High Quality Test Pattern Generation Techniques for Digital VLSI Circuits

by

Stelios N. Neophytou

Diploma in Computer Engineering and Informatics, 2003

A Dissertation
Submitted in Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy
at the University of Cyprus

Recommended for Acceptance by the
Department of Electrical and Computer Engineering
University of Cyprus

March, 2009

AN ABSTRACT OF THE DISSERTATION OF

Stelios N. Neophytou, for the Doctor of Philosophy degree in Computer Engineering, presented on 28 of January 2009, at the University of Cyprus
TITLE: High Quality Test Pattern Generation Techniques for Digital VLSI Circuits
SUPERVISOR PROFESSOR: Dr. Maria K. Michael

While traditional fault models, such as the stuck-at and transition delay fault models are still widely used, they have been shown to be inadequate to handle the increased complexity of modern digital integrated circuits. The goal of this dissertation is to provide a set of novel test generation methodologies that increase the quality of post-manufacturing tests for digital circuits, without increasing the complexity of the underlying fault models.

The first part of the thesis examines test generation under a linear variation of the transition delay fault model for obtaining transition tests that are robust and excite critical path delays in the circuit. Thus, an enhanced quality transition delay fault model is considered and a method to implicitly derive all tests per transition fault, using well defined fault sensitization criteria each of which provides different detection quality, is proposed.

The thesis also examines the problem of generating test patterns with a large number of unspecified bits. The inherent flexibility of such test sets benefit different applications in VLSI circuit testing. After appropriate unspecified bit fixing, the obtained test sets can provide desired solutions to special applications and, thus, are of higher quality. Two versions of this problem are examined: dynamic test generation and static test set relaxation. For the dynamic problem, two different algorithms, based on fault compatibility properties represented by an appropriate undirected acyclic graph, are proposed. The obtained results give very compact test sets with a large number of unspecified bits. Under the static problem, two different techniques have been developed to maximize the number of unspecified bits in the test set, without compromising the fault coverage or increasing the test set size. Experimental results show increased reduction rates compared to existing methods, even when the input test set is very compact or already contains unspecified bits. The impact of using test sets with small number of specified bits for on-chip test set embedding is also investigated. Two popular Built-In Self-Test schemes for deterministic test set embedding, are used to demonstrate considerable reduction to the total number of on-chip bits required to encode the generated test set.

The last part of the thesis investigates test set generation and relaxation for "n-detect" test sets to increase the quality of such test sets. First, the novel problem of relaxing n-detect test sets is addressed. A systematic algorithm is proposed, where each test is replaced by a new one that detects a subset of the faults detected by the first one. This new test has fewer specified bits, yet the replacement algorithm ensures that the n-detect fault coverage is maintained. Next, the thesis proposes a novel technique which guarantees diversity in the n different tests that target the same fault, by propagating the fault via different propagation paths. This is achieved by a linear and systematic partitioning of the circuit, into non-overlapping propagation sub-circuits. The experimental results show increased coverage in non-modeled faults without invalidating any desired attributes of the initial test set.

ΠΕΡΙΛΗΨΗ

Η εξέλιξη της τεχνολογίας ολοκλήρωσης επέτρεψε την υλοποίηση ολοκληρωμένων κυκλωμάτων (μικροτσίπ) με εκπληκτικές δυνατότητες. Η δραματική αύξηση της πολυπλοκότητας των ολοκληρωμένων, πολλές φορές δυσανάλογα ως προς το όφελος απόδοσης, είναι το φυσιολογικό τίμημα για το σημαντικό αυτό επίτευγμα. Τόσο η αύξηση της πολυπλοκότητας όσο και η πυκνότητα των τρανζίστορ στο ολοκληρωμένο, οδηγούν σε μεγαλύτερα ποσοστά ελαττωματικών ολοκληρωμένων κατά την παραγωγή. Τα προβλήματα που προκύπτουν σε σχέση με τον σχεδιασμό των ολοκληρωμένων είναι σημαντικά, όμως οι προκλήσεις που αφορούν τον έλεγχο της ορθότητάς τους, είναι ακόμη σημαντικότερες.

Λόγω της αυξανόμενης πολυπλοκότητας, τα παραδοσιακά μοντέλα σφαλμάτων αποδεικνύονται αναποτελεσματικά για την διασφάλιση της ποιότητας του ελέγχου ορθότητας στην παραγωγή ολοκληρωμένων κυκλωμάτων. Ο σκοπός της παρούσας διατριβής είναι να προτείνει ένα σύνολο από, υψηλής ποιότητας, καινοτόμες μεθοδολογίες αυτοματισμού της παραγωγής ελέγχων ορθότητας, για ψηφιακά κυκλώματα πολύ μεγάλης κλίμακας ολοκλήρωσης. Η ποιότητα ποσοτικοποιείται αναλόγως της εφαρμογής που εξετάζεται σε κάθε περίπτωση.

