of circuit $C$ of Fig. 4.5. Fig. 4.19 is the ZBDD that contains three *PDFs*. There are 4 routes from the root to the terminal node 1 representing the 4 *PDFs* that start from primary input $c$ of circuit $C$. Since the number of primary inputs of circuit $C$ is 3 then 6 test variables are needed, 0 to 5. Thus the path variables begin from variable number 6. Thus $R$ is computed as $R = P \cap_6 Q$. The resulting ZBDD-based ISOP structure that represents $R$ is shown in Fig. 4.20. Observe that there are exactly 3 *PDFs* along

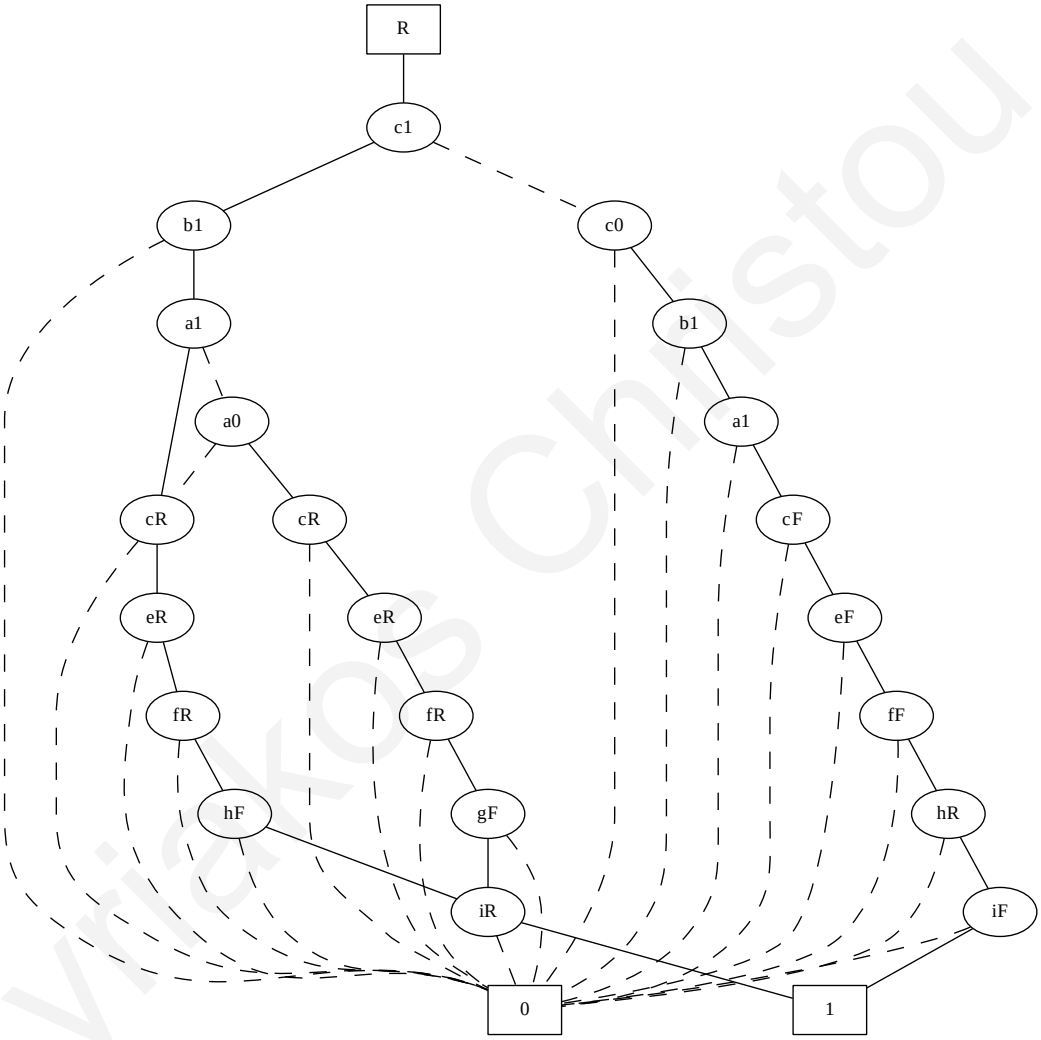



Figure 4.20: ISOP/ZBDD Graph $R = P \cap_n Q$

with their functionality and that the order of the test variables is $c > b > a$. The resulting ZBDD $R$ eliminates all those $PDFs$ along with their functionalities that are not contained in $Q$.

**Operator New-Product** $P \star_n Q$

```
   Data: ZBDD P,ZBDD Q,n;
   Result: ZBDD R;
 1 if (P = ∅||Q = ∅) then return ∅ ;
 2 if (P=Q) then return P;
 3 if (cache(P ∩_n Q) ≠ ∅) then return cache(P ∩_n Q);
 4 P_Top(Q_Top) ← Top Variable of P(Q);
 5 P_1(P_0) ← T(P)(E(P));
 6 Q_1(Q_0) ← T(Q)(E(Q));
 7 if (P_Top ≧ n) then
 8     if (P_Top > Q_Top) then R ← P ∩_n Q_0;
 9     else if (P_Top < Q_Top) then R ← P_0 ∩_n Q;
10     else
11         Z_0 ← P_0 ∩_n Q_0;
12         Z_1 ← P_1 ∩_n Q_1 ;
13         R ← ZBDD(P_Top,Z_1,Z_0);
14     end
15 else
16     Z_0 ← P_0 ∩_n Q;
17     Z_1 ← P_1 ∩_n Q;
18     R ← ZBDD(P_Top,Z_1,Z_0);
19 end
20 cache(P ∩_n Q) ← R;
21 return R;
```

**Algorithm 4.4:** New-Intersect: $P \cap_n Q$

This operator takes as input two ZBDDs $P$,$Q$ and a number, $n$, which denotes a certain variables order in the set. In this operator, from the top variable of both *ZBDDs* until $n$ has been reached, the binate product of the 2 covers is performed and from $n$ variable and below

the unate product of the 2 covers. The test variables have two ZBDD variables adjacent in order (binate) while the path variables use only one (unate). Thus, the complexity of this operator is of polynomial in the worst case.

Take for instance the example shown in Fig. 4.21, Fig. 4.22 and Fig. 4.23. As an input for the new product operator the two ISOP/ZBDD structures in Fig. 4.21 and Fig. 4.22 will be used. Fig. 4.21 denotes the ISOP/ZBDD structure, $P$, for the falling transition on line $g$ while Fig. 4.22 denotes the ISOP/ZBDD structure, $Q$, for the falling transition on line $h$. Fig. 4.23 denotes the ISOP/ZBDD structure, $R$, which is the $R = P \star_6 Q$, $n = 6$ since only 6 test variables exist. In $P$ there are 2 $PDFs$, namely {bR.fR.gR.gF.iR, cR.eR.fR.gF.iR} and in $Q$ there are also 2 $PDFs$, {bR.fR.hF.iR, cR.eR.fR.hF.iR}. The resulting ISOP/ZBDD based structure contains 3 doubly sensitized $PDFs$. In the example the $PDFs$ that $P$ and $Q$ consist of, are all non-robustly singly sensitizable. This example aims to show the potentials of the new proposed operator.

**Data**: ZBDD $P$,ZBDD $Q$, $n$;
**Result**: ZBDD $R$;
1 $P_{Top}(Q_{Top}) \leftarrow$ Top Variable of $P(Q)$ ;
2 **if** $(Q_{Top} > P_{Top})$ **then** $R \leftarrow Q \star_n P$;
3 **else if** $(P_{Top} < n)$ **then** $R \leftarrow P$ *Binate Product* $Q$;
4 **else** $R \leftarrow P$ *Unate Product* $Q$;
5 **return** $R$

**Algorithm 4.5:** New-Product: $P \star_n Q$

Now observing the resulting ISOP/ZBDD structure, 3 $PDFs$ can be distinguished, namely {bR.cR.eR.fR.gF.hF.iR, bR.fR.gF.hF.iR, cR.eR.fR.gF.hF.iR}. Each of these $PDFs$ consist of two $PDFs$ one in set $P$ and the other in $Q$. For instance $PDFs$ {cR.eR.fR.gF.iR} and {cR.eR.fR.hF.iR} form the double $PDF$ {cR.eR.fR.gF.hF.iR}. The power of this operator hangs from the fact that someone can use this operator and extract the multiply sensitized $PDFs$ without any restrictions on the cardinality of the multiple $PDFs$ it considers.

### Operator Segment $\wedge_n P$

The third operator takes as input one ZBDD $P$, and a number, $n$, which denotes a variable in the set. It returns another ZBDD that consists of routes of the given ZBDD $P$ that contain the variable number $n$. This actually is an operator that answers the following question: *What*

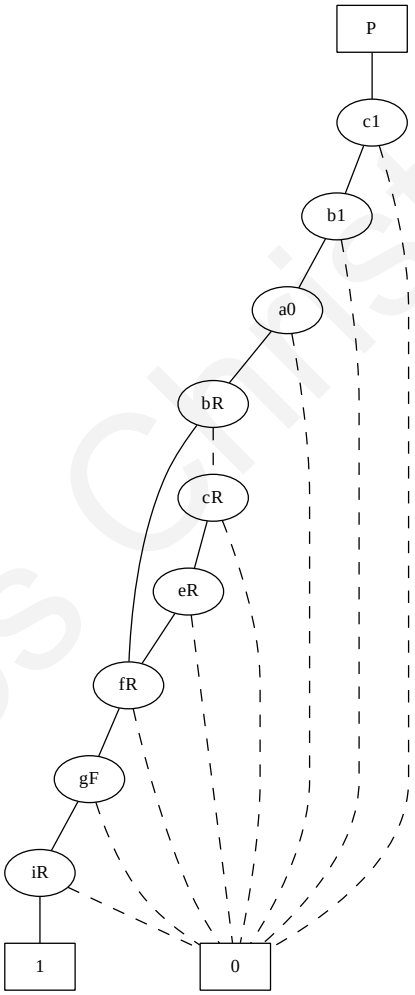



Figure 4.21: *P*-ISOP/ZBDD Graph for falling *PDFs* of line *g*

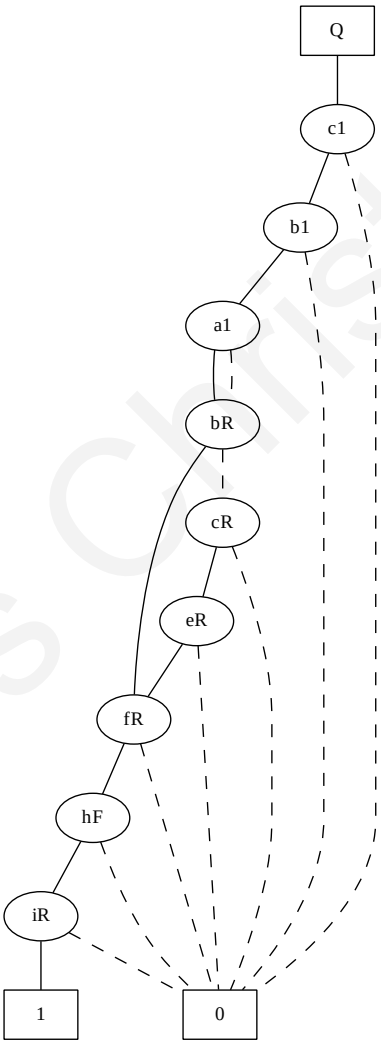



Figure 4.22: $Q$-ISOP/ZBDD Graph for falling $PDFs$ of line $h$

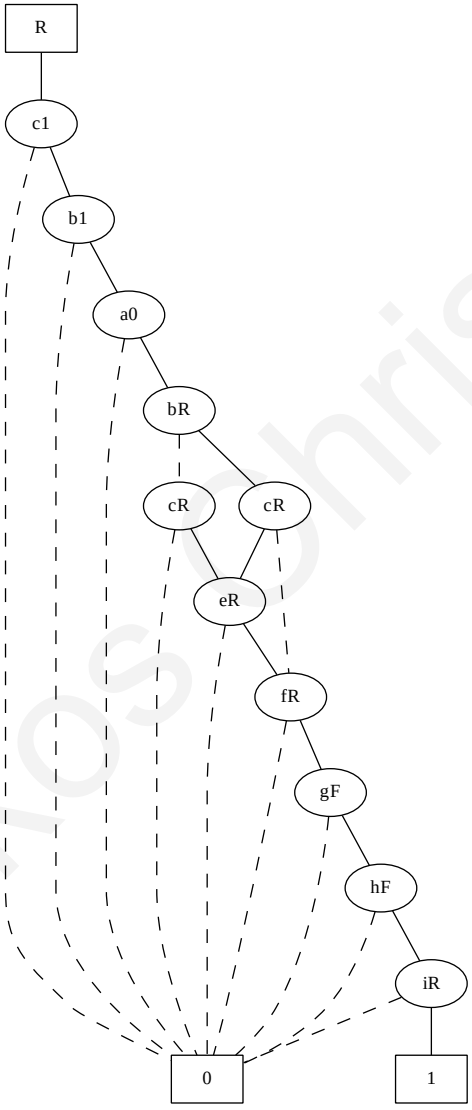



Figure 4.23: ISOP/ZBDD Graph for $R = P \star_6 Q$

*are the Rising(Falling) PDF segments that start from a specific circuit line.*

**Data**: ZBDD $P$,$n$;
**Result**: ZBDD $R$;
**if** $(P == \emptyset)$ **then return** $\emptyset$;
**if** $(P == 1)$ **then return** 1;
$P_{Top} \leftarrow$ Top Variable of $P$;
**if** $(P_{Top} == n)$ **then**
    **if** $(T(P) == \emptyset)$ **then** $R = \emptyset$;
    **else** $R = ZddGetNode(P_{Top}, T(P), NULL)$;
**else if** $(P_{Top} > n)$ **then** $R = \emptyset$;
**else**
    $Z_0 \leftarrow \wedge_n T(P)$;
    $Z_1 \leftarrow \wedge_n E(P)$;
    **if** $(Z_0 \neq \emptyset || Z_1 \neq \emptyset)$ **then** $R \leftarrow \cup(Z_1, Z_0)$;
    **else if** $(Z_0 \neq \emptyset)$ **then** $R \leftarrow Z_0$;
    **else if** $(Z_1 \neq NULL)$ **then** $R \leftarrow Z_1$;
    **else** $R \leftarrow \emptyset$;
**end**
**return** $R$;

**Algorithm 4.6:** Segment: $\wedge_n P$

The segment operator is illustrated through an example. Consider the ZBDD that contains all the circuit *PDFs* of circuit of Fig. 4.5. This ZBDD, denoted by $P$, is showed on Fig. 4.6. Fig. 4.24 shows all the *PDF* segments of the circuit beginning from line $f$ that have a rising transition on $f$ until the primary output has been reached. There are exactly 2 segments from line $f$ that have a rising transition and these are {fR.gF.iR, fR.hF.iR}. The resulting ZBDD, $R$, has been obtained by applying the segment operator on ZBDD $P$ and the number that $fR$ is denoted by, $R = \wedge_{Index(fR)} P$. Fig. 4.25 shows all the *PDF* segments of circuit $C$ from line $f$ that have a falling transition on $f$ until the primary output has been reached. There are 2 *PDF* segments, {fF.gR.iF, fF.hR.iF}, and have been obtained by applying the segment operator on ZBDD $P$ and the number that $fF$ is denoted by, $R = \wedge_{Index(fF)} P$.

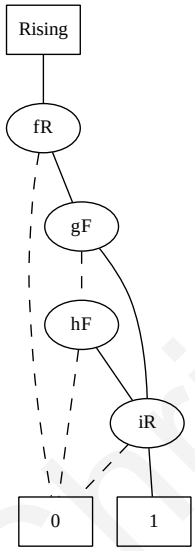



Figure 4.24: ZBDD from $f$ with Rising *PDF* segments of Circuit $C$

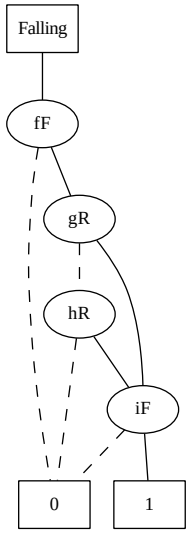



Figure 4.25: ZBDD from $f$ with Falling *PDF* segments of Circuit $C$

## 4.4 Experimental Results

The proposed methodology was implemented in C language on top of the *CUDD* package of [107] and the extra library of [76], and run on a 1GHZ SunBlade 1500 workstation with 4GB of memory. We experimented with the *ISCAS′*85 and the full-scanned versions of the *ISCAS′*89 benchmark circuits. The BDD variable ordering provided along with the tool of [107] was used, which is optimal for most of these circuits. The circuits were processed on a primary output basis, i.e., only the part of the circuit with paths terminating to a single primary output was considered at a time. All paths in a primitive *PDF* terminate at the same output and, thus, no fault is missed or double counted under this scenario. In this manner, the proposed method remains exact, while achieving considerable speedup since all primary output partitions can be processed in parallel. The total number of critical primitive *PDFs* is the sum of the critical primitive *PDFs* in each primary output partition.

The set of potentially testable and critical single *PDFs* ($\mathscr{C}$), needed as an input to the proposed method, was derived using some of the non-enumerative algorithms in [85], although any existing enumerative or non-enumerative method that determines potentially testable critical single *PDFs* can be utilized. Any set of such paths can be easily represented by a ZBDD, as discussed in [25], [84], [85]. The bounded delay model was considered, and the delay ranges per gate were obtained from the 0.18u TSMC library using the corner values for the nMoS and pMOS transistors. Criticality is determined by a user defined delay threshold $\mathscr{T}$, which is a % of the maximum path delay as determined by static timing analysis. Hence, a threshold of 80% will identify paths with delay greater than 80% of the maximum path delay. We have experimented for numerous thresholds between 20%-90%, but we only report the results for $\mathscr{T}$ ={90%, 95%} since are the most interesting ones. Potentially testable critical paths (these are paths that [85] cannot classify as critical or non-critical) were not included in order to consider a strictly critical path set. This is an implementation detail which has no impact on the conclusions drawn by the obtained results, especially since any delay model and critical path selection criterion and method can be used for this pre-processing step.