Το πρώτο μέρος της διατριβής πραγματεύεται την διαδικασία ελέγχου ορθότητας χρονισμού και παρουσιάζει μια αποδοτική και αποτελεσματική μεθοδολογία για την παραγωγή διανυσμάτων ελέγχου που δεν χάνουν την ικανότητα ανίχνευσης σφαλμάτων ακόμα και κάτω από διαφορετικές συνθήκες χρονισμού. Για το σκοπό αυτό θεωρούμε μια παραλλαγή του παραδοσιακού μοντέλου σφάλματος μετάβασης και προτείνουμε μεθοδολογίες για την παραγωγή όλων των διανυσμάτων ελέγχου με βάση καλώς ορισμένα κριτήρια ευαισθητοποίησης που παρέχουν διαφορετικής ποιότητας ανίχνευση.

Επιπλέον, η διατριβή εξετάζει το πρόβλημα της παραγωγής διανυσμάτων ελέγχου τα οποία έχουν ένα μεγάλο αριθμό από μη-καθορισμένες λογικές τιμές. Το πρόβλημα αυτό είναι ιδιαιτέρως σημαντικό για ένα αριθμό από εφαρμογές όπως, λ.χ. η μείωση κατανάλωσης ισχύος κατά τον έλεγχο καθώς και η συμπίεση, ο περιορισμός και ο εμπλουτισμός του συνόλου διανυσμάτων ελέγχου ορθότητας. Μελετήθηκαν δύο διαφορετικές εκδοχές του προβλήματος, (ι) δυναμική παραγωγή διανυσμάτων ελέγχου και (ιι) στατική χαλάρωση του συνόλου ελέγχου.

Για το δυναμικό πρόβλημα προτείνονται δύο αλγόριθμοι οι οποίοι χρησιμοποιούν ιδιότητες συμβατότητας σφαλμάτων οι οποίες αναπαριστώνται με ένα μη-κατευθυνόμενο ακυκλικό γράφο. Τα πειραματικά αποτελέσματα παρουσιάζουν σύνολα διανυσμάτων ελέγχων τα οποία είναι συνεπτυγμένου μεγέθους και, ταυτοχρόνως, περιέχουν μεγάλο αριθμό εισόδων με μη-καθορισμένες λογικές τιμές. Εξετάζεται επίσης, η επίπτωση της χρήσης αυτών των τεχνικών σε αρχιτεκτονικές ενσωμάτωσης του ελέγχου στο ολοκληρωμένο. Τα αποτελέσματα καταδεικνύουν αξιοσημείωτη μείωση στις απαιτήσεις σε επιπλέον υλικό για την κωδικοποίηση των παραγόμενων διανυσμάτων ελέγχου.

Για το στατικό πρόβλημα αναπτύχθηκαν δύο τεχνικές για την χαλάρωση ενός δοθέντος συνόλου διανυσμάτων ελέγχου, ώστε να περιέχει ένα μεγάλο αριθμό μη-καθορισμένων εισόδων χωρίς την μείωση του ποσοστού ανίχνευσης σφαλμάτων ή αύξηση του μεγέθους του. Οι τεχνικές αυτές αντικαθιστούν κάθε διάνυσμα ελέγχου, στο δοθέν σύνολο, με κάποιο εναλλακτικό το οποίο ανιχνεύει μόνο τον απαραίτητο αριθμό σφαλμάτων. Στα πειραματικά αποτελέσματα παρουσιάζονται υψηλά ποσοστά μείωσης, ακόμη και στις περιπτώσεις όπου το δοθέν σύνολο περιέχει ήδη εισόδους με μη-καθορισμένη τιμή.

Τέλος, μελετάται η παραγωγή και η χαλάρωση συνόλων διανυσμάτων ελέγχου ορθότητας με πολλαπλές ανιχνεύσεις για κάθε σφάλμα. Καταρχάς, διατυπώνεται, για πρώτη φορά, το πρόβλημα της χαλάρωσης υφιστάμενου συνόλου διανυσμάτων με πολλαπλές ανιχνεύσεις σφαλμάτων και προτείνεται ένας συστηματικός αλγόριθμος αντικατάστασης διανυσμάτων έτσι που να μειώνεται ο συνολικός αριθμός των εισόδων με καθορισμένη τιμή. Εν συνεχεία, προτείνεται μια πρωτότυπη τεχνική που εγγυάται ότι οι πολλαπλές ανιχνεύσεις σφαλμάτων θα είναι διαφορετικές. Ο αλγόριθμος περιγράφει ένα συστηματικό τρόπο για την κατάτμηση του μοντέλου του κυκλώματος, σε μη-αλληλοεπικαλυπτόμενα υπο-κυκλώματα, στα οποία πραγματοποιείται η παραγωγή των διανυσμάτων ελέγχου. Όπως φαίνεται από τα πειραματικά αποτελέσματα, τα παραγόμενα σύνολα διανυσμάτων ελέγχου παρέχουν αυξημένο ποσοστό ανίχνευσης μη-μοντελοποιημένων σφαλμάτων.

ACKNOWLEDGMENTS

This thesis is the result of my research activities during the last five years held at the Electrical and Computer Engineering Department, University of Cyprus. However, this work would not be possible without the precious help of many people, some of whom I mention here.

First, I would like to thank my advisor Dr. Maria Michael, Assistant Professor in ECE Department, for the continuous academic help she provided me through all these years. She was both a valuable colleague and an admonishing teacher.

Special thanks to all faculty and administrative staff of the Electrical and Computer Engineering Department, University of Cyprus for all kind of help that have provided during the years.

Furthermore, I am indebted to the members of my PhD examination committee Professor Christoforos Hadjicostis, Dr. Theocharis Theocharides (University of Cyprus), Dr. Vassos Soteriou (Cyprus University of Technology) and Professor Alex Orailoglu (University of California, San Diego), for their helpful suggestions.

I would, also, like to thank Professor Irith Pomeranz (Purdue University), Professor Nur Touba (University of Texas at Austin) and Professor Kohei Miyase (Japan Science and Technology Agency) which made available to us their tools and/or experimental data which helped in developing a number of the proposed techniques.

Last, but not least I would like to thank my wife Maria. Her patience, love and encouragement have upheld me, especially at those days that research was my only concern.

Parts of the research described in this thesis has been funded by the Cyprus Research Promotion Foundation under the program ΠΕΝΕΚ/ΕΝΤΑΞ/25 and a project funded by Intel Corporation.

Stelios Neophytou

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Integration technology improvement has allowed the realization of circuits with astonishing capabilities. The nanometer era is an incontestable fact with billions of transistors fit in an integrated circuit. The natural price paid for this achievement is the increase on the complexity of the circuits, in some cases disproportional to the benefit obtained. Both circuits' complexity and transistor density lead to larger defective parts ratio, which implies lower values for the manufacturing yield. According to the 2007 edition of the *International Technology Roadmap for Semiconductors* (ITRS) [1], in the near future, integrated circuits will have a number of cores, possibly of different types. Following the trace of the telecommunication evolution, future devices will be categorized as Systems-on-Chips (SoCs), System-in-Package (SiPs), Multi-Chip Packaging (MCP), among others. Hence, the trend is towards taking advantage of the large integration capabilities in order to put as many as possible components of a system in an integrated circuit. This is imposing a number of questions regarding the designing of an integrated circuit. At the same time, it gives rise to great challenges regarding the testing methodologies of such circuits. Modern digital circuits are accompanied with technologies such as RF, analog, optical etc [2, 3, 4]. The interaction of all these diverging components impose new issues on testing the whole integrated circuit, as well as on testing the digital part of the circuit [5, 6, 7]. Moreover, emerging multicore architectures are of increased complexity due, mainly, to their interconnection and synchronization structure as well as the memory access mechanism. These parameters need to be taken into consideration to ensure the quality of digital circuit testing [8, 9, 10]. Furthermore, since modern circuits run at very high frequencies, the need for low power testing arises in order to avoid circuit overheating while test application. Tight timing constraints imposed by technology scaling should also be accommodated effectively, without affecting the quality of the test application and/or on-line generation process [11, 12]. Considering all these issues, the *Test and Testability* chapter of the 2007 edition of the ITRS suggests that extended attention should be given to high quality testing procedures in order to maintain Moore's law and, at the same time, keep the testing cost small.