Tables 4.2 and 4.3 lists the obtained results. Column 2 reports the total number of single *PDFs* per circuit. Columns 3-6 and 7-10 list the results for delay threshold $\mathscr{T}$ equal to 90% and 95%, respectively. The number of the potentially testable critical single *PDFs* (obtained by [85], as explained above) under each delay threshold is given in Columns 3, and 7 (indicated by $|\mathscr{C}_{\mathscr{T}}|$). The number of critical single primitive *PDFs*, i.e., critical primitive *PDFs*

| Circuit Name | Single *PDFs* | $\mathscr{T} = 90\%$ of max delay | | | |
|---|---|---|---|---|---|
| | | $\lvert\mathscr{C}_{\mathscr{T}}\rvert$ [85] | $\lvert\Pi_1^{\mathscr{C}}\rvert$ | $\lvert\Pi^{\mathscr{C}}$-$\Pi_1^{\mathscr{C}}\rvert$ (*) | Single *PDFs* in (*) |
| b06.opt | 168 | 8 | 8 | 0 | 0 |
| s208.1 | 284 | 20 | 20 | 0 | 0 |
| s386 | 414 | 14 | 14 | 0 | 0 |
| s298 | 462 | 56 | 29 | 0 | 0 |
| s344 | 710 | 28 | 26 | 0 | 0 |
| s400 | 896 | 52 | 37 | 3 | 4 |
| s444 | 1.070 | 96 | 68 | 0 | 0 |
| b13.opt | 1.328 | 90 | 70 | 0 | 0 |
| b08.opt | 1.406 | 106 | 30 | 11 | 22 |
| b10.opt | 1.426 | 26 | 12 | 0 | 0 |
| b03.opt | 1.592 | 94 | 87 | 0 | 0 |
| s953n | 2.266 | 42 | 42 | 0 | 0 |
| b09.opt | 2.330 | 1.456 | 968 | 0 | 0 |
| s1512 | 6.972 | 318 | 195 | 0 | 0 |
| s991 | 14.920 | 224 | 120 | 0 | 0 |
| b11.opt | 15.024 | 1.904 | 28 | 6 | 9 |
| c880 | 17.284 | 2.338 | 2.086 | 0 | 0 |
| s5378 | 27.046 | 1.344 | 1.114 | 6 | 12 |
| b12.opt | 31.550 | 412 | 116 | 0 | 0 |
| s3271 | 38.388 | 1.704 | 1.040 | 72 | 68 |
| s3384 | 39.582 | 806 | 559 | 0 | 0 |
| b07.opt | 68.464 | 24.448 | 5.314 | 878 | 1.134 |
| s1269 | 79.140 | 6.104 | 2.672 | 0 | 0 |
| s1423 | 89.452 | 798 | 214 | 0 | 0 |
| b04.opt | 112.884 | 31.808 | 10.480 | 768 | 768 |
| s9234.1 | 489.708 | 76.032 | 0 | 8.992 | 4.192 |
| c2670 | 1.359.908 | 413.100 | 26.964 | 2.688 | 3.117 |
| c7552 | 1.452.988 | 302.592 | 7.000 | 2.712 | 2.888 |
| c1908 | 1.458.112 | 591.744 | 33.377 | 5.121 | 6.245 |
| c5315 | 2.682.610 | 505.456 | 15.824 | 11.104 | 8.704 |
| b05.opt | 2.822.046 | 1.686.784 | 10.799 | 9.760 | 14.482 |
| s4863 | 2.636.114.122 | 444.006.400 | 4.543.201 | 59.472 | 119.143 |
| s6669 | 431.685.738.673.270 | 374.177.550.827.520 | 102.529 | 613.325 | 130.416 |

Table 4.2: Results for delay threshold $\mathscr{T} = 90\%$.

| Circuit Name | Single *PDFs* | $\mathscr{T} = 95\%$ of max delay | | | |
|---|---|---|---|---|---|
| | | $\lvert\mathscr{C}_{\mathscr{T}}\rvert$ [85] | $\lvert\Pi_1^{\mathscr{C}}\rvert$ | $\lvert\Pi^{\mathscr{C}}$-$\Pi_1^{\mathscr{C}}\rvert$ (*) | Single *PDFs* in (*) |
| b06.opt | 168 | 8 | 8 | 0 | 0 |
| s208.1 | 284 | 12 | 12 | 0 | 0 |
| s386 | 414 | 2 | 2 | 0 | 0 |
| s298 | 462 | 2 | 1 | 0 | 0 |
| s344 | 710 | 6 | 5 | 0 | 0 |
| s400 | 896 | 8 | 4 | 0 | 0 |
| s444 | 1.070 | 36 | 21 | 0 | 0 |
| b13.opt | 1.328 | 36 | 20 | 0 | 0 |
| b08.opt | 1.406 | 32 | 6 | 5 | 9 |
| b10.opt | 1.426 | 10 | 3 | 0 | 0 |
| b03.opt | 1.592 | 22 | 19 | 0 | 0 |
| s953n | 2.266 | 8 | 8 | 0 | 0 |
| b09.opt | 2.330 | 480 | 240 | 0 | 0 |
| s1512 | 6.972 | 80 | 48 | 0 | 0 |
| s991 | 14.920 | 60 | 34 | 0 | 0 |
| b11.opt | 15.024 | 44 | 0 | 0 | 0 |
| c880 | 17.284 | 1.512 | 1.368 | 0 | 0 |
| s5378 | 27.046 | 792 | 788 | 2 | 4 |
| b12.opt | 31.550 | 34 | 1 | 0 | 0 |
| s3271 | 38.388 | 240 | 120 | 0 | 0 |
| s3384 | 39.582 | 224 | 128 | 0 | 0 |
| b07.opt | 68.464 | 276 | 46 | 2 | 4 |
| s1269 | 79.140 | 672 | 204 | 0 | 0 |
| s1423 | 89.452 | 72 | 36 | 0 | 0 |
| b04.opt | 112.884 | 1.680 | 524 | 48 | 48 |
| s9234.1 | 489.708 | 30.720 | 0 | 0 | 0 |
| c2670 | 1.359.908 | 88.128 | 5.904 | 672 | 672 |
| c7552 | 1.452.988 | 56.416 | 1.296 | 108 | 204 |
| c1908 | 1.458.112 | 19.328 | 0 | 0 | 0 |
| c5315 | 2.682.610 | 196.992 | 4.800 | 1.408 | 1.408 |
| b05.opt | 2.822.046 | 386.048 | 2.166 | 1.676 | 2.888 |
| s4863 | 2.636.114.122 | 52.435.456 | 11.498 | 1.594 | 3.188 |
| s6669 | 431.685.738.673.270 | 82.257.213.652.992 | 22.962 | 147.406 | 29.648 |

Table 4.3: Results for delay threshold $\mathscr{T} = 95\%$.

of cardinality 1 (indicated by $\Pi_1^{\mathscr{C}}$) is given in Columns 4, and 8. The difference between Columns 3-4 (also 7-8) gives the number of critical *PDFs* that are not single primitive and must be considered to identify the critical primitive *PDFs* of cardinality higher than 1 (these faults are also referred in the literature as singly or non-robustly untestable [21]). For example, for circuit s5378 and delay threshold $\mathscr{T} = 90\%$ (Columns 3-4), there are 230 singly untestable faults (1344-1114). For some circuits (s953.n and b06.opt), all faults are single primitive and, thus, there is no need to consider multiple faults. Circuit s6669 has a large number of single *PDFs* and thus we have abstracted the numbers in the table to fit in the columns. Circuit s6669 has a total of *431.685.738.673.270* single *PDFs* and for $\mathscr{T} = 90\%$ the number of critical *PDFs* equals to *374.177.550.827.520* and for $\mathscr{T} = 95\%$ the number of critical *PDFs* equals to *82.257.213.652.992*.

Columns 5, and 9 list the number of critical primitive PDFs of cardinality higher than 1 identified by the proposed method (indicated by $|\Pi^{\mathscr{C}} - \Pi_1^{\mathscr{C}}|$). Hence, the total number of critical faults needed to be tested ($|\Pi^{\mathscr{C}}|$) is the summation of the faults in Columns 4 and 5, for the case of $\mathscr{T} = 90\%$ (similarly for the other values of $\mathscr{T}$). In the reported experiments we bounded the maximum cardinality of a primitive fault to 9 in order to maintain reasonable execution times. Thus, $\Pi^{\mathscr{C}}$ here is actually $\sum \Pi_i^{\mathscr{C}}, i = 1 \cdots 9$. This is a reasonable bound since it is unlikely that more than 9 critical paths [105], each of them having delay faults that violate the maximum circuit delay, can be co-sensitized. This is also supported by the fact that the majority of the primitive faults identified in set $\Pi^{\mathscr{C}} - \Pi_1^{\mathscr{C}}$ have cardinality 2 or 3. Nevertheless, the proposed method is exact, thus, critical primitive faults of any cardinality can be identified in the expense of additional computation time. As expected, the number of faults in $\Pi^{\mathscr{C}} - \Pi_1^{\mathscr{C}}$ increases as the delay threshold $\mathscr{T}$ decreases, since the number of paths considered critical increases. However, in all circuits $|\Pi^{\mathscr{C}} - \Pi_1^{\mathscr{C}}|$ remains small which means that adding a small number of tests for the faults in $\Pi^{\mathscr{C}} - \Pi_1^{\mathscr{C}}$ can increase the quality of the delay test set since necessary critical multiple (i.e., primitive) faults will also be tested. Columns 6 and 10 report the number of critical singly untestable faults participating in the formation of at least one multiple fault of those reported in Columns 5, and 9, respectively. For example, consider again circuit s5378 and delay threshold $\mathscr{T} = 90\%$. Out of the considered 1344 potentially testable critical single *PDFs*, 1114 are single primitive and 12 can be tested under the multiple fault criterion by participating in at least 1 of the 6 multiple faults. The remaining 218 potentially testable critical single *PDFs* (1344-(1114+12)) are identified as *PDFs* that do not need to be tested under the primitive fault conditions.

Table 4.4 reports the CPU time of the proposed method. We observe that time does

| Circuit Name | $\mathscr{T} = 90\%$ | ATPG | $\mathscr{T} = 95\%$ | ATPG | ATPG (Avg) |
|---|---|---|---|---|---|
| b06-opt | 0,00 | 0,00 | 0,00 | 0,00 | 0,00 |
| s208.1 | 0,13 | 0,01 | 0,02 | 0,01 | 0,01 |
| s386 | 0,91 | 0,01 | 0,88 | 0,00 | 0,01 |
| s298 | 2,24 | 0,02 | 0,98 | 0,00 | 0,01 |
| s344 | 0,70 | 0,03 | 0,28 | 0,01 | 0,02 |
| s400 | 3,11 | 0,04 | 0,93 | 0,01 | 0,03 |
| s444 | 1,23 | 0,02 | 0,54 | 0,01 | 0,02 |
| b13-opt | 1,68 | 0,02 | 0,12 | 0,01 | 0,02 |
| b08-opt | 0,00 | 0,02 | 0,00 | 0,01 | 0,02 |
| b10-opt | 0,01 | 0,01 | 0,00 | 0,01 | 0,01 |
| b03-opt | 0,51 | 0,02 | 0,20 | 0,01 | 0,02 |
| s953n | 0,67 | 0,09 | 0,59 | 0,07 | 0,08 |
| b09-opt | 0,08 | 1,01 | 0,01 | 0,22 | 0,62 |
| s1512 | 0,14 | 0,86 | 0,05 | 0,02 | 0,44 |
| s991 | 0,12 | 0,08 | 0,07 | 0,01 | 0,05 |
| b11-opt | 0,22 | 0,90 | 0,02 | 0,01 | 0,46 |
| c880 | 5,54 | 3,21 | 0,96 | 1,52 | 2,37 |
| s5378 | 4,21 | 1,31 | 3,11 | 1,21 | 1,26 |
| b12-opt | 0,05 | 0,04 | 0,00 | 0,01 | 0,03 |
| s3271 | 0,96 | 1,20 | 0,54 | 0,54 | 0,87 |
| s3384 | 0,07 | 0,77 | 0,01 | 0,45 | 0,61 |
| b07-opt | 1,33 | 1,25 | 0,21 | 0,02 | 0,64 |
| s1269 | 0,88 | 1,02 | 0,20 | 0,02 | 0,52 |
| s1423 | 4,41 | 3,11 | 1,01 | 0,18 | 1,65 |
| b04-opt | 0,21 | 2,22 | 0,05 | 0,82 | 1,52 |
| s9234.1 | 3,21 | 3,00 | 1,01 | 0,02 | 1,51 |
| c2670 | 110,32 | 24,69 | 23,24 | 9,51 | 17,10 |
| c7552 | 299,18 | 15,65 | 57,21 | 9,43 | 12,54 |
| c1908 | 1.750,19 | 145,65 | 304,22 | 28,96 | 87,31 |
| c5315 | 46,84 | 55,21 | 11,81 | 11,23 | 33,22 |
| b05-opt | 18,36 | 30,21 | 1,97 | 10,22 | 20,22 |
| s4863 | 250,33 | 53,20 | 44,21 | 11,02 | 32,11 |
| s6669 | 350,21 | 19,52 | 74,21 | 12,04 | 15,78 |

Table 4.4: CPU time(secs) requiremnts for the proposed method.

not necessarily increase with the size of the circuit or the number of the considered paths. For example, circuit s4863 and s6669 are considerably larger, in terms of number of gates and number of faults considered, than circuit C1908, but its required time are considerably lower of that of C1908. This occurs because the time complexity of the proposed method depends on the size of the underlined ZBDDs, which in turn depend on the structure and functionality of the circuit rather than its size. Column 3,5 and 6 (averaged over the two different thresholds),shows the time needed to generate a test set that detects all single and multiple critical faults in $\Pi^{\mathscr{C}}$ using simple linear traversals on the generated ZBDD structure in a manner similar to that proposed in [25].

An important conclusion can be extracted by examining the experimental results. The proposed method can be scalable to very large circuits. The methodology is fast if the depth of the circuit under test is not excessively long. This can be explained by examining the size of the underlined ZBDDs. In short circuits the ZBDD constructed has a lot of sharing and its really compact. Thus the size of the ISOP/ZBDD graph examined is small. Nowadays, industrial circuits tend to have a huge number of gates with a short depth.

## 4.5 Conclusions

The testable critical primitive *PDF* set is defined and computed in a non-enumerative fashion for any potentially testable critical *PDF* set. It is shown that the number of critical multiple faults is a very small, compared to the number of single critical faults, and therefore only a small number of additional test patterns is needed to guarantee a circuit's timing correctness.

# Chapter 5

# Generation of Functional Programs to Test Path Delay Faults within Microprocessors Cores

## 5.1 Introduction

In order to guarantee product quality for todays microprocessor cores, traditional stuck-at tests are no longer sufficient and more complex fault models have to be considered when devising test strategies. At-speed delay fault testing, in particular, has been widely addressed by academia and is becoming common practice in industry [69], [61], [17], [49]. Among all existing delay fault models, the path-delay fault model is considered the most accurate since it can detect both lumped and distributed delays [17], [55], but also the most challenging, due to the enormous number of faults (paths).

Delay test has been approached adopting different strategies, purely relying on an external tester or applying structural self-testing methodologies such as Built-In Self-Test (BIST), or exploiting the execution of suitable self-test programs. The latter strategy is usually referred to as Software-Based Self-Test (SBST) and is generally more affordable, as it exploits the processor instructions in the normal mode of operation; it can be used in stand-alone modules as well as when the processors are deeply embedded in a System on Chip (SoC) and their accessibility is reduced.

Regarding test generation addressing path-delay faults, several techniques exist for enhanced full-scan circuits, based on either structural ATPG tools [32], [112] or function based tools using Binary Decision Diagrams (BDDs) [10], [74], [13] and Boolean-SAT [19], [117]

implementations. Some work on software-based test generation has been done exploiting deterministic techniques [104], [59], [35], [58], [18]. If a microprocessor design is tested using its functional vectors, i.e., using instruction sequences [58], [18], [59] the input signals to the embedded block of the processor are derived by their functionality. Evolutionary algorithms, are population-based searching algorithms that mimic natural evolution and exploit the population of individuals to simultaneously evolve solutions. Evolutionary algorithms have been successfully exploited for the automatic generation of program sets for verification, test [29], and diagnosis [95] for processors described at different levels of abstraction. In most cases, the evolutionary algorithm faces the test set generation as a single-objective optimization problem, e.g., resorting to a multi-run strategy. However, hardware optimization techniques belong to a real-world classification of problems that usually require the simultaneous optimization of many objectives. Therefore, hardware optimization problems could be addressed resorting to multiobjective optimizers. Multiobjective Evolutionary Algorithms (MOEAs) were initially introduced in 1985, by the implementation of the first evolutionary algorithm dealing with multiobjective optimization problems [100]. Roughly speaking, MOEAs produce a set of potentially optimal solutions, rather than a unique solution, that represents a subset of the Pareto optimal set.

This chapter presents an innovative approach for the automatic generation of path delay functional test programs for microprocessors exploiting both gate- and RT-level descriptions. The former is used to select the set of critical paths to be considered and to obtain path excitation requirements based on BDD analysis; the latter is used for effectively identifying the test programs able to reproduce the conditions activating the targeted fault (excitation), and to make the fault effect(s) visible on the processor outputs (propagation). For automatically generating test programs, the new implementation of an evolutionary algorithm addressing multi objective optimization is employed. The main advantage introduced is the improvement in the flow performances compared to other approaches based only on gate-level simulation [9]. Section 5.2 provides the needed background/preliminaries. Section 3 gives in detail description the proposed methodology used while Section 4 presents a study case. Finally, in Section 5 concludes the chapter.

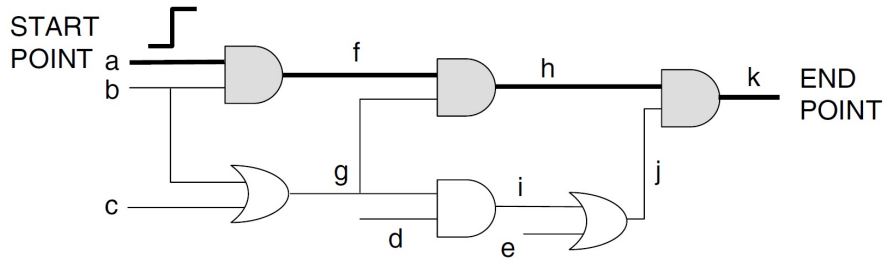



Figure 5.1: PDF Example:on(off)-path signals given in thick(normal) lines $(a, f, h, k)(b, g, j)$.

## 5.2 Preliminaries

### 5.2.1 Software-Based Path Delay Testing

A path-delay fault occurs when a defect in a circuit causes the cumulative delay of a combinational path to exceed some specified duration [55], [16]. The combinational path begins at a primary input or a clocked flip-flop (startpoint), includes a connected chain of gates, and ends at a primary output or a clocked flip-flop (endpoint) (see Fig. 5.1). The specified time duration can be the duration of the clock period (or phase), or the vector period. The propagation delay is the time that an event (i.e., a transition) takes to traverse the path. For each combinational path in a circuit, there are two path-delay faults, corresponding to rising and falling transitions on the startpoint. Signals that compose the path and feed the traversed gates are called on-path signals; signals that are not on the path but feed the gates on the path are called off-path signals.

In order to examine the timing operation of a circuit we should examine signal transitions: delay tests consist of vector pairs ($V_1 \rightarrow V_2$) to be applied on the inputs feeding the path ($a$, $b$, $c$, $d$ and $e$ in Fig. 5.1), so that an input transition on the startpoint propagates to the endpoint.

Path-delay test application can be performed resorting to suitable scan-chains or by employing functional techniques. In scan-based test methodologies, the patterns are serially loaded into the scan chains (at reduced speed if necessary). Consequently, the two test vectors are applied in succession with a defined timing and the test results are shifted out through the scan chains, thus achieving full observability. In the case of Software-Based path-delay testing the test vectors $V_1 \rightarrow V_2$ reach the targeted path inputs during the normal at-speed circuit operations, hence depending on the sequence of data feed (instructions in case of processors) and allowing continuous application of test vectors. When targeting microprocessors, a test program must be made to ensure that the excitement conditions of the targeted path-delay fault are met in a consecutive pair of clock cycles, and that the fault effect(s)

propagate to suitable observable points (e.g., output ports).

If a test can be applied in the normal operations of a circuit, we refer to it as a functional test. A path is functionally testable if there exists a functional test for that path. Otherwise, the path is functionally untestable [59]. Functionally untestable faults never determine the performance in normal operations of the circuit, and if detected during testing may lead to overkill (i.e., discarding functioning chips). On the other hand, defects on functional testable paths may degrade the circuit performance when path-delay faults occur. Software-based testing concentrates on the latter class, intrinsically avoiding over-testing redundant paths.

### 5.2.2 Exploiting Gate and RT Level Descriptions for Path Delay Testing

Commonly adopted solutions for path-delay test generation in sequential circuits are mostly based on the analysis of gate-level descriptions. Addressing a fault list provided by timing analysis tools, test patterns for path excitation are calculated. At this phase it is seldom possible to assess whether the faults are functionally testable. The test patterns correspond to two consecutive vectors to be applied at speed to the inputs of the combinational circuit partition including the selected path. From this point forward, they will be referred as $V_1$ and $V_2$.

When dealing with functional test (in the absence of scan structures) $V_1$ and $V_2$ are functionally justifiable if and only if they can be consecutively reproduced on the memory elements and primary inputs feeding the path by a sequence of instructions and data. In this case, the processor RT-level description may be employed to establish whether an instruction sequence is able to apply $V_1$ and $V_2$ to the selected combinational part. Since the observation of flip-flop values is required, only, it is possible to relate each considered flip-flop in the gate-level description to a signal in the RTL one.

### 5.2.3 BDDs for Structural Path Delay Fault Tests

Rather than devising a specific couple of vectors $V_1$ and $V_2$ that excite a specific fault, through BDD analysis of the gate-level netlist it is possible to derive a wider set of requirements for the combinational sub-circuit inputs to excite the path it contains.

A reduced ordered Binary Decision Diagram (referred to as a BDD here) is a canonical graphical representation of a Boolean function [13]. BDDs have been widely used in test generation, for various fault models. For the case of path-delay faults in enhanced scan designs [10], [74], [85], given one (or more) fault(s) a Boolean function can be formulated

whose solution space is all the possible pairs of test vectors that can detect the fault(s). This function is derived based on all the necessary values on on-path and off-path signals of the path-delay fault(s). The variables of the function correspond to the primary inputs of the circuit. When such a function is given by a BDD, we have a very compact (due to the suppression of variables with the $x$ value) and implicit (non-enumerative) representation of the entire solution space. This is of high importance for several issues in test generation: untestable faults are very easily determined; hard-to-detect faults, that require a lot of time in structural-based ATPG tools, are also efficiently handled (BDD is very small since it contains a small number of cubes); fault simulation, for fault dropping, can be trivially performed on the BDD and not on the gate-level netlist. Moreover, if an input pattern is not a valid test, the BDD can be used to quickly determine how far the input pattern is from becoming a valid test (% of bits that must be changed in the input pattern). The latter is of particular importance in the proposed methodology, since it can quickly and accurately guide the evolutionary engine to generate the necessary path-delay fault tests.

### 5.2.4 Basic Concepts on MOEAs

Multiobjective evolutionary algorithms, as their single-objective counterpart, are population-based searching algorithms that mimic natural evolution. However, differently from single-objective algorithms, MOEAs exploit the population of individuals to simultaneously evolve solutions to multiple and usually conflicting goals [28], [43]. The expected result from a MOEA is a set of trade-off individuals called nondominated solutions, Pareto-optimal solutions, or Pareto optimal set. For each individual into the population, a fitness vector $f_i = (x_1, x_2, x_n)$ represents the figures of merit obtained by the individual regarding to the n pursued objectives.

Pareto optimality is defined using the concepts of domination: given two individuals A and B, A dominates B if and only if A is at least as good as B in all objectives, and better in at least one. A is equivalent to B iff results on A and B are identical in all objectives. A covers B if A either dominates or is equivalent to B. Similarly, given two sets of individuals Y and Z, Y dominates Z if every individual of Z is dominated by some individual of Y. Similar definitions relative to sets of individuals can be made for equivalence and coverage concepts. Thus, the Pareto optimal set is the set of all Pareto optimal individuals, and the corresponding set of fitness vectors is the Pareto optimal front. Individuals belonging to the Pareto optimal set are equally important. Indeed, for the individuals belonging to the Pareto

optimal set, no improvement is possible in any objective without harming at least one of the other objectives.

Different strategies have been proposed in order to properly sort individuals belonging to the population; for example: aggregation-based approaches, lexicographical ordering, target-vector approaches, criterion-based approaches, and Pareto-based approaches. Some of them do not incorporate directly the concept of optimality outlined before, whereas others not only exploit it but include additional mechanisms to guarantee the diversity of the population. One of the most popular strategies used by MOEAs is based on a ranking scheme that divides the whole population on different sets, in such a way that each set contains only non-dominated individuals, and lower ranked sets are dominated by higher ones [28]. It is interesting to highlight that in a successful experiment the highest set contains the individuals belonging to the Pareto optimal set.

## 5.3 The Proposed Approach

The proposed methodology is a low-cost generation procedure that exploits both the gate and RT-level description of the processor. On the gate level description, it retrieves information about testability of paths and performs a BDD analysis to derive the path (fault) excitation requirements. On the RT-level description, it performs the heaviest part of the process, which consists in generating and evaluating instruction sequences. The generation process includes four main steps:

- Path list grouping: this is a preliminary step for minimizing the cost for the generation step that will follow; this phase analyzes the processor path list provided by timing analysis tools and produces a set of shorter fault lists, each one corresponding to a coherent set of critical paths in the circuit, i.e., a set of paths related to the same processor elements. As a matter of fact, excitation conditions for faults belonging to the same structurally coherent fault group are likely to be stressed by the same instructions. Details on this topic can be found in [9].

- Circuit subdivision and BDD analysis: Given the gate-level netlist and the addressed path list, for each path a combinational subcircuit (or chunk) is automatically extracted, which contains the path and, therefore, all the information needed for the analysis of its excitation conditions. A BDD is then derived that contains all the possible input vectors that bring necessary excitation values at the inputs of the path under considera-

tion. Structurally untestable faults are removed in this phase. The BDD representation will be used in the sequential fault excitation step for evaluating the ability of each program to excite specific faults: the fitness function depends on the minimum hamming distance of the vectors applied from the set of vectors that can excite the path. It can be computed optimally and quickly when the set of vectors is represented by a BDD.

- Sequential Fault excitation: this step aims at generating the test programs that effectively excite the considered PDFs. An evolutionary algorithm is exploited to automatically generate instruction sequences, whose fitness is evaluated through RT level simulation, avoiding highly expensive gate level simulations, and relying on the already available BDD representations. A MOEA is exploited to automatically generate instruction sequences, whose fitness is evaluated through RT-level simulation, avoiding highly expensive gate-level simulations, and relying on the already available BDDs. This step will be analyzed in detail.

- Sequential Error propagation: this step targets error propagation to the processor output ports and uses an evolutionary algorithm implementing a single-objective strategy. For this task, during the RTL simulation of the test program execution, the values of the flip-flops feeding the investigated path are analyzed at each clock cycle in order to check for the excitation conditions (both on on-path and off-path); whenever they are met, a faulty value is forced on the path endpoint for one clock cycle (fault injection, [9]). From that point in time, the state of all flip-flops is saved at each clock cycle and compared to the original (fault-free) simulation: if the simulation of the already generated program on the sabotaged RTL introduces a change on the processor output ports at any time following the fault injection, the test program achieves excitation and observation of the addressed fault and is complete. Otherwise, the number of flip-flops with different contents with respect to the fault-free simulation is used as a fitness function to be maximized, until the fault effects are propagated to the outputs.

This preliminary step aims at directing and minimizing the costs for the successive phases. Differently from the approaches in [19], [35], [95], information about the processor structure that can help in the instruction sequence calculation is not derived from the RT-level description, but is indirectly obtained from the path list produced by a timing analysis tool on the processor gate-level netlist. This list is demonstrated to contain the complete set of functionally testable PDFs [59].

A fault classification process decides if two PDFs belong to the same functionally coherent fault group, based on the following criteria:

- the startpoints of the two considered paths are flip-flops in the same register,
- the off-path conditions for enabling the PDF activation are structurally justified by flip-flops values in the same set of registers,
- the endpoints of the two considered paths are flip-flops in the same register.

The purpose of this process is to concentrate the generation efforts on functionally correlated processor areas. As a matter of fact, excitation conditions for faults belonging to the same structurally coherent fault group are likely to be stressed by the same instructions.

### 5.3.1 BDDs for path delay fault excitation

Given the processor netlist and the addressed path list, for each path a combinational sub-circuit (or chunk) is automatically extracted, which contains the path and, therefore, all the information needed for the analysis of its excitation conditions. This sub-circuit primary inputs correspond to the original circuit primary inputs and flip-flops that feed the path (being all and only the ones which affect its behavior) and its only output signal is the path endpoint. Fig. 5.1 gives the circuit chunk for the targeted path $a-f-h-k$. Signals $a$, $f$, $h$, $k$ are on-path signals and signals $b$, $g$, $j$ are off-path signals.

Deriving a BDD that contains all the possible pairs of vectors that bring necessary excitation values at the on-path and off-path signals of a PDF (described in a chunk) has been addressed in [12][13], for singly-testable (robust and non-robust) as well as multiply testable path delay faults. Fig. 5.2 gives the BDD containing all possible non-robust pairs of vectors for exciting a rising fault through the targeted path of Fig. 5.1, denoted by $\uparrow a-f-h-k$. This BDD corresponds to the function given below:

$$\begin{aligned} C_{\uparrow a-f-h-k}(a1,a,b,c,d,e) &= \overline{a1}.(a.f_b.f_g.f_j) \\ &= \overline{a1}.a.b.(d+e) \end{aligned} \tag{5.1}$$

Here, $f_i$ denotes the functionality of an off path signal $i$, with respect to the input variables. The first parenthesis contains the necessary conditions for vector $V_1$, which only require a 0 value on the path input, whereas the second one gives all the necessary conditions for $V_2$, which require a 1 value on the path input and a non-controlling value on the off input

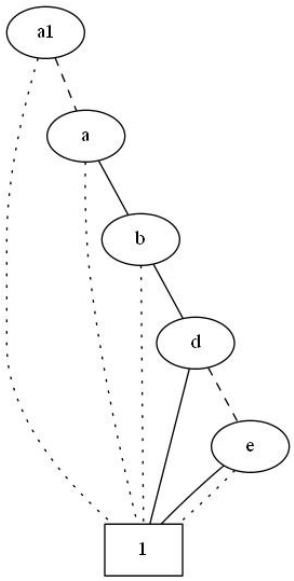



Figure 5.2: BDD for NRS excitation requirement for PDF $\uparrow a-f-h-k$ of the PDF example depicted on Fig. 5.1.

signals. The BDD of Fig. 5.2 contains two cubes $\{\overline{a1}.a.b.d, \overline{a1}.a.b.e\}$, which represent all input vector pairs that excite the fault (the $x$ implies a don't care value):

$(V_1(a_1,x,x,x,x) \rightarrow V2(a,b,c,d,e)) = \{(0xxxx) \rightarrow (11x1x), (0xxxx) \rightarrow (11xx1)\}$.

Given a specific vector pair $(V_1 \rightarrow V_2)$, the BDD for a PDF can be used to quantify its ability to excite a fault,corresponding to finding the % of bits whose value must be changed (flipped) in order for $(V_1 \rightarrow V_2)$ to become a solution in the BDD.

Consider $(V_1 \rightarrow V_2) = (00000 \rightarrow 11x01)$ and the BDD of Fig. 5.2. Let $V = (a1, b1, c1, d1, e1, a, b, c, d, e) = 0000011x01$ be the concatenation of $V_1$ and $V_2$. Then, $V$ satisfies all necessary excitation requirements of $\uparrow a-f-h-k$ if and only if $C_{\uparrow a-f-h-k}(V) = 1$, i.e., if and only if $V$ is a solution of $C_{\uparrow a-f-h-k}()$. This is a very simple operation on a BDD, performed in linear time to its size. This operation corresponds to fault simulating $V$, however it is performed on the BDD and not on the gate-level netlist. For the given V, $C_{\uparrow a-f-h-k}(0000011x01) = 1$ which means that $V$ is a good vector pair for $\uparrow a-f-h-k$. Observe that any missing variable from the BDD (such as $b1$, $c1$, $d1$ and $e1$) is considered a don't care. We say that V has a 100% fitness. The fitness of $V$ gives the % of bits in $V$ that have a correct value.

Consider now $(V1 \rightarrow V2) = (00000 \rightarrow x0001)$. In this case, $C_{\uparrow a-f-h-k}(V) = 0$ indicating that $V$ is not a good choice. Under the proposed framework, $V$ is derived from the execution of a test program and is evaluated at each clock cycle. The fitness value of $V$ is an important parameter that can help the evolutionary engine proceed towards a correct solution

(i.e., a test program that excites the targeted PDF). Determining the minimum number of bits in $V$ that are incorrect (must change value) amounts to finding the minimum hamming distance of $V$ from all 1-cubes represented by the BDD. The 1-cubes of the BDD of Fig. 5.2 are $\{\overline{a1}.a.b.d, \overline{a1}.a.b.e\}$. A don't care value ($x$) in V is considered an incorrect value, since the evolutionary engine has not yet determined the value. On the other hand, an $x$ value in a BDD cube (indicated by the absence of a variable in the cube) does not give a bit mismatch since it can be set to either value 1 or 0. Thus, the minimum hamming distance of $V = 00000x0001$ is $min\{3,2\} = 2$ (3 bit mismatches for bits a, b and d in cube $\overline{a1}.a.b.d$ and 2 for bits a and b in $\overline{a1}.a.b.e$). The fitness value of $V$ is therefore 80% (there are only 2 out of 10 bit mismatches).

The minimum hamming distance of a vector from a set of vectors, as defined above, can be computed quickly when the set of vectors is represented by a BDD. The operation amounts to a systematic and linear examination of the BDD edges, similar to the standard $ite()$ BDD recursive operator, (Algorithm 5.1). The overall complexity is linear to the size of the given BDD vectors. The fitness value is then computed by:

$$fitness\_value = ((nbit\_error())/n) * 100, \quad where\ n = |V|. \tag{5.2}$$

### 5.3.2 Sequential Fault Excitation

The purpose of the sequential fault excitation phase (Fig. 5.3) is the generation of suitable instruction sequences that excite the path-delay faults in coherent lists. This process is based on the usage of a new implementation of a well known evolutionary algorithm (EA), called $\mu GP3$ [121], able to automatically generate suitable test programs.

Roughly speaking, an EA is a population-based optimizer that imitates the natural process of biological evolution. Following this perspective, a test program is an individual and the tool handles a population of individuals (i.e., a collection of assembly programs). The initial population is generated randomly, then iteratively refined mimicking the Darwinian Theory: new individuals are generated either by mutation (an individual is slightly modified) or by recombination (two or more individuals are mixed in some way); the best performing individuals are selected for survival. The process is blocked after a certain number of steps, called generations, or when a steady state is reached. The best individual is eventually provided as output.

Differently from the standard approach described in [69], the evolutionary tool imple-

**Data**: $vectorV[1..n], BDDC_{pdf}, vectorH[1..n]$;

**Result**: $H[i]$;

1 % $bit(BDD(f,V,i)) =$ bit value of $V$ for variable $i$ of BDD $f$;

2 % $H[1..n]$ initialized to -1 before the first call of bit_error ;

3 $i =$ Top Variable of $C_{pdf}$;

4 **if** *(*$i == Constant\_Zero$*)* **then return** $\infty$;

5 **if** *(*$i == Constant\_One$*)* **then return** 0;

6 % Solution Found ;

7 **if** *(*$H[i] \neq -1$*)* **then return** $H[i]$;

8 % $H[i]$ already computed ;

9 **if** *(*$bit(C_{pdf},V,i) == 0$*)* **then**

10 $d_0 = bit\_error(V,,Else(C_{pdf}),H)$;

11 $d_1 = bit\_error(V,,Then(C_{pdf}),H)+1$;

12 **end**

13 **if** *(*$bit(C_{pdf},V,i) == 1$*)* **then**

14 $d_0 = bit\_error(V,,Else(C_{pdf}),H)+1$;

15 $d_1 = bit\_error(V,,Then(C_{pdf}),H)$;

16 **end**

17 **if** *(*$bit(C_{pdf},V,i) == x$*)* **then**

18 $d_0 = bit\_error(V,,Else(C_{pdf}),H)+1$;

19 $d_1 = bit\_error(V,,Then(C_{pdf}),H)+1$;

20 **end**

21 $H[i] = min\{d_0,d_1\}$;

22 **return** $H[i]$;

**Algorithm 5.1:** Pseudocode for procedure bit_error

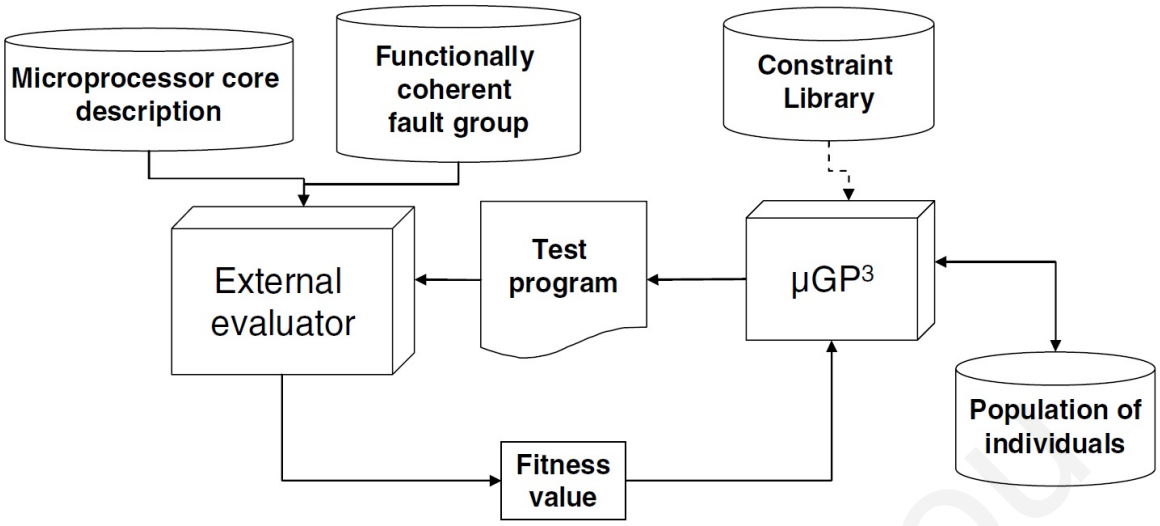



Figure 5.3: Sequential fault excitation phase.

ments a MOEA [28] able to deal with several path-delay faults at a time. In this case the main goal of the evolutionary process is not to obtain a single best program but a set of best programs able to correctly excite the targeted faults. The main idea behind the MOEA implementation of $\mu GP3$ is to simultaneously optimize a complete functionally coherent group. As mentioned before, faults belonging to the same structurally coherent fault group are probably excited by similar test programs. Thus, the MOEA will evolve a population of individuals working on a specific portion of the processor core rather than a single program focusing on a unique fault.

Algorithm $\mu GP3$ bases its evolutionary process on a constrained tagged graph, which is a directed graph whose elements may own one or more tags, and that in addition has to respect a set of constraints. The constraints may affect both the information contained in the graph elements and its structure. Graphs are initially generated in a random fashion; subsequently, they may be modified by genetic operators (e.g., the classical mutation and recombination, but also by different operators, as required; the tool architecture has been specially thought for easy addition of new genetic operators).

The purpose of the constraints is to limit the possible productions of the evolutionary tool, and also provide them with semantic value. The constraints are provided through a user-defined library that provides the genotype-phenotype mapping for the generated individuals, describes their possible structure and defines which values the existing parameters (if any) can take. Constraint definition is left to the user to increase the generality of the tool; it is flexible enough to allow the definition of complex entities to easily describe a wide range of

processor instruction sets architecture (ISA).

The evolutionary core reads the constraint library in order to adequately generate assembly programs. For each generated program, a vector of fitness values are computed by the external evaluator considering the targeted faults provided by the functionally coherent fault list. Differently from the classical approach, the sequence of values in the fitness vector does not represent a priority list but each of them describes the figure of merit obtained by the individual regarding to a specific fault.

The task of the $\mu GP3$ core is to progressively improve the population of individuals or test programs. Thus, the population is ordered following a ranking strategy based on the Pareto-dominance principles described before. Choice of the individuals for reproduction is performed by means of a tournament selection based on the ranking position. However, since individuals belonging to the same group are by definition non-dominated ones, the selection is performed resorting to the delta entropy value of the individual [29]. The purpose of the entropy value is not to rank a population in absolute terms, but to detect whether the amount of genetic diversity in a set of individuals is increasing or decreasing. The tournament size is also endogenous. The population size is set at the beginning of a run, and the tool employs a variation on the plus $(\mu + \lambda)$ strategy: a configurable number $\lambda$ of genetic operators are applied on the population. Since different operators may produce different number of offspring, the number of individuals added to the population is variable; the activation probability and strength for every operator is an endogenous parameter. All new unique individuals are then evaluated, and the population resulting from the union of old and new individuals is ordered resorting to the ranking approach described previously. Clearly, if a new individual dominates the complete population, a new individuals set is created and it is placed at the top of the rank list. Finally, only the first $\mu$ individuals are kept.