The increased complexity of digital ICs limits the testing efficiency of traditional fault models. While traditional fault models, such as the stuck-at and transition delay fault models, are still widely used, they have been shown inadequate to handle the new challenges [13]. The goal of this thesis is to provide a set of novel test generation methodologies which increase the quality of post-manufacturing tests for digital circuits. One important motivation of this thesis is to avoid the use of complex fault models that can introduce increased complexity in the test generation effort. Instead, the thesis investigates how traditional models, whose usage is already established in the industry, can be enhanced to increase test quality. Quality is measured based on the specific application under consideration. In the case of test set compaction and on-chip or off-chip compression applications, the measure of quality could be the size of the (compressed) test set. Alternatively, the quality of a single test pattern, typically, depends on the capability of the pattern to detect modeled, as well as non-modeled defects. In this case, some measures of quality include robustness of a test (guarantees that the applied test cannot get invalidated), test criticality (whether a test detects critical faults), encoding efficiency of a test, power dissipation, flexibility of enhancement for multiple fault detection etc. Each of the problems examined in this thesis considers at least one of the aforementioned quality measures.

Chapter 2 of the thesis examines test generation under the transition delay fault model and shows how to effectively and efficiently generate transition delay tests which are robust and excite critical path delays in the circuit. The proposed methodology considers an enhanced version of the traditional transition fault model. Thus, it proposes a method to implicitly derive all tests per transition fault, under established robust (and possibly other) fault sensitization criteria. The derived high quality test functions are further enhanced in three different ways to derive better quality test sets. The first enhancement restricts fault sensitization along critical sub-circuits whose paths have long delays under a fixed delay model. The second one manipulates the functions in order to generate compact test sets. The last one enriches the test set with additional test vectors, so that each new vector detects transition faults through different activation and propagation paths, without any path enumeration. The results of this work have already been published in [14, 15, 16].

Chapters 3 and 4 of the proposed thesis examine the problem of generation of test patterns with a large number of unspecified bits (bits with *don't care* value). This is a very important problem since it has implications to a number of applications, such as low power test (unspecified bits can be fixed appropriately to reduce power dissipation during test), test set compaction (a test pattern with large number of unspecified bits has a higher probability to detect additional faults), on-chip and off-chip test set compression (typically, the compression ratio increases as the number of unspecified bits in the test set increases), and test set enrichment for multiple fault detection or additional fault type detection. In these two chapters, systematic procedures that generate "flexible" test sets (test sets that have a large number of don't care bits) are proposed. Therefore, quality here is measured with respect to the number of unspecified bits in the generated test set. Further elaboration on the motivation is provided in the corresponding chapters. Two distinct problems are examined:

(i) static test set relaxation, i.e. increasing the number of unspecified bits in a given test set, and

(ii) dynamic test set generation with a large number of unspecified bits.

For problem (i), two different techniques have been developed to relax a given test set by maximizing the number of unspecified bits in the test set, without compromising the fault coverage or increasing the test set size (Chapter 3). The first method replaces each pattern in the test set with one targeting as few faults as necessary. The second method iterates among faults and tries to enforce detection of each fault by only a single test. This test is selected among all tests that detect that fault such that the largest overall specified bits reduction is achieved. Experimental results show increased reduction rates, when compared to existing methods, even when the input test set has been compacted or already contains unspecified bits. Part of this work has been published in [17].
While the proposed algorithms targeting the static test set relaxation problem result in test sets with a large number of unspecified bits, their effectiveness is always biased on the given test set. This is exactly the reason for examining problem (ii). In Chapter 4, two techniques are proposed to generate test sets with a large number of unspecified bits, without considering a previously generated test set as a basis. Both techniques are deterministic algorithms that consist of iterations on a constrained fault compatibility graph. The first algorithm identifies compatible faults that can be tested by a single test with a small number of specified bits by considering the entire graph per iteration. The second method is, essentially, a hi-

erarchical algorithm which considers a small part of the graph per iteration to find a locally optimized solution which is in term used in subsequent iterations. This hierarchical method is more scalable than the first one and, thus, more applicable to larger circuits. The obtained results give very compact test sets with a large number of unspecified bits. These approaches were presented in [18] and [19] and have been submitted for publication to a peer-reviewed international journal.