In order to customize this architecture to the specific goal we address here, we use the BDD-based fitness function described above, which is effective in guiding the algorithm towards the solution, and can be computed in reasonable times.

In this case, the evaluation of the generated test programs (or instruction sequences) is performed on the RT-level microprocessor core description by means of a logic simulation: during the simulation, at each clock cycle the vectors feeding the path are passed to the fitness function, and the maximum value obtained during the program run identifies the program's fitness.

## 5.4 Experimental Results

The methodology proposed has been evaluated on a 8051 microcontroller description, addressing non-robust path delay fault testing. The processor reads the test programs from an external memory and its output ports are directly accessible. The critical timing analysis of the synthesized architecture has been performed utilizing the Synopsys PrimeTime suite ver. X-2005.12.

A total number of 92,430 worst paths were selected (based on an in-house developed library). For each path delay fault, a combinational sub-circuit was extracted automatically from the circuit. The captured path delay fault is then represented by a BDD among with all sensitizing test cubes. If such a BDD representation does not exist this implies an un-testable fault and thus it is dropped.

The number of the structurally testable path delay faults is 10.394. These faults have been divided in classes depending on their structural coherence in an automated way, using a simple tool based on set covering principles and obtaining 96 coherent fault lists, each one including an average of about 108 faults.

The sequential fault excitation step has been performed resorting to the new MOEA implementation of $\mu$GP3 [24], which also includes a new operator called local-scan mutation, whose purpose is the generation of a reduced set of individuals in the neighborhood of the selected parent by performing slight mutations to only one determined parameter. In this case the fitness evaluator comprised a commercial logic simulator (Mentor Graphics ModelSim v.6.2h) and an ad-hoc C-language software monitor implemented in the simulator environment. The evolutionary experiment has been set up with the aim of performing a multi-objective optimization. The initial population is composed of 300 random individuals; the population size is 100 and at each generation 80 genetic operators are applied. For each of the coherent fault lists, the evolutionary experiment was set up in the following manner:

- the first 20 faults in the list are initially considered (in order not to slow excessively the simulation, not all faults in the list are addressed together) and the EA is started, evaluating the excitation fitness (20 paths implies 20 fitness values)
- whenever a test program fitness hits 100% for one of the inspected faults, that fault is removed from the experiment and replaced from a new one from the same list (fault dropping strategy). The obtained test program is saved.
- if the algorithm does not improve the fitness for a set number of generations (10 in this

case), the 20 paths are replaced with the following 20 in the list.

The process continues until all paths in the coherent fault list have been considered. This phase took about 110 hours for the whole fault list. The error propagation step took about 35 hours. The fitness has been evaluated resorting to the ModelSim simulator running a script performing fault injection and to an ad-hoc tool elaborating the simulation dump. The majority of the test program set achieves test observability without modification; for the ones whose fault effects are still not propagated, the EA modifies the original test program maximizing the observability fitness, making sure that the excitation conditions are still met.

The obtained coverage values (Table 5.1) are comparable to the ones obtained using other approaches [14][19]. It must be noted that not-covered faults include functionally untestable ones, which do not determine the circuit performances and cannot be tested functionally. The required time computation compares favourably with the time required in [19]. The experiments run on an Intel E6400 @2.13 GHz.

In order to detail the behavior of the approach, the following pictures describe the evolution of an experiment targeting one coherent fault list that contains 84 faults. Fig. 5.4 shows the first 300 steps of the evolutionary process: the continuous dark line represents the average of the 20 considered fitness values (mean value on the population), while vertical bars indicate the maximum fitness obtained at each step. For this coherent fault list, the final coverage is 50%. It is important to notice that whenever excitation is found for a fault (e.g., step 28), the average fitness falls down due to the fault dropping strategy. Similarly, this average value undergoes a big depression each time the steady state is reached and all targeted faults are replaced (steps 68, 118 and 189). Nevertheless, the average fitness tends to increase along the experiment. Fig. 5.5 shows the first 50 steps of the same experiment; in this case, 5 out of the 20 evaluated fitness values are shown (average values on the population). Fitness 5 and 7 show that when a 100% is found the fitness value decreases, due to the substitution of the path-delay fault under inspection; however, the other fitness values seem not to be considerably affected by the replacement mechanism. It is also interesting to note that fitness 9 is continuously increased without finding a 100%. Finally, fitness 0 and 2 describe a very similar trajectory during the first 50 steps, thus demonstrating the advantage of evolving coherent fault lists in the same experiment.

| Propagation & Excitation Reports | BDD based approach (# faults) | Gate Level Simulation (# faults) |
|---|---|---|
| Path Set (Complete) | 92.430 | 92.430 |
| Structurally Justified Paths | 10.394 | 234 |
| Excited Path Delay Faults | 2.731 | 46 |
| Propagated Faults (before error prop.) | 1.536 | 27 |
| Propagated Faults (final) | 2.489 | 86 |

Table 5.1: Propagation & Excitation based on (a)BDD approach (b)Gate Simulation

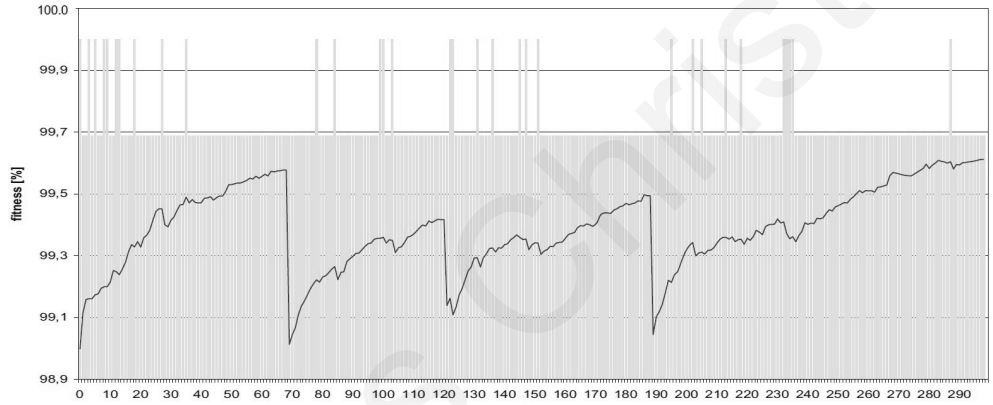



Figure 5.4: Fitness behavior on a coherent path list, average and maximum values.

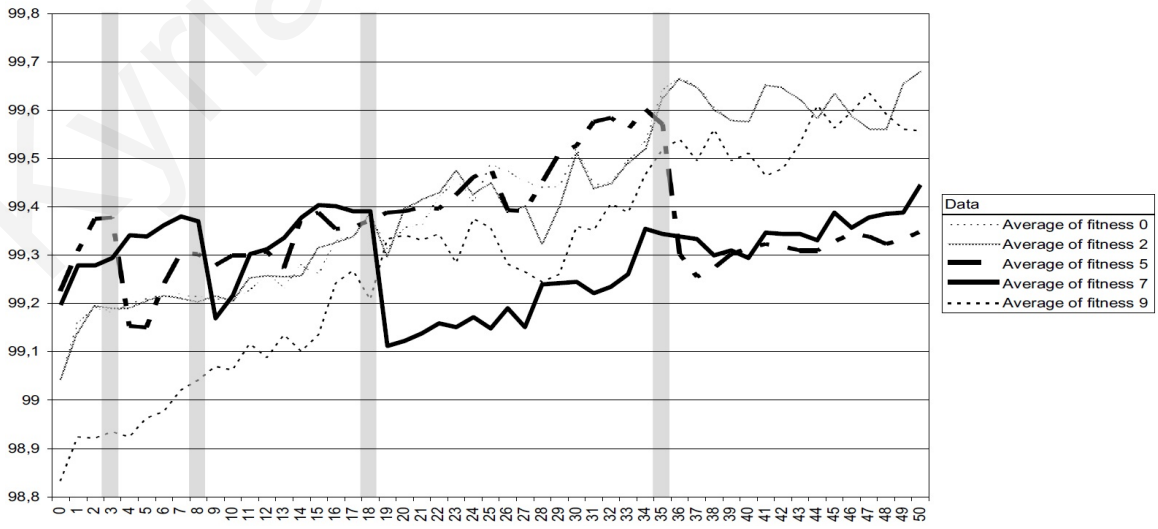



Figure 5.5: Trajectories of 5 fitness values during the first 50 steps.

## 5.5 Conclusions

This chapter presents an innovative approach for automating the generation of Path-Delay test programs for microprocessors cores. This has been achieved by employing a novel approach that uses BDDs analysis. A fitness value for directing the test generation flow is computed by finding a modified hamming distance algorithm on a Binary Decision Diagram that has been generated from the gate level description of the microprocessor core used. The produced fitness value seeks an optimal solution. Experimental results show that this methodology allows limiting the test generation time, by concentrating on suitably classified structurally coherent fault lists and avoiding computation intensive gate-level simulations. The obtained coverage results are comparable to manual/deterministic approaches in literature and shows the advantage of the methodology used over the existing ones.

# Chapter 6

# A Non-Enumerative Technique for Measuring Path Correlation in Digital Circuits

## 6.1 Introduction

A number of digital design automation procedures involve the examination and the characterization of the physical paths of a circuit's netlist. Applications including, automatic routing, design verification, fault diagnosis and test generation use information regarding the physical paths in order to accommodate various issues and provide efficient algorithms and techniques. The number of physical I/O paths in the circuit, as well as the number of path segments between two circuit locations, play an important role in almost all of the above processes. Also, it is often important to have an indication on the similarities (e.g. the number of common lines) between two or more paths. As an example of the above, the work in [6] considers a circuit partitioning problem where minimal cutsize and minimal delay per partition are desired. Special consideration should be given so that paths with large line overlap are not part of the same cut since this gives partitions with unbalanced delays, although the cutsize can be minimal.

Apart from numerous design procedures that can use to their benefit information on the similarities of physical paths, techniques for design verification, manufacturing testing and diagnosis can also employ this information to significantly increase their efficiency. Specifically, simulation-based applications for design error and fault diagnosis, such as those in [4, 41], can dramatically increase their diagnostic resolution if the fault/error propagation

path(s) of one fault is not a sub-path of those of other faults/errors. In this manner, the fault is distinguishable. In such cases, diagnostic test generation approaches should consider path overlaps during the test generation process.

Path overlap is also important in testing. Due to the increased complexity and radical technology scaling in modern digital circuits, it is now apparent that older fault models such as the stuck-at and transition delay fault models, no longer suffice in detecting emerging chip defects. Test generation procedures try to enhance the quality of the generated test set in terms of various measures beyond stuck-at/transition fault coverage and test set size, in an attempt to increase the defect coverage. Some examples of such measures are testing for low power dissipation, multiple fault detections ($n$-detect), path delay fault (PDF) coverage, small delay defect (SDD) coverage, etc. A common concept that must be handled by the test generation procedures for many of the above models and measures is that of path activation and path propagation. The quality of the generated test sets in terms of defect coverage can be enhanced if the various tests activate (propagate) faults via activation (propagation) paths with small overlap [50, 83].

An important usage of the information regarding the similarity among a set of physical paths of a circuit is in test generation for delay faults. Delay faults caused by emerging technology defects and process variations can dramatically change the timing behavioral of a circuit and can lead to unacceptable yield loss, especially for high performance products [49]. The most difficult to identify delay faults are those that are due to small delays and are distributed across various circuit lines, widely known as small delay defects [50,63,111,118]. Since the accumulation of small delay defects is highly related to the circuit's physical paths, the Path Delay Fault model (PDF) proposed in [106] is considered among the most accurate models for delay related defects. According to the PDF model, a fault is designated as the late transition across a physical path, either rising of falling. Since the number of physical paths in a circuit can be, in the worst case, exponential to the circuit size, deploying the PDF model in test generation is impractical, especially for dense circuits. A number of previously proposed test generation methodologies try to address the complexity issue of the PDF model by targeting a subset of the fault universe which only contains the *critical faults* [64, 110, 112, 119]. Different criteria and delay models define the "criticality" attribute of the faults according to the specific problem examined, with main objective to reduce the fault list and, thus, make the test generation procedure simpler and the obtained test set smaller. Most of the previous proposed techniques relate criticality with the length of the path or the slack of the path, i.e., the difference between the expected delay of the path and the maximum allowable

delay in the circuit. The slack is either estimated using statistical methods [118, 119] or calculated using process variation data obtained from the manufacturing process [110]. The work in [118] takes an $n$-detect test set for stuck-at faults as input and reduces its size by selecting those tests that sensitize the most critical paths according to a statistical approach. In an attempt to further improve the selection procedure the techniques of [64, 112] model additional technology parameters like coupling noise and power supply noise effects. The latter two approaches have proven the need for adding more criticality characteristics to physical paths than just their length.

The work in this chapter proposes a non-enumerative methodology to identify path (segment) overlap in a given set of physical paths, in order to quantify the correlation between the paths of the set considered. This set can be a subset of the circuit's I/O paths (i.e., set of critical paths), or path segments representing, for example, error/fault activation or propagation paths. A major contribution of the proposed technique is that it avoids explicit path of path segment enumeration by using a canonical representation of the paths based on Zero-Suppressed Binary Decision Diagrams (ZBDDs) [78] which have been shown to cope well with huge numbers of paths [84]. The method applies a constant number of standard ZBDD operators (of polynomial complexity) to find the path overlaps and represent them implicitly in a ZBDD providing a metric for the correlation of the paths in the given set. The different statistics obtained by this procedure are then summarized into a single number that can be used to characterize the correlation between the paths of each set. As mentioned in the previous paragraph, different applications can use this measure (or the provided statistics) according to their nature and indented goal. For example, path selection approaches for PDF or SDD test generation can use this measure to avoid selecting paths similar to paths that have already been selected, since this will, probably, not identify additional delays in the circuit. We elaborate more on the motivation for this particular application in Section 6.2.

Section 6.3 gives the detailed description of the methodology and all the appropriate definitions of the ZBDD operations used. Specifically, we define a measure for the average path overlap and explain its usage and meaning. We then provide some basic preliminaries on ZBDDs and explain how we use them in the proposed technique. Finally, we present and describe the basic steps of the proposed approach as well as a newly proposed operator that is used for calculating the overlap sizes in ZBDDs.

Section 6.4 shows the experimental results and presents the relevant discussion. We have used two different approaches in order to demonstrate the applicability of the proposed method and measure. The first one considers different groups of critical I/O paths

and quantifies the overlap between the paths for each group. The second approach identifies the overlaps between the propagation paths obtained from different test sets and investigates their relation with defect coverage. Section 6.5 summarizes this chapter.

## 6.2 Motivation

The discussion in Section 6.1 mentions a number of different design automation applications that can benefit from using path overlap information. Here, we briefly present one such possible application which we use in our experiments to evaluate the proposed method.

Consider test generation methods for small delay defects (SDDs). The majority of these methodologies use the PDF model to model such defects and try to reduce the number of faults by selecting the most important PDFs known as critical PDFs (see [64, 110, 112, 118, 119] among others). Criticality can be defined by various issues (often combined), such as path length and process variation parameters. These approaches do not explicitly consider path similarities and may result into selecting highly correlated paths. The latter is not necessarily undesired, but when distributed delays due to SDDs are targeted, selecting two (or more) correlated paths may not be the best choice. We demonstrate this case with an example.

Consider the circuit in Fig. 6.1 and assume, without any loss of generality, the fixed delay model with each NAND gate having nominal delay of 3 time units while the buffer and the inverter have 2 and 1 units, respectively. Let us assume that the maximum allowable I/O delay is set to 12 units in order to allow a small slag from the maximum path delay (11 units). Additionally, assume that the PDF test budget for this circuit is only 2, i.e., we can

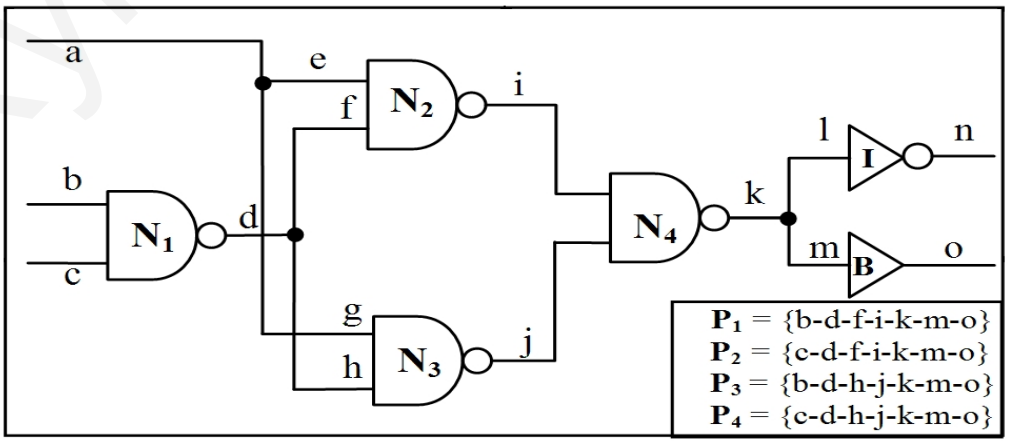



Figure 6.1: A path selection method example

afford only to consider two physical paths in the test generation process. Hence, the path selection method must select only 2 paths out of the 12 paths of the circuit to use as input to the test generation process. According to the gate delays, we identify that the critical paths in the circuit are four, $P_1$={b-d-f-i-k-m-o}, $P_2$={c-d-f-i-k-m-o}, $P_3$={b-d-h-j-k-m-o} and $P_4$ ={c-d-h-j-k-m-o}. There exist six different groups of paths with cardinality 2; considering any one of these groups can be viewed as a valid input candidate to the PDF test generation. If, however, we examine the overlap of each group the conclusion may result in a different decision. The overlap information for the path pairs is given at the first 6 rows of Table 6.1. For instance, the paths of pair $G_1$ have an overlap of 85.71% as there are 6 common lines out of the average path length which is 7 in this case (more details on this calculation are given in the next section). If a group with large overlap (e.g. $G_1$) is selected to be considered in the test generation procedure, its contribution in identifying small delays will be limited because the cumulative effect of the distributed delays will fall in the paths' common segment i.e., in {d-f-i-k-m-o}. To make it more specific, consider that due to process variations gates $N_1$, $N_3$ and $B$ exhibit excessive delay of 0.5 units while all the remaining gates exhibit their nominal delay. This situation can give unacceptable total delay across an I/O path of the circuit, as in the case of $P_3$ (delay is 3.5+3.5+3+2.5=12.5 units). Recall that the test generation process has a 2 path budget so a decision should be taken on which of the 6 pairs should be considered. By testing the two paths in $G_1$ the excessive cumulative delay is not detected as both $P_1$ and $P_2$ will give total delay of 12 (i.e., 3.5+3+3+2.5) which is acceptable. On the other hand, if a group with smaller correlation (overlap) between its paths (any other that $G_1$ and $G_6$) is selected, the delay will be detected.

Table 6.1: Path overlap for the example of Fig. 6.1

| Group | Paths | Overlap |
|---|---|---|
| $G_1$ | $P_1, P_2$ | 85.71% |
| $G_2$ | $P_1, P_3$ | 71.43% |
| $G_3$ | $P_1, P_4$ | 57.14% |
| $G_4$ | $P_2, P_3$ | 57.14% |
| $G_5$ | $P_2, P_4$ | 71.43% |
| $G_6$ | $P_3, P_4$ | 85.71% |
| $G_A$ | $P_1, P_2, P_3$ | 71.43% |

Generally, when the group with the smaller correlation is selected, the possibility to detect an excessive cumulative delay across anyone of the considered paths is increased. For instance, consider that for the circuit in Fig. 6.1 we are given 3 groups of paths, $G_1$, $G_2$, and $G_3$, and we are asked to decide which of this group is better to be given as input to a test generation procedure. In order to get excessive delay in the circuit, more than 1 time units of delay should be distributed among the gates of a critical path. By selecting the group with the smaller overlap (i.e., $G_3$) the obtained tests detect any excessive delay across any of the paths considered here as critical. This happens because paths $P_1$ and $P_4$ cover all the possible combinations of sub-paths between the 4 paths that are considered as critical.

Path correlation is calculated across consecutive and non-consecutive path segments like the case of group $G_5$, i.e., paths $P_2$ and $P_4$. It is important to ensure that this calculation does not affect the efficiency of the path selection approach, so it must avoid explicit path or path segment enumeration, since the number of paths in a circuit can be exponential to the circuit size. This is a very important attribute, since the modern design optimization tools tend to give circuits with paths of similar length, increasing the number of candidate critical paths. We elaborate further on the path correlation calculation on Section 6.3.

## 6.3 Proposed Methodology

This section discusses the proposed methodology. Section 6.3.1 gives the necessary definitions for pairwise path overlap in a set of paths. All the steps described here use a canonical data structure (ZBDD) for implicit path representation and manipulation. We provide a brief overview of ZBDD based path encoding in Section 6.3.2. Sections 6.3.3 and 6.3.4 present the proposed ZBDD-based operations to calculate path overlap and an illustrative example, respectively.

### 6.3.1 Pairwise Path Overlap Calculation

**Definition 1.** *Given two paths $P_1$ and $P_2$ the overlap operator $C(P_1, P_2)$ gives the set of lines that are common in $P_1$ and $P_2$.*

The calculation of the path overlap between two paths of the same size is done by dividing the number of common lines over the paths' size i.e., $|C(P_1, P_2)| / |P_1|$. For paths of different size, an averaging of the two paths' length is required. Given two paths $P_1$ and $P_2$ the overlap percentage is given as the size of the overlap between the two paths over the average size of

the two paths, i.e., $|C(P_1,P_2)|/(|P_1|+|P_1|)/2$.

**Definition 2.** *Given a set of paths $G=\{P_1,P_2,..P_n\}$, $\mathscr{D}$ is the pairwise path overlap operator computing the path overlap between all unique pairs of paths in G, given by:*

$$\mathscr{D}(G)=\{C(P_1,P_2),C(P_1,P_3),\ldots C(P_1,P_n),C(P_2,P_3),\ldots C(P_{n-1},P_n):\left|C(P_i,P_j)\right|\neq 0\} \quad (6.1)$$

The above operator checks all the pairs of paths ($P_i$,$P_j$), where $P_i$, $P_j \in G$ and $i \neq j$ and gives the sets of lines that form the overlap between the paths of each pair and have a non-zero size. Eventually, the obtained set $\mathscr{D}(G)$ contains all the path segments that are common in any of these pairs. Using the resulting set $\mathscr{D}(G)$ we give the following definition for the average overlap of a set of paths:

**Definition 3.** *Given a set of paths $G=\{P_1,P_2,...P_n\}$, the average pair-wise overlap measure $O(G)$ is given by:*

$$O(G)=\frac{\sum\limits_{(P_i,P_j)\in\mathscr{D}(G)}\left|C(P_i,P_j)\right|/|\mathscr{D}(G)|}{\sum\limits_{P_i\in G}|P_i|/|G|} \quad (6.2)$$

This measure gives the average pairwise overlap between the paths of a set of paths $G$. For example, the calculation of the average pairwise overlap for a path set $G_A$ which contains $P_1$, $P_2$ and $P_3$ of the circuit of Fig. 6.1 is 71.43% calculated as the average between the overlaps in $G_1$, $G_2$ and $G_4$, the pairwise combinations of the paths of $G_A$.

Note that the measure was selected to consider pairs of paths and not 3-tuples, 4-tuples and so on, in order to be generic and applicable in a broader range of applications. By selecting pairs we actually include all the combinations of paths (pairs, 3-tuples, 4-tuples, etc) and, thus, incorporate the total correlation between the considered paths in the measure. On the other hand, the averaging in Definition 3 ensures that no double counting of common sub-paths occurs and each common sub-path contributes equally to the measure. Fig. 6.2 illustrates three different examples of path overlap between three different paths, using Venn Diagrams, in order to demonstrate the application of the operators presented in the definitions above. Let each set correspond to a circuit path. The elements of a set correspond to the circuit lines constituting the path. Elements in the intersection of two sets are the lines that are common in the corresponding paths, i.e., form the overlap between

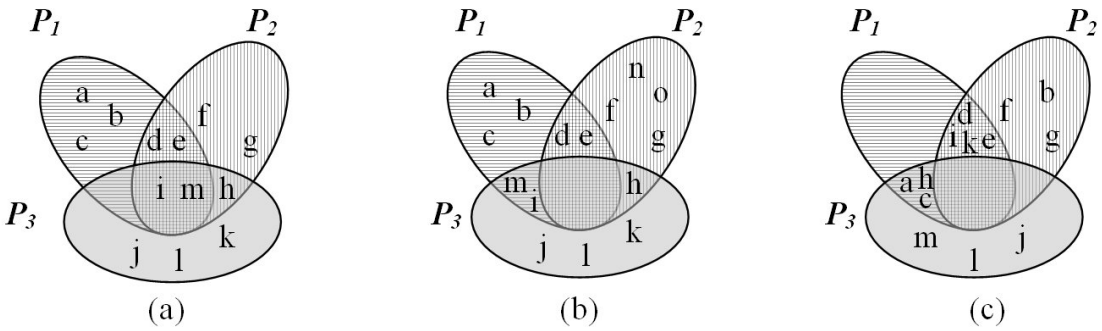



Figure 6.2: Examples for the calculation of the average path overlap measure of Definition 3.

the two paths. Consider first Fig. 6.2(a) with $P_1$={a-b-c-d-i-m}, $P_2$={d-e-f-g-h-i-m} and $P_3$={h-i-j-k-l-m}. The overlap operator of Definition 1 gives $C(P_1,P_2) = \{$d, e, i, m$\}$ and $|C(P_1,P_2)| = 4$. Similarly, $|C(P_1,P_3)| = 2$ and $|C(P_2,P_3)| = 3$. Applying Definition 2 on this set of paths gives $\mathscr{D}(G) = \{C(P_1,P_2), C(P_1,P_3), C(P_2,P_3)\} = \{\{$d,e,i,m$\},\{$i,m$\},\{$i,m,h$\}\}$, i.e., all the non-empty path overlaps. Using the average pairwise overlap measure of Definition 3 we calculate the average overlap between the considered paths to be:

$$
\begin{aligned}
O(G) &= \frac{\sum\limits_{\substack{i,j=1,\\ C(P_i,P_j)\in\mathscr{D}(G)}}^{3} \left|C(P_i,P_j)\right| / |\mathscr{D}(G)|}{\sum\limits_{\substack{i=1,\\ P_i\in G}}^{3} |P_i| / |G|} \\
&= \frac{(|C(P_1,P_2)| + |C(P_1,P_3)| + |C(P_2,P_3)|)/3}{(|P_1| + |P_2| + |P_3|)/3} \\
&= \frac{(4+2+3)/3}{(7+7+6)/3} \\
&= 45\%
\end{aligned}
\tag{6.3}
$$

Observe that the two lines that form the overlap between all the three paths (i.e., *i* and *m*) contribute more in the above measure than the lines that are common only in two sets (paths). This becomes more clear if we consider the paths in Fig. 6.2(b). Here, $P_1 = \{$a-b-c-d-i-m$\}$, $P_2 = \{$d-e-f-g-h-n-o$\}$ and $P_3 = \{$h-i-j-k-l-m$\}$. For this case we get $O(G) = \frac{(2+2+1)/3}{(7+7+6)/3} = 25\%$ indicating that the average overlap is smaller, although the number of lines in the overlap of any two paths as well as the size of all paths considered are the same as in case (a). Obviously, the fact that two lines are in the overlap of all three paths in the diagram of Fig. 6.2(a) increases the average overlap compared with the diagram of Fig. 6.2(b). This is reflected to the value of the measure introduced in Definition 3 even though the paths are

considered in pairs and not in 3-tuples (25% for (b) against 45% for(a)). Furthermore, the averaging in both the numerator and the denominator ensures that the values of the measure for different path sets are comparable, when the sets are obtained for the same circuit. For instance, the paths in Fig. 6.2(c) have clearly higher average overlap than the previous two cases. Here, $P_1 = \{$a-c-d-e-h-i-k$\}$, $P_2 = \{$b-d-e-f-g-i-k$\}$ and $P_3 = \{$a-c-h-j-l-m$\}$. However, one pair gives zero overlap (i.e., $|C(P_2,P_3)| = 0$) and it should not be considered by the measure. If we had only considered the sizes of the overlaps and the paths, the measure would indicate that the paths in Fig. 6.2(a) have more overlap than those in Fig. 6.2(c) (45% for (a) against 35% for (c)), which is not confirmed by the diagrams. Applying the measure to the paths represented by the diagrams in Fig. 6.2(c) gives a higher average overlap value i.e., $O(G) = \frac{(4+3)/2}{(7+7+6)/3} = 52.5\%$ confirming the larger overlap shown in the diagrams.

### 6.3.2 Path Representation and Manipulation

The proposed methodology uses Zero-Suppressed Binary Decision Diagrams (ZBDDs) to represent a large number of paths and obtains information on their overlap using standard ZBDD operations. ZBDDs [78] have been successfully used in representing a huge number of paths compactly in a non-enumerative fashion [83]. The representation of paths in a circuit requires only a polynomial number of standard ZBDD operations and can be generated and represented in a ZBDD format (as in [83]), by a single topological traversal. Extensive experimentation has shown that, even circuits with huge number of paths can be effectively represented using ZBDDs following the approach of [83], which avoids path enumeration.

Fig. 6.3(a) shows the representation of set $G_A$ of Table 6.1 that consists of paths $P_1$, $P_2$ and $P_3$ of Fig. 6.1, using ZBDDs. The vertices in the diagram correspond to circuit lines, and always follow a predefined order (here b $<$ c $<$ d $<$ f $<$ i $<$ h $<$ j $<$ k $<$ m $<$ o). A path can be obtained by following the paths of the diagram that end in the terminal vertex 1 and considering only the solid (true) edges. The true edges correspond to the existence of a circuit line in a path, whereas the false (dashed) edges correspond to the absence of a line in a path. The shaded part of the diagram represents path $P_3$=$\{$b-d-h-j-k-m-o$\}$. Observe that, while vertex *f* falls in the shaded area, it is not considered in the path since it is connected to *h* via a dashed edge. Likewise, the other two paths are represented in the diagram. In the same way ZBDDs can represent path segments that may not necessarily consist of consecutive circuit lines.

### 6.3.3 Overlap Identification Algorithm

Given a circuit $C$ and a set of paths $G$, the proposed algorithm identifies a set of overlapping path segments D($G$) in a non enumerative way using a constant number of standard ZBDD operators. Set $G$ does not necessarily contain I/O full paths, but it can contain path segments as well. This allows the algorithm to handle path segments as well, which is desirable for a number of applications like fault diagnosis, where we want to distinguish between different sub-paths. The algorithm first implements the pairwise path overlap operator and then performs a final step to obtain the required overlap information. The complexity of algorithm is in the worst case polynomial to the ZBDD size.

**Data**: $C$: Circuit Under Examination;
$G$: Set of Paths (or path segments) in $C$;
**Result**: Overlap Statistics $H$;
1 $Z_0^T \leftarrow$ ZBDD_represent($C$, $G$) ;
2 $Z_1^T \leftarrow Z_0^T \otimes Z_0^T$ ;
3 $Z_2^T \leftarrow$ **maximal** ($Z_0^T$) ;
4 $Z_3^T \leftarrow Z_2^T \cap Z_1^T$ ;
5 $Z^T \leftarrow Z_1^T$ / $Z_3^T$ % $Z^T$ *corresponds to D(G)* ;
6 $H \leftarrow$ retrieve_overlap_histogram($Z^T$) ;
7 **return** overlap_statistics($H$);

**Algorithm 6.1:** Path_Overlap_Identification

The procedure of Algorithm 6.1 outlines the basic steps of the algorithm. In the first step (Line 1) the procedure builds the ZBDD that contains the paths in the given set G. The step in Line 2 applies a standard ZBDD operation called pairwise intersection ($\otimes$) which takes all pairs of the paths in the given set and identifies the overlap between the paths of each pair. The pairwise overlap operation examines all the pairs of paths in $Z_0^T$ and gives a set containing the sets of lines (sub-paths) in each pair that form the overlap of the two paths. Hence, $Z_1^T$ contains the overlaps that were identified between any two paths of set $G$. At this point we note that as a side effect of the pairwise intersection operator all the paths (or sub-paths) initially in $G$ will be also contained in $Z_1^T$, since the operator does not exclude pairs of the same path. In this case a path initially in $G$ is identified as overlap since the overlap between two identical paths is the path itself. Using the notation of Definition 1, $Z_1^T$ contains all the $C(P_i,P_i)$ for all $i = 1,2\ldots|G|$ which are equal to $P_i$ for all $i$. This situation is not desired when only the overlap between distinct paths is required. The next three

steps of the algorithm concentrate on cancelling this side effect considering all the possible complications. Lines 3 and 4 give the diagram that contains all the unwanted paths that should be removed from the pairwise intersection result. Intuitively, if we remove from $Z_1^T$ all the paths in $G$ this problem will be solved. However, this is not true when $G$ contains not only I/O full paths (from a primary input to a primary output) but sub-paths as well. In this case it is not clear whether an overlap in $Z_1^T$ is a true overlap or a sub-path initially contained in the given set $G$. The maximal operator (Line 3) removes from $G$ all the paths that are proper sub-paths of other paths, and, hence, removes this disambiguation. This step is not necessary if $G$ contains only full paths. The maximal operation on a set of full paths will return the set itself and so this step does not affect the case where $G$ consists only of full paths. Line 4 identifies the paths that must be removed from $Z_1^T$. The set difference operation of Line 5 removes all the unwanted paths from the obtained overlap set, keeping only the sub-paths that corresponds to the overlap of any two distinct paths in set $G$. At this point we have all the overlapping sub-paths in $Z^T$ which is the ZBDD representation of the set D($G$) used in Definition 2.

In Line 6, the overlap information is retrieved from the ZBDD in a histogram-like structure, using a recursive operator, while Line 7 returns different statistics for the obtained overlap set the most important of which is the average pairwise overlap (Definition 3) and the standard statistical measure of skewness of the histogram $H$. We elaborate further on the usage of the skewness measure in Section 6.4.

The operator of Step 6 is a newly proposed ZBDD operator, which is executed in linear time to the size of the ZBDD and, thus, does not increase the method's complexity. It avoids explicit path or path segment enumeration, and hence, does not contribute further to the algorithm's complexity. The pseudocode of Algorithm 6.2 describes the operator of Line 6 in Algorithm 6.1. The operator takes a ZBDD ($Z^T$) as an argument and calculates an array per ZBDD vertex that is used to represent the overlap histogram, in a recursive manner. Each position of the array holds the number of paths (or sub-paths) that have size (length) equal to the position. For example, array [0,3,5,4,1] represents a set of paths with 3 overlaps of size 1, 5 with size 2, 4 with size 3 and 1 overlap with size 4. The size of each array is statically defined to be equal to the largest path is the circuit (circuit's depth). The first position of the array indicates a zero size overlap and is only used for terminal vertex 1 to provide termination of the recursion (Line 1). Lines 2 to 4 check if the root vertex of $Z^T$ is a terminal vertex (terminal zero or terminal one) to indicate the termination of the recursive process. If the root of $Z^T$ is the terminal 0 vertex, a zero histogram is returned to the calling

process (Line 3); for terminal 1 a histogram with a single one in the $0^{th}$ position is returned (Line 4). In all other cases (i.e., where the root of $Z^T$ is not a terminal vertex) the procedure calls itself to calculate the histograms for the two children of $Z^T$ root (Lines 7 and 9). Next, the two arrays are merged together with the one coming from the true edge shifted by one position on the right (Line 10) indicating the contribution of this child to a sub-path overlap. The procedure returns the histogram array to the calling process in line 12, until the root of $Z^T$ is reached and the histogram is returned to the procedure of Algorithm 6.1. Each vertex is only visited once. Even when a vertex is a child of two or more parents, the calculated array is cached after the first visit and used for all parents of that vertex. Thus, the complexity of the operator is linear to the size of the diagram.

In the next subsection, we present an illustrative example of the overall process.

```
   Data: Z^T: ZBDD representing a set of Sub-Paths
   Result: Histogram H
 1 H_zero ← [0, 0, 0 ..., 0], H_one ← [1, 0, 0 ..., 0],;
 2 K = root(Z^T) % root() gives the root vertex of ZBDD Z^T ;
 3 if (k==terminal_vertex(0)) then return(H_zero);
 4 else if (k==terminal_vertex(1)) then return(H_one);
 5 else
 6     % true_edge() gives the ZBDD rooted at solid line child of K ;
 7     H_T ← Retrieve_Overlap_Histogram(true_edge(K));
 8     % false_edge() gives the ZBDD rooted at dashed line child of K ;
 9     H_E ← Retrieve_Overlap_Histogram(false_edge(K));
10     H_K ← right_shift(H_T) + H_E;
11 end
12 return H_K;
```

**Algorithm 6.2:** Retrieve_Overlap_Histogram (Linear Complexity Operator)

### 6.3.4 A Path Overlap Calculation Example

We next give an example to illustrate the execution of the algorithm. Consider again set $G_A$ of the circuit of Fig. 6.1 (i.e., $P_1$, $P_2$ and $P_3$). Fig. 6.3(a) shows the ZBDD representation that corresponds to the paths in $G_A$ ($Z_0^T$ in Algorithm 6.1). Fig. 6.3(b) shows the ZBDD after applying the pairwise intersection operator ($Z_1^T$ in Algorithm 6.1). Observe that this diagram contains a total of 6 paths (or sub-paths): (i) 3 sub-paths corresponding to all possible overlaps between the 3 paths and (ii) the 3 paths of the initial diagram. For instance,

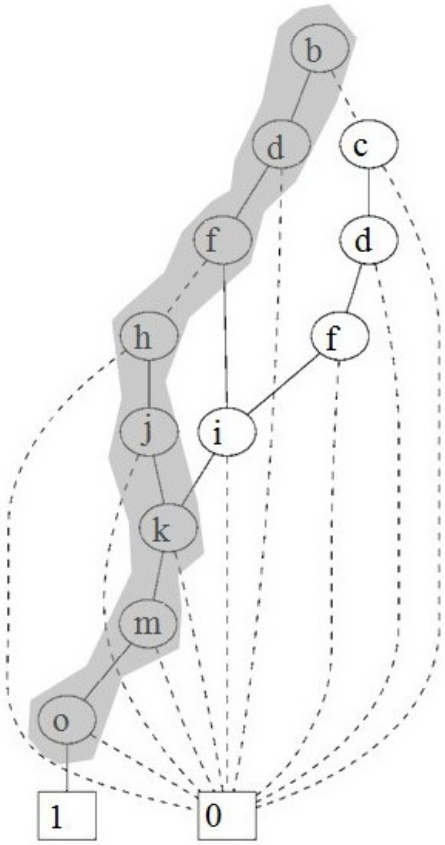
the shaded part of the diagram corresponds to the sub-path {d-k-m-o} which is the overlap between paths $P_2$, $P_3$. Note that, this information does not correspond to path segments only, but it also contains overlap information than spans in more than one path segment like the case of the shaded path. The intersection and set difference operations (lines 4 and 5 in Algorithm 6.1) remove all the unwanted paths and keep only the overlaps as shown in Fig. 6.3(c) ($Z^T$ in Algorithm 6.1).

Diagram representing (a) the paths in $G_A$, (b) after applying the pairwise intersection operator ($Z_1^T$ in Algorithm 6.1), (c) after applying intersection and set difference which results in keeping only the overlaps ($Z^T$ in Algorithm 6.1), and (d) execution of the algorithm that retrieves the overlap histogram (operator of Algorithm 6.2).

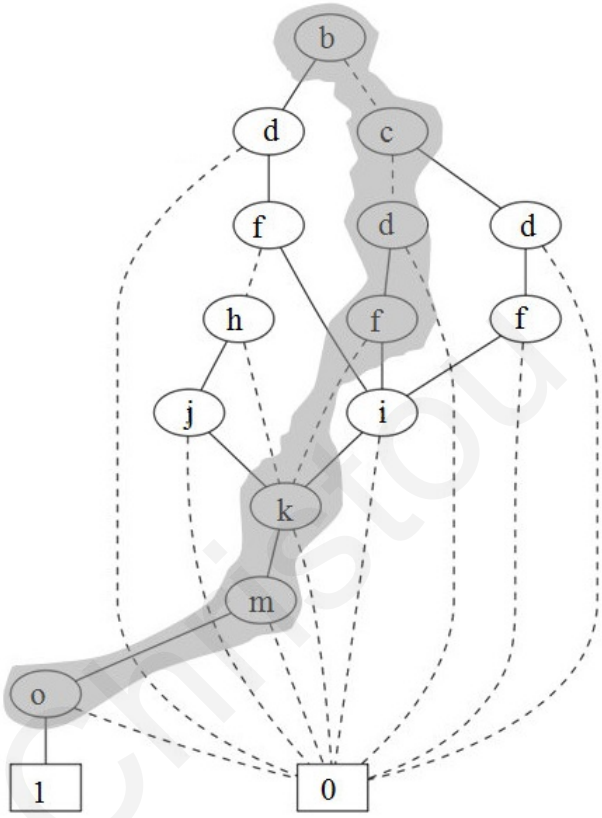
Finally, in the diagram of Fig. 6.3(d) we describe the execution of the recursive operator of Algorithm 6.2 which calculates the overlap histogram. The operator maintains a size array per vertex (shown in black) holding the numbers and sizes of the overlaps so far, from bottom to top of the ZBDD. For example, the array of vertex $f$ denotes that up to this vertex two sub-paths exist, one with size 3 (3rd position in the array) and one with size 5 (5th position in the array). The size of the array is statically determined by the largest true path in the ZBDD (6 in this case) plus one for the $0^{th}$ position needed for recursion termination. Starting from the root vertex of the diagram ($a$) the operator calculates the histogram array of each vertex by merging the histogram arrays of the two children of the vertex. For example, for vertex $f$ it merges the arrays of vertices $i$ and $k$ by shifting the array coming from the solid edge (vertex $i$) to the right and adding it to the (non-shifted) array coming from the dashed line (vertex $k$). This way all the solid edges are considered in the overlap calculation and all the dashed lines are omitted. The process stops the recursion when it reaches one of the two terminal vertices where predetermined arrays are returned ($H_{one}$ and $H_{zero}$ of Algorithm 6.2). The final array associated with the root of the diagram denotes that the obtained overlap set has three overlaps, one with size 4, one with size 5 and one with 6. This array corresponds to the histogram structure $H$ of the algorithm. The average overlap percentage from Definition 3 is then calculated as the average of the three overlaps over the average size of path length in the given set, i.e., O($G_A$)=[(4+5+6)/3] / [(7+7+7)/3]=71.43%.

## 6.4 Experimental Results

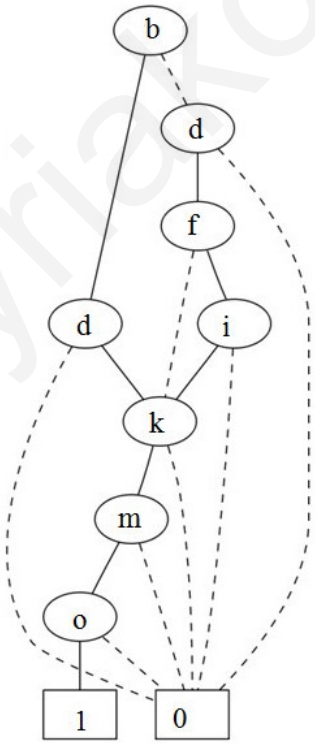
The proposed method was implemented in C language using the decision diagram package of [107] (for all ZBDD related operations), and run on a 1GHZ SunBlade 2500 workstation



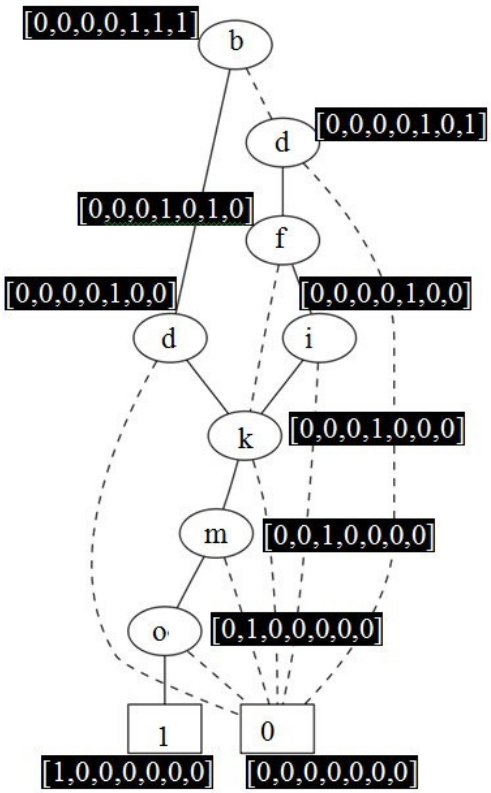

(a) paths in $G_A$




(b) pairwise intersection




(c) intersection and set difference




(d) algorithm execution

Figure 6.3: Using ZBDDs for path set representation and overlap identification.

with 4GB of memory. We report results for two different test-related procedures involving paths. In Subsection 6.4.1 we give as input to the proposed methodology path sets that consist of critical I/O paths with less than 10% slag. We use the library of the TSMC 0.18um process to calculate the delays and obtain the slag of the paths and we report statistics for a number of indicative circuits from the ISCAS'85, ISCAS'89 and ITC'99 benchmark suites. In Subsection 6.4.2 we compare the average propagation path overlap (path segments), as well as other statistics, of two different test sets generated considering stuck-at faults.

### 6.4.1 Set of critical I/O paths with different average path overlap

We first show resources requirements for the proposed method in Table 6.2. The second column reports the number of paths in the considered path set, while Columns 3 and 4 report the proposed methodology's requirements in CPU run time and memory, respectively. These numbers demonstrate that the method is very efficient and with memory requirements that are not prohibitive even for circuits with a large number of paths. In order to point out the efficiency of the proposed method, we compare it with a brute-force approach that does not avoid path enumeration (in the absence of any existing non-enumerative comparable method). The brute-force approach actually selects all path pairs enumeratively and identifies their common lines in order to give the overlap measure according to Definition 2. Column 5 reports the run time in seconds for the brute-force technique. For the circuits with a star (*) the brute-force technique was allowed to run for 48 hours before aborted. It is obvious that the proposed methodology is significantly faster than the brute-force technique providing the desired results in seconds, even for circuits with a large number of paths like those in the last rows of Table 6.2.

Next we give the results for the overlap between different sets of paths, as well as some other measures. For each circuit we use as input, four different path sets of equal size. Sets $G_1$, $G_2$, $G_3$ and $G_4$ consist of critical paths (with less than 10% slag) and have been selected from a pool of critical paths using four different approaches. $G_1$ and $G_2$ were obtained in a random fashion, $G_3$ was obtained by selecting paths with more overlaps in the longest paths and, finally, $G_4$ was obtained by selecting paths from different I/O cones. We first discuss in detail the obtained results for the medium size circuit s713 from the ISCAS'89 suite. For each one of the 4 path sets we show the overlap distribution in the histograms of Fig. 6.4. In the X axis of each histogram we show the overlap size in lines and in the Y axis the number of the sub-paths having this overlap size. The histograms provide a complete picture of the

Table 6.2: Comparison with brute-force approach

| Circuit | # Critical I/O Paths | Proposed Method | | Brute Force |
|---|---|---|---|---|
| | | CPU Time (s) | Mem (MBs) | CPU Time (s) |
| s641 | 104 | 0.02 | 5.65 | 1.26 |
| s5378 | 360 | 0.02 | 13.2 | 15.21 |
| b09_opt | 372 | 0.04 | 8.57 | 16.25 |
| b12_opt | 472 | 0.02 | 5.83 | 26.17 |
| s3271 | 650 | 0.02 | 12.95 | 49.66 |
| c880 | 682 | 0.02 | 5.35 | 54.67 |
| s1423 | 726 | 0.02 | 6.73 | 61.96 |
| b11_opt | 1,033 | 0.02 | 7.64 | 125.49 |
| s713 | 3,830 | 0.02 | 8.65 | 1726.22 |
| b07_opt | 6,659 | 0.02 | 6.65 | 1.45 h |
| b04_opt | 8,960 | 0.2 | 5.27 | 2.62 h |
| s38584.1 | 24,300 | 1.02 | 75.53 | 19.30 h |
| b14_1_opt | 33,528 | 0.6 | 26.54 | 36.75 h |
| c7552 | 80640 | 0.02 | 23.32 | * |
| c2670 | 116,100 | 0.82 | 15.01 | * |
| c5315 | 138,720 | 0.02 | 31.1 | * |
| c1908 | 161280 | 0.17 | 29.7 | * |
| b14_opt | 175,456 | 7.06 | 41.75 | * |
| b05_opt | 439,103 | 0.14 | 37.64 | * |
| b21_1_opt | 1,949,696 | 0.16 | 74.71 | * |
| b20_1_opt | 1,950,650 | 0.16 | 77.81 | * |
| c1355 | 1,959,600 | 5.78 | 84.65 | * |
| b15_1_opt | 2,209,040 | 0.78 | 95.41 | * |
| c3540 | 4,641,360 | 2.14 | 124.82 | * |

overlap distribution; however this information is not practical, especially if we want to use it to guide a path selection algorithm. The average pairwise overlap measure (O($G$) from Definition 3) gives an indication of this distribution using a single number. For example, O($G_2$) is $8.43 \times 10^{-3}$ while O($G_4$) is $5.14 \times 10^{-3}$ indicating that the overlap is larger in $G_2$ than in $G_4$. Indeed the distribution of the overlaps, as well as the numbers for each overlap size indicate that this observation in true. Similarly, the overlap in $G_3$ is larger than in $G_1$ even thought their distributions seem to be similar. In the case of $G_3$ however, the number of sub-paths per overlap is much larger than in $G_1$.

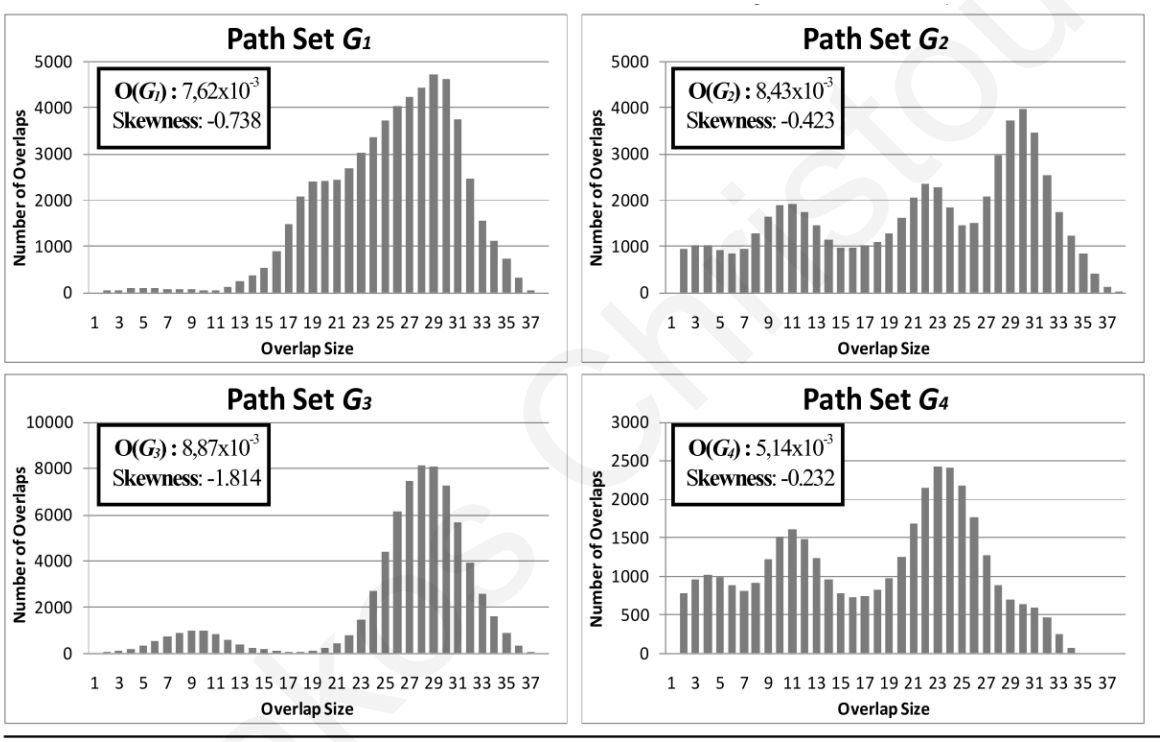



Figure 6.4: Path Overlap Distribution for four different sets of paths of circuit s713

While the measure from Definition 3 gives a very good indication about the amount of overlap in the circuit, it gives no information on where the overlaps lay in the graph. For this reason we use the standard statistical measure of skewness which denotes if the majority of the values are situated on the left of the average (positive skewness) or on the right of the average (negative skewness). It is clear that if the desired property of the path set is to have small correlation, the skewness measure should be positive or very close to 0 (zero). For instance, observe that while sets $G_2$ and $G_3$ have similar average overlap values, their distributions are very different, and hence, their skewness values are different. The larger negative value of $G_3$ indicates that it contains larger overlaps than those in $G_2$, which is verified from the histogram. The combination of the skewness with the average overlap

measure gives a very good picture of the actual histogram obtained in step 6 of the proposed algorithm (Algorithm 6.1).

Table 6.3 reports the obtained statistics for all the circuits considered, for each of the four path sets $G_1$, $G_2$, $G_3$ and $G_4$ Columns 1 and 2 report the circuit name and the number of paths considered in each set. Next, we report the statistics for each path set. In columns 3, 6, 9 and 12, we report the average pairwise path overlap and in columns 5, 8, 11 and 14 the values for the skewness measure. Columns 4, 7, 10 and 13 show the relative difference between the average overlap values, in order to provide a more indicative relationship between these values. The differences are calculated with respect to the smaller average overlap between the four sets. For instance, the average overlap of $G_3$ of circuit s38584.1 differs by 52.894% from the corresponding value of $G_2$ which has the smaller overall value i.e., (45.090-29.491)/29.491=52.894%. On the other hand, the difference between $G_2$ and $G_4$ is only 8.779%. Note that this difference is in some cases significant implying large differences in the average overlap of the considered set, even though the paths sets have exactly the same cardinality.

### 6.4.2 Evaluation of stuck-at test sets for propagation path overlap

In this subsection, we use as input to the proposed methodology sets that consist of fault propagation paths during test application. For this, we have used three different multiple-detect test sets for stuck-at faults, i.e., test sets that explicitly target each modeled stuck-at fault multiple (more than one) times. Multiple-detect test sets were selected for this case since they have been proven to increase the fault coverage of non-modeled defects including timing defects [71] that are tightly related to paths. We have used the multiple-detect test set of [83] for both ISCAS'85 and ISCAS'89 benchmark circuits, and the $n$-detect test sets of [82] and [87] for the ISCAS'85 and ISCAS'89, respectively. We use three different test sets for the comparison. The test sets obtained from the work in [87] are directly comparable with the method proposed in [83], yet there are not test sets available for ISCAS'85. For this purpose we have used the method of [82] to generate $n$-detect test sets that are comparable with those obtained from [83]. The experimentation consist of two steps. We first apply the test sets coloring the propagation paths for each pattern and collecting all the propagation paths in a single group of paths. Then we apply the proposed methodology and report the statistics from the obtained histograms. The overlap statistics are compared in relation to an estimation of non-modeled defect coverage obtained by the corresponding test.

Table 6.3: Statistics for the four different path sets obtained by the proposed method

| Circuit | # of Critical I/O Paths | $G_1$ | | | $G_2$ | | | $G_3$ | | | $G_4$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | O($G_1$) (x$10^{-4}$) | Diff (%) | Skns | O($G_2$) (x$10^{-4}$) | Diff (%) | Skns | O($G_3$) (x$10^{-4}$) | Diff (%) | Skns | O($G_4$) (x$10^{-4}$) | Diff (%) | Skns |
| s641 | 104 | 316.044 | 7.358 | 0.048 | 469.986 | 59.650 | 0.287 | 298.688 | 1.462 | -0.011 | 294.384 | 0.000 | 0.688 |
| s5378 | 360 | 265.609 | 9.570 | 0.014 | 267.453 | 10.331 | 0.000 | 277.949 | 14.661 | -0.001 | 242.410 | 0.000 | 0.127 |
| b09_opt | 372 | 76.692 | 2.392 | -0.281 | 74.901 | 0.000 | -0.365 | 95.817 | 27.925 | -0.454 | 95.290 | 27.222 | -0.445 |
| b12_opt | 472 | 209.224 | 28.262 | 0.570 | 166.461 | 2.047 | 0.689 | 221.836 | 35.994 | 0.531 | 163.122 | 0.000 | 0.700 |
| s3271 | 650 | 304.365 | 8.659 | 0.122 | 280,111 | 0,000 | -0,044 | 320,580 | 14,448 | -0,218 | 281,012 | 0,322 | 0,549 |
| c880 | 682 | 151.840 | 18.093 | -0.433 | 147.502 | 14.719 | -0.242 | 137.515 | 6.951 | -0.612 | 128.577 | 0.000 | -0.008 |
| s1423 | 726 | 212.248 | 13.080 | -0.273 | 187.697 | 0.000 | -0.179 | 275.993 | 47.042 | -0.251 | 210.165 | 11.970 | -0.356 |
| b11_opt | 1,033 | 524.203 | 13.740 | 0.268 | 460.877 | 0.000 | 0.310 | 726.997 | 57.742 | 0.295 | 564.773 | 22.543 | 0.518 |
| s713 | 3,830 | 76.187 | 48.315 | -0.738 | 84.278 | 64.067 | -0.424 | 88.718 | 72.709 | -181.843 | 51.368 | 0.000 | -0.232 |
| b07_opt | 6,659 | 99.841 | 11.540 | 0.444 | 92.621 | 3.474 | 0.465 | 104.296 | 16.517 | 0.466 | 89.511 | 0.000 | 0.660 |
| b04_opt | 8,960 | 110.845 | 16.029 | 0.460 | 112.985 | 18.270 | 0.614 | 141.346 | 47.957 | 0.580 | 95.532 | 0.000 | 0.553 |
| s38584.1 | 24,300 | 36.944 | 25.275 | 0.149 | 29.491 | 0.000 | 0.228 | 45.090 | 52.894 | 0.310 | 32.080 | 8.779 | 0.479 |
| b14_1_opt | 33,528 | 7.603 | 24.179 | 0.273 | 6.791 | 10.915 | 0.154 | 7.409 | 21.003 | -0.008 | 6.123 | 0.000 | 0.330 |
| c7552 | 80,640 | 28.852 | 13.137 | 0.107 | 30.023 | 17.730 | 0.012 | 28.252 | 10.788 | -0.170 | 25.501 | 0.000 | -0.021 |
| c2670 | 116,100 | 5.922 | 5.923 | 0.149 | 5.825 | 4.187 | 0.129 | 7.437 | 33.021 | 0.146 | 5.591 | 0.000 | 0.219 |
| c5315 | 138,720 | 25.835 | 0.000 | 0.035 | 27.883 | 7.927 | 0.020 | 33.055 | 27.950 | 0.074 | 27.574 | 6.731 | 0.166 |
| c1908 | 161,280 | 6.382 | 4.483 | 0.022 | 6.341 | 3.820 | -0.017 | 6.108 | 0.000 | -0.049 | 6.264 | 2.551 | 0.098 |
| b14_opt | 175,456 | 1.241 | 1.245 | 0.992 | 1.285 | 4.843 | 0.819 | 1.226 | 0.000 | 0.166 | 1.348 | 9.983 | 0.840 |
| b05_opt | 439,103 | 5.719 | 31.182 | 0.348 | 5.118 | 17.391 | 0.266 | 4.360 | 0.000 | 0.200 | 4.967 | 13.932 | 0.376 |
| b21_1_opt | 1,949,696 | 0.195 | 0.000 | 0.309 | 0.487 | 149.857 | 0.309 | 0.798 | 309.992 | 0.107 | 0.487 | 149.908 | 0.309 |
| b20_1_opt | 1,950,650 | 0.815 | 93.894 | -0.335 | 0.462 | 9.946 | -0.252 | 0.420 | 0.000 | -0.335 | 0.480 | 14.167 | -0.335 |
| c1355 | 1,959,600 | 0.342 | 2.858 | -0.135 | 0.332 | 0.000 | -0.089 | 0.531 | 59.584 | -0.161 | 0.342 | 2.812 | 0.160 |
| b15_1_opt | 2,209,040 | 0.477 | 31.444 | 0.602 | 0.379 | 4.419 | 0.515 | 0.507 | 39.657 | 0.422 | 0.363 | 0.000 | 0.702 |
| c3540 | 4,641,360 | 0.420 | 5.723 | 0.107 | 0.397 | 0.000 | 0.154 | 0.416 | 4.637 | 0.112 | 0.401 | 0.906 | 0.141 |

Table 6.4 reports the statistics obtained for the different test sets generated for stuck-at faults. After the circuit name in Column 1, we report results for the multiple-detect test set of [83]. Column 2 gives the size of the test set. Columns 3 and 4 report the average path overlap and the skewness measures, respectively. Column 5, reports the Bridging Fault coverage using a popular estimator called BCE+ proposed in [42]. The following four columns give the same results for the two $n$-detect test sets. For the ISCAS'85 the test sets are obtained using the method of [82] while for the ISCAS'89 the test sets are obtained from the work in [87]. Observe that in most cases, the test set with the largest value of the average path overlap measure ($O(G)$) gives larger defect coverage (bridging faults are used as surrogates). The test set sizes are very similar and so cannot attribute the difference in the BCE+. For example, for circuit s9234, the $n$-detect test set has a smaller value for the average path overlap than the multiple-detect and although its test set size is smaller, the defect coverage is higher. In cases where the value of $O(G)$ is similar for the compared test sets, like the case of s526, the skewness value resolves the issue. For s526, the $n$-detect test set has a larger value positive skewness indicating that the overlap sizes are smaller than the $n$-detect test set (closer to zero) and this could explain the larger defect coverage.

## 6.5 Conclusions

This chapter presents an efficient way of identifying the pairwise path correlation between the paths in a set. A new methodology based on ZBDDs is proposed that gives a comprehensive statistical characterization for a given path set. The method is based on standard ZBDD operations of polynomial, to the size of the diagram, complexity. Experimentation using the proposed measure demonstrates its effectiveness via two different approaches. The first one shows how the proposed technique identifies similarities among various I/O critical path sets and can distinguish their characteristics based on just two measure values per test. The second one uses the proposed methodology to compare the propagation paths between multiple-detect and $n$-detect test sets in relation to their corresponding defect coverage.

Table 6.4: Comparing path overlaps between multiple-detect and $n$-detect test sets

| Circuit | **Multiple-detect Test Set [83]** | | | | **$n$-detect Test Set [82]$^{\pm}$ and [87]$^{\pm\pm}$** | | | |
|---|---|---|---|---|---|---|---|---|
| | # of Tests | $O(G_1)$ $\times 10^{-4}$ | Skns | BCE+ | # of Tests | $O(G_1)$ $\times 10^{-4}$ | Skns | BCE+ |
| c880 | 199 | 4.226 | -0.858 | 0.97330 | 200 $^{\pm}$ | 4.118 | -0.811 | 0.97044 |
| c1355 | 841 | 0.725 | -0.583 | 0.92898 | 840 $^{\pm}$ | 0.723 | -0.607 | 0.92412 |
| c1908 | 1052 | 1.445 | -0.397 | 0.96506 | 1070 $^{\pm}$ | 1.521 | -0.424 | 0.95872 |
| c2670 | 538 | 1.085 | -1.202 | 0.95853 | 550 $^{\pm}$ | 1.089 | -1.252 | 0.95863 |
| c3540 | 1163 | 1.935 | -0.721 | 0.96127 | 1000 $^{\pm}$ | 1.366 | -0.994 | 0.96100 |
| c7552 | 1165 | 3.234 | 2.124 | 0.98045 | 780 $^{\pm}$ | 4.125 | 1.956 | 0.97986 |
| s208 | 279 | 42.334 | 0.423 | 0.98248 | 271 $^{\pm\pm}$ | 46.221 | 0.370 | 0.95213 |
| s298 | 232 | 28.651 | 0.541 | 0.95529 | 234 $^{\pm\pm}$ | 26.772 | 0.571 | 0.99595 |
| s344 | 136 | 28.782 | -0.035 | 0.99627 | 138 $^{\pm\pm}$ | 28.815 | -0.035 | 0.98643 |
| s382 | 252 | 17.096 | 0.140 | 0.98391 | 253 $^{\pm\pm}$ | 17.766 | 0.196 | 0.99722 |
| s386 | 209 | 13.400 | -0.279 | 0.99707 | 201 $^{\pm\pm}$ | 14.348 | -0.231 | 0.94471 |
| s420 | 410 | 14.628 | -0.124 | 0.95167 | 433 $^{\pm\pm}$ | 19.784 | 0.035 | 0.85641 |
| s510 | 541 | 14.841 | 0.635 | 0.89850 | 543 $^{\pm\pm}$ | 12.484 | 0.635 | 0.98383 |
| s526 | 469 | 11.146 | 0.431 | 0.98386 | 492 $^{\pm\pm}$ | 11.351 | 1.799 | 0.99382 |
| s641 | 226 | 9.596 | -0.475 | 0.99322 | 227 $^{\pm\pm}$ | 13.604 | -0.417 | 0.99125 |
| s820 | 950 | 5.946 | 0.207 | 0.98227 | 949 $^{\pm\pm}$ | 6.046 | 0.200 | 0.99959 |
| s953 | 767 | 5.060 | 0.233 | 0.99941 | 766 $^{\pm\pm}$ | 5.186 | 0.255 | 0.90938 |
| s1196 | 1233 | 4.200 | -0.113 | 0.90742 | 1233 $^{\pm\pm}$ | 4.391 | -0.011 | 0.96564 |
| s1423 | 269 | 5.001 | -0.730 | 0.96881 | 269 $^{\pm\pm}$ | 4.415 | -0.684 | 0.97374 |
| s1488 | 212 | 3.877 | -0.278 | 0.96352 | 209 $^{\pm\pm}$ | 4.157 | -0.260 | 0.89162 |
| s9234 | 1190 | 7.129 | -0.868 | 0.87967 | 1132 $^{\pm\pm}$ | 6.577 | -0.843 | 0.90058 |
| s13207 | 2334 | 9.316 | -0.593 | 0.89989 | 2341 $^{\pm\pm}$ | 9.124 | -0.523 | 0.90784 |
| s38417 | 789 | 18.471 | -1.327 | 0.98811 | 784 $^{\pm\pm}$ | 18.311 | -1.250 | 0.98789 |

# Chapter 7

# Conclusions and Future directions

This Chapter first gives an overall thesis summary with the main contributions and results of this dissertation, followed by short and long term research directions.

## 7.1 Thesis summary

Increasing complexity of modern digital circuits and current manufacturing processes limit testing efficiency using traditional fault models. A circuit with correct timing, proved by applying appropriate input stimuli, is said to be delay-verifiable/correct. The well established Path Delay Fault (PDF) model is the most accurate delay fault model since it can detect both lumped and distributed delay defects. This thesis addresses the identification of the testable PDFs, single and multiple, and generation of suitable test sets for these PDFs. This is currently an open problem, mainly due to the huge number of PDFs that exist, even when restricting to the critical PDF set. Both, fully scanned circuits and circuits with no scan capabilities, e.g., microprocessors, are considered. This dissertation also examines the correlation between the physical paths of a digital circuit. This problem has important implications in various design automation problems, such as timing analysis, test generation and diagnosis. When considering the complexity and tight timing constraints of modern circuits, this correlation affects both the design process and the testing approaches followed in manufacturing. A major underlying tool used in this thesis is decision diagrams. A variant of decision diagrams, ZBDDs, has been previously shown to be very efficient in representing a huge number of PDFs. Hence, this thesis utilizes ZBDDs along with newly proposed diagrams and necessary operators to tackle the examined problems. This thesis focuses on the exploitation of the four main problems discussed below.

The *first problem* exploited in this thesis is the identification of the all (critical) singly testable PDF set in combinational or enhanced fully-scanned sequential circuits under different sensitization criteria and the generation of a compact test set for the identified PDF set. The major challenge was the large number of PDFs considered and the fanout stems exploitation when restricting to the critical PDF set. The methodology proposed takes as an input a set of potentially testable (critical) PDFs and returns a specialized BDD that incorporates all single testable (critical) PDFs along with their test cubes and a single routed DAG for deriving a compact test. The main *motivation* in this problem was to investigate how the generated specialized BDD, could be efficiently manipulated to derive a compact test set. The *major contributions* of this work is the specialized BDD that incorporates all (critical) singly sensitizable PDFs of a circuit in an implicit and compact manner. Function formulations are given for three sensitization types namely robust, non-robust and functionally sensitization. Moreover, an efficient methodology (T-Graph generation) for deriving a compact test set was found. Experimental results clearly demonstrate the practicality of the method and its superiority over existing methods in terms of high test efficiency (compact test) especially for the critical PDF set.

The *second problem* studied in this thesis is the identification of the multiple testable (critical) primitive PDF set in enhanced fully-scanned sequential circuits. The issue here is the huge number of paths/combinations of paths considered, which can be doubly-exponential in the worst case (see complexity analysis in Chapter 2) for the primitive multiple PDF identification, even when the problem restricts to the critical set. The input to the methodology used is a set of potentially single critical PDFs and the multiple critical primitive PDF set is the methodology's output. The main *motivation* for this problem is that it has been previously shown that in order to guarantee the temporal correctness of a circuit, only the primitive PDF set needs to be tested. No existent methods can be trivially extended to identify primitive faults using ZBDDs, since the standard ZBDD operators cannot handle multiple faults. New operators, polynomial to the size of the ZBDD, are developed for the efficient and non-enumerative manipulation of the multiple faults and function-based formulations with appropriate data structures (ZBDDs) are utilized for implicit and compact representation and non-enumerative manipulation of PDFs. The *major contributions* of this work is the definition of the testable critical primitive PDFs and the efficient identification of faults with no enumeration considering any set of potentially testable critical PDF set. A data structure is generated that represents the targeted faults also contains ATPG data (thus easily incorporated in a very efficient test generation framework). Experimental results show that only a

small number of multiple primitive PDFs is testable, thus only a small number of additional tests suffices to guarantee the circuit's timing correctness under the multiple fault criterion

The *third problem* examined in this dissertation explores the generation of path-delay test programs for microprocessor cores, with no scan capabilities thus the functional delay test application methodology is used. The input to the methodology is the RT and gate (logic) description of the microprocessor and the output is a test program set (Instruction Sequences) that can efficiently detect PDFs. The *motivation* for this problem is to investigate how existing evolutionary computation algorithms can take advantage of a BDD-based fitness evaluation function for guided generation of test programs. An evolutionary algorithm runs RT level simulations and BDDs for deriving fault excitation conditions is constructed based on the gate level description of the microprocessor core, that computes a fitness value. This newly introduced BDD-based fitness evaluation function directs the test programs generation flow towards an optimal solution. If for a PDF a BDD representation can not be found this implies that the PDF is un-testable and thus is droped. The *major contributions* of this work is BDDs are used for deriving fault excitation conditions, exploited for the automatic generation of test programs able to excite and propagate fault effects, based on an evolutionary algorithm and fast RTL simulation innovative approach for the generation of functional programs to test PDFs within microprocessors cores. Experimental results demonstrate that this methodology allows reducing the test generation time, by concentrating on suitably classified structurally coherent fault lists and avoiding computation-intensive gate-level simulations.

The *last problem* examined in this work is the identification of the pairwise physical paths correlation between the paths in a set, a difficult to produce a metric. The input to this problem is a set of circuit paths and the output is a comprehensive statistical characterization for the given path set. The *motivation* for this problem is that the correlation between the paths of an integrated circuit has important implications in various design automation problems, such as timing analysis, test generation and diagnosis. With the increasing complexity and tight timing constraints of modern circuits, this correlation affects both the manufacturing design process and testing. The methodology proposed is based on standard ZBDD operations that gives a comprehensive statistical characterization for a given path set in polynomial complexity that avoids enumeration. The *major contribution* of this work is a methodology that avoids enumeration of paths or path segments and, hence, a large number of paths can be considered in practical time. The proposed ZBDD method is has a polynomial, to the size of the diagram, complexity. Effectiveness through experimentation that, show how the

proposed technique identifies similarities among various input/output critical path sets and can distinguish their characteristics based on just two measure values per test. Effectiveness is also demonstrated through another experiment using the proposed methodology that compares the propagation paths between the multiple-detect and the n-detect test sets in relation to their corresponding defect coverage.

## 7.2 Future work

Some future directions and motivations are given below, to follow relevant research areas, extracted while working on this thesis.

In Chapters 3 and Chapter 4 a specialized BDD data structure, ISOP/ZBDD, has been proposed that incorporated all the single and multiple PDFs along with their sensitizing test cubes. This data structure could be explored to identify and exploit graph properties such as the connectivity and the diameter. It is believed that important conclusions can be extracted regarding:

- An optimized variable ordering, in terms of number of nodes, for the produced decision diagram can be investigated. Some indications point out that if the variable order used in the BDD, follows a similar pattern with the topological traversal of the circuit examined, this actually produces a ”good” variable ordering.

- Important graph properties may produce a ”better critical” PDF set, e.g, point out which path segment or path are the most visited/common, thus must be tested. This information may be extracted by appropriate manipulations/operations on the produced data structure.

- Another important problem it to try to explore this data structure for minimizing the number of tests used by changing the test generation policy. Instead of trying to find the test with the maximum number of newly detected PDFs per test, try to find a test set that overall this set would had the smallest cardinality. The target of the previous methodology used throughout this thesis, the $\mathscr{T}-$graph generation approach, is to find the ”best” test by test approach instead of considering all the tests as a set and trying to minimizing the set as a whole.

- Important conclusions and information can be extracted with respect a circuit's internal structure. Using the specialized BDD structure, important circuits gates or segments or

even paths could be pointed out where the probability that an error will occur increases. This information would be critical for generating a ”suitable” test set for this circuit.

- Low power test generation would be another interesting problem to look at. This specialized BDD data structure extracts compact tests, that is tests that can check a large number of PDFs together. This would imply a lot of switching of transitions, thus this would need more power. This may result in a violation of power, where overheating could occur.

In Chapter 5 a methodology for guided generation of path-delay test programs for microprocessor cores with no scan capability was presented. This approach, processes each PDF separately. An interesting problem would be to investigate how existing evolutionary computation algorithms can take advantage of a BDD-based fitness evaluation function for a set of PDFs together. To solve this, a technique to select appropriate PDF sets, would be needed and it would be interesting if this could be exploited using decision diagrams.

Finally, Chapter 6 gives a comprehensive statistical characterization for the given path set. Additional methodologies could be explored for providing a metric that may concentrate and would make better sense if it addresses a specific area, e.g., a generic metric for test compactness.

# Bibliography


[1] N. Ahmed, M. Tehranipoor and V. Jayaram, "Timing-based delay test for screening small delay defects.", *Proc. of DAC*, pp. 320-325, 2006.

[2] N. Ahmed, M. Tehranipoor, C.P. Ravikumar, "Enhanced Launch-Off-Capture Transition Fault Testing.", *Proc. of ITC*, Paper 11.1, 2005.

[3] S. B. Akers, "Binary Decision Diagrams.", *IEEE Transactions on Computers*, Vol. 27(6), pp. 509-516, June 1978.

[4] S. B. Akers, "Failure diagnosis of structured VLSI.", *IEEE Design & Test of Computers*, Vol. 6(4), pp. 49-60, June 1989.

[5] B. Arslan and A. Orailoglu, "Delay test quality maximization through process-aware selection of test set size", *Proc. of ICCD*, pp. 390-395, October 2010.

[6] K. Bazargan, N. Selvakkumaran, G. Karypis and C. Ababei, "Multi-objective circuit partitioning for cutsize and path-based delay minimization.", *Proc. of ICCAD*, pp. 181-185, 2002.

[7] J. Benkoski, E. V. Meersch, L. J. M. Claesen and H. De Man,"Timing Verification using statically sensitizable paths", *IEEE Trans. on CAD*, Vol. 9, pp. 1073-1084,September 1990.

[8] P. Bernardi, K. Christou, M. Grosso, M. K. Michael, E. Sanchez and M. S. Reorda, "Exploiting MOEA to Automatically Generate Test Programs for Path-Delay Faults in Microprocessors", in Springer Lecture Notes in Computer Science, Vol. 4974/2008, pp. 224-234.

[9] P. Bernardi, M. Grosso, E. Sanchez and M. S. Reorda,"On the Automatic Generation of Test Programs for Path-Delay Faults in Microprocessor Cores.", *IEEE ETS*, pp. 179184, 2007.

[10] D. Bhattacharya, P. Agrawal and V. D. Agrawal, "Test Pattern Generation for Path Delay Faults using Binary Decision Diagrams", *IEEE Trans. on Computers*, Vol. 44(3), pp. 434-447, March 1995.

[11] S. Bose, P. Agrawal, and V. D. Agrawal, "Generation of compact delay tests by multiple path activation", *Proc. of ITC*, pp. 714-723, 1993.

[12] S. Bose and V. D. Agrawal, "Sequential Logic Path Delay Test Generation by Symbolic Analysis", *Proc. of AST*, pp. 353-359, Nov. 1995.

[13] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation", *IEEE Trans. on Computers*, Vol. C-35(8), pp. 677-691, August 1986.

[14] R. E. Bryant,"Symbolic Boolean manipulation with ordered binary-decision diagrams.", *ACM Comput. Surv.*, Vol. 24(3), pp. 293-318, September 1992.

[15] R. E. Bryant and Y.A. Chen, "Verification of arithmetic circuits using binary moment diagrams", *Proc. CAD*, pp. 535-541, 1995.

[16] M. L. Bushnell and V. D. Agrawal, "Essentials of Electronic Testing for Digital, Memory & Mixed-Signal VLSI Circuits", *Kluwer Academic Publishers, Dordrecht*, 2000.

[17] T. J. Chakraborty, V. D. Agrawal and M. L. Bushnell, "Delay fault models and test generation for random logic sequential circuits.", *ACM/IEEE Design Automation Conference*, pp. 165-172, 1992.

[18] L. Chen and S. Dey, "Software-Based Self-Testing Methodology for Processor Cores.", *IEEE Trans. on CAD*, Vol. 20(3), pp. 369-380, March 2001.

[19] C. A. Cheng and S. K. Gupta, "Test generation for path delay faults based on satisfiability", *IEEE Design Automation Conference*, 1996.

[20] K. T. Cheng, "Transition Fault Testing for Sequential Circuits", *IEEE Trans. on CAD*, Vol. 12(12), pp. 1971-1983, December 1993.

[21] K.T. Cheng and H.C. Chen, "Classification and Identification of Nonrobust Untestable Path Delay Faults", *IEEE Trans. on CAD*, Vol. 15, pp. 845-853, August 1996.

[22] K. Christou, M. K. Michael, P. Bernardi, M. Grosso, E. Sanchez and M. S. Reorda, "A Novel SBST Generation Technique for Path-Delay Faults in Microprocessors Exploiting Gate- and RT-Level Descriptions",*Proc. of IEEE VLSI Test Symposium*, pp.389-394, 2008.

[23] K. Christou, M. K. Michael and S. Neophytou,"Identification of critical primitive path delay faults without any path enumeration", *Proc. of IEEE VLSI Test Symposium*, pp.9-14, 2010.

[24] K. Christou, M. K. Michael and S. Tragoudas,"Implicit Critical PDF Test Generation with Maximal Test Efficiency",*Proc. of IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, pp.50-58, 2006.

[25] K. Christou, M. K. Michael and S. Tragoudas., "On the Use of ZBDDs for Implicit and Compact Critical Path Delay Fault Test Generation", *Journal of Electronic Testing*, Vol. 24, pp. 203-222, January 2008.

[26] M. K. Michael, K. Christou and S. Tragoudas,"Towards finding path delay fault tests with high test efficiency using ZBDDs",*Proc. of IEEE International Conference on Computer Design*, pp.464-467, 2005.

[27] J. Chung, J. Xiong, V. Zolotov, and J. A. Abraham, "Path Criticality Computation in Parameterized Statistical Timing Analysis Using a Novel Operator.", *IEEE Trans. on CAD*, Vol. 31(4), April 2012.

[28] C. A. CoelloCoello, D. A. Van Veldhuizenand and G. B. Lamont,"Evolutionary Algorithms for Solving Multi-Objective Problems", *Kluwer Academic Publishers, Dordrecht*, 2002.

[29] F. Corno, E. Sanchez and G. Squillero, “Evolving Assembly Programs: How Games Help Microprocessor Validation”, *IEEE Trans. on Evolutionary Computation*, Vol. 9, pp. 695706, 2005.

[30] R. Drechsler and D. Sieling, “Binary Decision Diagrams in theory and practice”, *Int’l Journal on STTT*, 3, pp. 112 - 136, 2001.

[31] B. Dervisoglu and G. Stong, “Design for Testability: Using Scanpath Techniques for Path-Delay Test and Measurement.”, *Proc. of ITC*, pp. 365-374, 1991.

[32] K. Fuchs, M. Pabst and T. Roessel, “RESIST: A Recursive Test Pattern Generation Algorithm.”, *IEEE Trans. on CAD*, Vol.13(12), pp. 15501561, 1994.

[33] F. Galarza, J. Garcia, V. H. Champac, A.Orailoglu “Small-Delay Defects Detection Under Process Variation Using Inter-Path Correlation.”, *Proc. of VTS*, 2012.

[34] M. A. Gharaybeh, M. L. Bushnell and V. D. Agrawal, “Classification and modeling of path delay faults and Test Generation Using Single Stuck-Fault Tests”, *Proc. of ITC*, pp. 139-148, October 1995.

[35] S. Gurumurthy, R. Vemu, J. A. Abraham and D. G. Saab, “Automatic Generation of Instructions to Robustly Test Delay Defects in Processors”, *IEEE ETS*, pp. 173178, 2007.

[36] S. Hamilton, “Taking Moore’s law into the next century.”, *IEEE Computer*, Vol. 32(1), pp. 43-48, 1999.

[37] K. R. Heloue, S. Onaissi, and F. N. Najm, “Efficient Block-Based Parameterized Timing Analysis Covering All Potentially Critical Paths.”, *IEEE Trans. on CAD*, Vol. 31(4), April 2012.

[38] K. Heragu, J. H. Patel and V. D. Agrawal, “Fast identification of untestable delay faults using implications”, *Proc. CAD*, pp. 642-647, 1997.

[39] K. Heragu, J. H. Patel and V. D. Agrawal, “Segment delay faults: a new fault model”, *Proc. of VLSI Test Symp.*, pp. 32-39, 1996.

[40] K. Heragu, J. H. Patel and V. D. Agrawal, “SIGMA: A Simulator for Segment Delay Faults”, *Proc. of International Conf. CAD*, pp. 502-508, 1996.

[41] S. Y. Huang and K. T. Cheng, “ErrorTracer: design error diagnosis based on fault simulation techniques.”, *IEEE Trans. on CAD*, Vol. 18(9), 1999.

[42] T. Huaxing, C. Gang, S. M. Reddy, W. Chen, J. Rajski and I. Pomeranz, “Defect aware test patterns.”, *Proc. of DFTS*, pp. 450-455, 2005.

[43] S. Huband, P. Hingston, L. Barone and L. While, “A Review of Multiobjective Test Problems and a Scalable Test Problem Toolkit”, *IEEE Trans. on Evolutionary Computation*, Vol. 10(5), pp. 477-506, 2006.

[44] V. S. Iyengar, B. K. Rosen and J. A. Waicukauski, “On Computing the Sizes of Detected Delay Faults”, *IEEE. Trans. on TCAD*, pp. 299-312, March 1990.

[45] A. M. Jabir and D. K. Pradhan “A Graph-Based Unified Technique for Computing and Representing Coefficients over Finite Fields”, *IEEE Trans. on Computers*, Vol. 56(8), August 2007.

[46] A. M. Jabir, D. K. Pradhan, A. K. Singh, and T.L. Rajaprabhu "A Technique for Representing Multiple-Output Binary Functions with Applications to Verification and Simulation", *IEEE Trans. on Computers*, Vol. 56(8), August 2007.

[47] S. Kajihara, M. Fukunaga, W. Xiaoqing, T. Maeda, S. Hamada, and Y. Sato, "Path Delay Test Compaction with Process Variation Tolerance" *Proc. DAC*, pp. 845-850, June 2005.

[48] D. Karayiannis and S. Tragoudas, "A Fast Non-enumerative Automatic Test Pattern Generator for Path Delay Faults", *IEEE Trans. on CAD*, Vol. 18(7), pp. 1050-1057, July 1999.

[49] K. S. Kim, S. Mitra and P. G. Ryan, "Delay defect characteristics and testing strategies.", *IEEE Design & Test of Computers*, Vol. 20(5), pp. 8-16, 2003.

[50] M. M. V. Kumar and S. Tragoudas, "High-Quality Transition Fault ATPG for Small Delay Defects.", *IEEE Trans. on CAD*, Vol. 26(5), pp. 983-989, 2007.

[51] W. Ke and P. R. Menon., "Delay-Verifiability of Combinational Circuits Based on Primitive Faults", *Proc. of ICCD*, pp. 86-90, 1994.

[52] W. Ke and P. R. Menon., "Synthesis of Delay-Verifiable Combinational Circuits.", *IEEE Trans. on Comp.*, Vol. 44(2), pp. 213-222, February 1995.

[53] F. Kocan and M. H. Gunes, "On the ZBDD-based nonenumerative path delay fault coverage calculation",*IEEE Trans. on CAD*, Vol. 24, pp. 1137-1143, July 2005.

[54] D. A. Kirkpatrick and A. L. Sangiovanni-Vincentelli, "Digital Sensitivity: Predicting Signal Interaction using Functional Analysis", *Proc. ICCAD*, pp. 536-541, November 1996.

[55] A. Krstic and K. T. Cheng, "Delay fault Testing for VLSI circuits.", *Kluwer Academic Publishers, Dordrecht*, 1998.

[56] A. Krstic, K. T. Cheng and S. T. Chakradhar, "Primitive Delay Faults: Identification, Testing, and Design for Testability.", *IEEE Trans. on CAD*, Vol. 18(6), pp. 669-684, June 1999.

[57] A. Krstic and K. T. Cheng, "Resynthesis of Combinational Circuits for Path Count Reduction and for Path Delay Fault Testability", *Proc. of European Design and Test*, pp. 486-490, March 1996.

[58] W. C. Lai, A. Krstic and K. T. Cheng, "On Testing the Path Delay Faults of a Microprocessor Using its Instruction Set.", *Proc. of VTS*, 2000.

[59] W. C. Lai, A. Krstic and K. T. Cheng, "Test Program Synthesis for Path Delay Faults in Microprocessor Cores", *IEEE International Test Conference*, pp. 1080-1089, 2000.

[60] C. Y. Lee, "Representation of Switching Circuits by Binary-Decision Programs.", *The BELL SYSTEM TECHNICAL JOURNAL*, pp. 985-999, July 1959.

[61] C. J. Lin and S. M. Reddy, "On Delay Fault Testing in Logic Circuits", *IEEE TCAD*, Vol.6, pp. 694-703, September 1987.

[62] Z. C. Li, R. K. Brayton and Y. Min, "Efficient identification of Non-Robustly Untestable Path Delay Faults using implications", *Proc. ITC*, pp. 992-997, 1997.

[63] X. Lin, K. H. Tsai, C. Wang, M. Kassab, J. Rajski, T. Kobayashi, R. Klingenberg, Y. Sato, S. Hamada and T. Aikyo, "Timing-Aware ATPG for High Quality At-speed Testing of Small Delay Defects", *Proc. of ATS*, pp. 139-146, 2006.

[64] J. J. Liou. A. Krstic, Y. M. Jiang and K. T. Cheng, "Path selection and pattern generation for dynamic timing analysis considering power supply noise effects", *Proc. of ICCAD*, pp. 493-496, 2000.

[65] X. Lu, Z. Li, W. Qiu, D. M. H. Walker and W. Shi, "Longest-path selection for delay test under process variation", *IEEE Trans. on CAD*, Vol. 24(12), pp. 1924-1929, December 2005.

[66] S. Y. Lu, P. Y. Hsieh and J. J. Liou, "Exploring linear structures of critical path delay faults to reduce test efforts", *Proc. of ICCAD*, pp. 100-106, 2006.

[67] A. K. Majhi, J. Jacob, L. M. Patnaik and V. D. Agrawal, "On test coverage of path delay faults", *Proc. of VLSID*, 1996.

[68] S. Majumder, V. D. Agrawal and M. L. Bushnell, "Path Delay Testing: Variable-clock versus Rated-clock", *Proc. of the 11th International Conference on VLSI Design*, pp.470-475, Jan. 1998.

[69] T. M. Mak, A. Krstic, K. T. Cheng and Li. C. Wang, "New challenges in delay testing of nanometer, multigigahertz designs.", *IEEE Design & Test of Computers*, Vol. 21(3), pp. 241-248, 2004.

[70] P. Manikandan, B. B. Larsen and E. J. Aas, "Experiments with ABIST test methodology applied to path delay fault testing.", *Design & Test Symposium (EWDTS)*, pp. 59-63, 2010.

[71] E. J. McCluskey and T. Chao-Wen, "Stuck-fault tests vs. actual defects.", *Proc. of ITC*, pp. 336-342, 2000.

[72] P. C. McGeer, A. Saldanha, P. R. Stephan, R. K. Brayton and A. L. Sangiovanni-Vincetelli, "Timing Analysis and Delay Fault Test Generation using Path Recursive Functions", *Proc. CAD*, pp. 180-183, 1991.

[73] P. C. McGeer and R. K. Brayton, "Integrating Functional and Temporal Domains in Logic Design", *Kluwer Academic Publishers, Dordrecht*, 1991.

[74] M. K. Michael and S. Tragoudas, "Function-based Compact Test Pattern Generation for Path Delay Faults", *IEEE Trans. on VLSI*, Vol.13(8), pp. 996-1001, April 2005.

[75] M. K. Michael, K. Christou and S. Tragoudas, "Towards finding path delay fault tests with high test efficiency using ZBDDs", *Proc. ICCD*, pp. 464-467, 2005.

[76] A. Mishchenko, "EXTRA CUDD:http://web.cecs.pdx.edu/ alanmi/ research/extra.htm". 2003.

[77] S. I. Minato, "Fast Generation of Prime-Irredundant Covers from Binary Decision Diagrams", *IEICE Trans. Fundamentals*, Vol. E76-A(6), pp. 976 - 973, June 1993.

[78] S. I. Minato, "Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems", *Proc. DAC*, pp. 272-277, 1993.

[79] S. I. Minato, “Fast Factorization method for implicit cube set representation”, *IEEE Trans. on CAD*, Vol. 15(3), pp. 377-384, April 1996.

[80] A. Narayan, J. Jain, M. Fujita and A. Sangiovanni-Vincentelli, “Partitioned ROBDDs - A Compact, Canonical and Efficiently Manipulable Representation for Boolean Expressions”, *Proc. ICCAD*, pp. 547-554, 1996.

[81] S. Neophytou, K. Christou and M. K. Michael,“An Approach for Quantifying Path Correlation in Digital Circuits without any Path or Segment Enumeration”, *Proc. of IEEE European Test Symposium*, pp.141-146, 2011.

[82] S. Neophytou and M. K. Michael, “Hierarchical Fault Compatibility Identification for Test Generation with a Small Number of Specified Bits.”, *Proc. of DFT*, pp. 439-447, 2007.

[83] S. Neophytou, M. K. Michael and K. Christou, “Generating Diverse Test Sets for Multiple Fault Detections Based on Fault Cone Partitioning.”, *Proc. of DFTS*, pp. 401-409, 2009.

[84] S. Padmanaban, M. K. Michael and S. Tragoudas, “Exact path delay fault coverage with fundamental ZBDD operations”, *IEEE Trans. on CAD*, Vol. 22, pp. 305-316, March 2003.

[85] S. Padmanaban and S. Tragoudas, “Efficient Identification of (Critical) Testable Path Delay Faults Using Decisions Diagrams”, *IEEE Trans. on CAD*, Vol. 24(1), pp. 77-87, January 2005.

[86] S. Padmanaban and S. Tragoudas, “Non-Enumerative Path Delay Fault Diagnosis”, *Proc. of DATE*, pp. 10322-10327, 2003.

[87] I. Pomeranz and S. M. Reddy, “Forming N-detection test sets without test generation.”, *ACM TODAES*, Vol. 12(2), No. 18, April 2007.

[88] I. Pomeranz, and S. M. Reddy, “On Synthesis-for-Testability of Combinational Logic Circuits”, *Proc. of DAC*, pp. 126-132, June 1995.

[89] I. Pomeranz and S. M. Reddy, “Sizes of test sets for path delay faults using strong and weak non-robust tests.”, *IET Comput. Digit. Tech*, Vol. 5(5), pp. 405-414, September 2011.

[90] I. Pomeranz, and S. M. Reddy, “Test Enrichment for Path Delay Faults Using Multiple Sets of Target Faults”, *IEEE Trans. on CAD*, Vol. 22(1), pp. 82-89, January 2003.

[91] I. Pomeranz, S. M. Reddy and P. Uppaluri, “NEST: A Non-enumerative Test Generation Method for Path Delay Faults in Combinational Circuits”, *IEEE Trans. on CAD*, Vol. 14(12), pp. 1505-1515, December 1995.

[92] D. K. Pradhan and C. Liu “EBIST: A Novel Test Generator With Built-In Fault Detection Capability”, *IEEE Trans. on CAD*, Vol. 24(9), September 2005.

[93] A. K. Pramanick and S. M. Reddy, “On the fault coverage of gate delay fault detecting tests”, *IEEE Trans. CAD*, Vol. 16(1), pp. 78-94, January 1997.

[94] R. Putman and R. Gawde, “Enhanced timing-based transition delay testing for small delay defects”, *Proc. of VTS*, 2006.

[95] E. Sanchez, M. Schillaci, M. Sonza Reorda and G. Squillero,"An Enhanced Technique for the Automatic Generation of Effective Diagnosis-oriented Test Programs for Processors", *IEEE Design, Automation and Test in Europe*, pp. 16, 2007.

[96] T. Sasao and J. T. Butler, "Worst and best irredundant sum-of-products expressions", *IEEE Trans. on Computers*, Vol. 50(9), pp. 935-948, September 2001.

[97] J. Savir and S. Patil, "On Broad-Side Delay Test.", *Proc. of VTS*, pp. 284-290, 1994.

[98] J. Savir, "Skewed-Load Transition Test: Part I, Calculus.", *Proc. of ITC*, pp. 705-713, 1992.

[99] J. Saxena and D. K. Pradhan, "A method to derive compact test sets for path delay faults in combinational circuits", *Proc. of ITC*, pp. 724-733, 1993.

[100] J. D. Schaffer,"Multiple Objective Optimization with Vector Evaluated Genetic Algorithms", *Intl Conf. on Genetic Algorithms and Their Applications*, pp. 93100, 1985.

[101] Y. Shao, S. M. Reddy, S. Kajihara and I. Pomeranz, "An Efficient method to identify Untestable Path Delay Faults", *Proc. ITC*, 2001.

[102] M. Sharma and J. H. Patel, "Testing of critical paths for delay faults", *Proc. of ITC*, pp. 634-641, 2001.

[103] T. Shinogi, T. Hayashi, T. and K. Taki, "Test generation for stuck-on faults in BDD-based pass-transistor logic SPL", *ATS*, pp. 16-21, Nov. 1997.

[104] V. Singh, M. Inoue, K. K. Saluja and H. Fujiwara, "Instruction-Based Delay Fault Self-Testing of Processor Cores", *IEEE International Conference on VLSI Design*, pp. 933938, 2004.

[105] M. Sivaraman and A. Strojwas, "Primitive Path Delay Faults:Identification and Their Use in Timing Analysis.", *IEEE Trans. on CAD*, Vol. 19(11), pp. 1347-1362, November 2000.

[106] G. L. Smith, "Model for Delay Faults Based upon Paths", *Proc. of ITC*, pp. 342-351, 1985.

[107] F. Somenzi, "CUDD: CU Decision Diagram Package", Dept. of ECE, The University of Colorado, release 2.3.0, 1999.

[108] U. Sparmann, D. Luxenburger, K. T. Cheng and S. M. Reddy,"Fast identification of robust dependent path delay faults", *Proc. of DAC*, pp. 119-125, 1995.

[109] P. Tafertshofer, A. Ganz and K. J. Antreich, "IGRAINE – An Implication GRaph bAsed engINE for fast Implication, Justification, and Propagation", *IEEE Trans. on CAD*, Vol. 19(8), pp. 907927, 2000.

[110] S. Tani, M. Teramoto, T. Fukazawa and K. Matsuhiro, "Efficient path selection for delay testing based on partial path evaluation.", *Proc. of VTS*, pp. 188-193, 1998.

[111] R. Tayade and J. Abraham, "Small-delay defect detection in the presence of process variations.", *Microelectronics Journal*, Vol. 39, pp. 1093-1100, 2010.

[112] R. Tayade and A. J. Abraham, "Critical Path Selection for Delay Testing Considering Coupling Noise.", *JETTA*, Vol. 25(4), pp. 213-223, 2009.

[113] R. C. Tekumalla and P. R. Menon, “Test generation for primitive path delay faults in combinational circuits”, *Proc. of ICCAD*, pp. 636-641, 1997.

[114] L. C. Wang, J. J. Liou and K. T. Cheng, “Critical Path Selection for Delay Fault Testing Based Upon a Statistical Timing Model”, *IEEE Trans. on CAD*, Vol. 23(11), pp. 1550-1565, November 2004.

[115] D. Xiang, K. Li, H. Fujiwara and J. Sun, “Generating compact robust and non-robust tests for complete coverage of path delay faults based on stuck-at tests.”, *Proc. ICCD*, pp. 446-451, 2006.

[116] H. Yan and A. D. Singh , “Evaluating the effectiveness of detecting delay defects in the slack interval: a simulation study”, *Proc. of ITC*, pp. 242-251, 2004.

[117] K. Yang, K. T. Cheng and L. C. Wang, “TranGen: A SAT-Based ATPG for Path-Oriented Transition Faults”, *ASP-DAC*, pp. 92-97, 2004.

[118] M. Yilmaz, K. Chakrabarty and M. Tehranipoor, “Test-Pattern Selection for Screening Small-Delay Defects in Very-Deep Sub-micrometer Integrated Circuits.”, *IEEE Trans. on CAD*, Vol. 29(5), 2010.

[119] V. Zolotov, X. Jinjun, H. Fatemi and C. Visweswariah, “Statistical Path Selection for At-Speed Test.”, *IEEE Trans. on CAD*, Vol. 29, pp. 749-759, 2010.

[120] “International Technology Roadmap for Semiconductors.”, *tech. rep., ITRS*, 2011.

[121] *MicroGP++*, “http://ugp3.sourceforge.net”