Extensive literature review, it is convincing that having test sets with a large number of unspecified bits can benefit techniques targeting a number of different problems including, but not limited to, test set compression, test set encoding, design-for-testability and low power testing. Chapter 5 investigates the impact of using the test sets obtained by all the methods described in Chapter 3 and Chapter 4 in one such possible application, that of deterministic test set embedding. Particularly, focus is given on LFSR-based reseeding schemes for mixed-mode Built-In Self Test (BIST). Such schemes are very popular, since they can easily be used in a large range of designs without modifying critical parts of the design, like the scan-in chain. Even though LFSR reseeding is chosen for demonstrating the applicability of the methods of Chapter 3 and Chapter 4, their impact in other applications is also significant. Two simple variations of LFSR reseeding, (i) multiple polynomial LFSR reseeding [20] and (ii) partial reseeding [21]. Both these embedding schemes can impose certain constraints on the test set to be embedded, e.g. the maximum number of specified bits in a single test pattern. This chapter proposes a generic framework which allows for a systematic exploration of parameters that are important to a specific embedding scheme, in order to derive those flexible test sets that are best for the considered scheme. Hence, it is shown how the techniques of Chapters 3 and 4 can be applied on top of some embedding scheme to further optimize the generated test set. The storage requirements are shown to be reduced when the obtained test sets are used. While this methods have been developed using the stuck-at fault model, any linear fault model (such as the transition fault model used in the techniques of Chapter 2) can be used without affecting the affecting the techniques' complexity. Preliminary results of this work have been published in [18].

Chapters 6 and 7 examine problems related to test sets that explicitly detect each modeled fault multiple times. Such test sets, known as n-detect test sets since they enforce detection of each fault with at least n different test patterns, have been shown to be of increased quality for detecting random defects and/or non-modeled faults. Chapter 6 investigates n-detect test set relaxation. This problem has not been studied before and, thus, is of increased interest.

Current n-detect test generation methods produce a large number of fully specified test patterns, limiting their practical application to large circuits. The problem is formulated as an optimization problem and propose a solution that iterates among faults. This methodology enforces detection of each fault only by those n tests resulting in the overall largest specified bits reduction. Experimental results showed that n-detect test set relaxation rates are very similar and, in some cases, even higher than those of 1-detect test sets. Moreover, experimentation with random bit fixing showed that the n-detect test sets maintain their non-targeted fault and defect coverage, after the relaxation process. As with 1-detect test set relaxation, the results where obtained using the stuck-at fault model, but any linear model can be used. A part of the results has appeared in [22].

Chapter 7 proposes a new test generation methodology for n-detect test sets, which increases their quality in terms of the number of fault propagation paths per generated test. Specifically, the n tests per stuck-at fault are, for the first time, guaranteed to propagate the fault effect via different propagation paths, without any path or path segment enumeration. The proposed method can easily be extended to other linear, to the circuit size, static or dynamic fault models for multiple fault detections, such as the transition fault model for which both different activation and propagation paths can be guaranteed. The generated test sets demonstrated increased numbers of propagation paths and non-modeled fault coverage, when compared to traditional n-detect test sets.

Chapter 8 summarizes the conclusions of this thesis and discusses some future research directions relevant to the presented work.

CHAPTER 2

FUNCTIONS FOR QUALITY TRANSITION FAULT TESTS AND THEIR APPLICATIONS IN TEST SET ENHANCEMENT

2.1 Introduction

The need for testing accurate temporal behavior, commonly known as delay testing [23, 24], is becoming increasingly important due to the demand for high performance in today's digital circuits as well as various process variations, manufacturing defects, and noise factors. The two most popular delay fault models are the *path delay fault* (PDF) model [25, 26, 27] and the *gate delay fault* model [28, 29, 30, 31]. Even though the PDF model is the most accurate one since it can detect both lumped and distributed delay defects, it can become impractical because an exponential, to the circuit size, number of faults is explicitly examined. On the other hand, gate delay fault models examine a linear, to the circuit size, number of faults and, therefore, they are more feasible for larger circuits. For both models, a test consists of a pair of vectors, $< v_1, v_2 >$, where v_1 initializes the target node/path and v_2 launches the appropriate transition and propagates it to an observable point (primary output).

The most commonly used gate delay fault model is the *transition fault* model,which models excessive delay on a single circuit node and associates two delay faults per node: a *slow-to-rise* (rising) fault and a *slow-to-fall* (falling) fault. The generation of a transition fault test involves two steps. First, the *activation* of the target fault by creating a transition on the faulty line through single or multiple paths and, second, the *propagation* of the fault effect to a primary output by single or multiple path sensitization. The underlying assumption of this model is that the extra delay caused by a transition fault is large enough to cause a timing failure at a primary output. Thus, the delay on any sensitizable propagation path from the fault site to a primary output is assumed to exceed the maximum allowable circuit delay. Based on this assumption, traditional transition faults are modelled as stuck-at faults [23], making test generation simpler. Specifically, for a fault site l, vector v_2 is a stuck-at 0 (stuck-at 1) test for a rising (falling) transition fault at l and vector v_1 asserts l to a 0 (1) logic value. However, such transition fault tests, also known as weak non-robust tests for their propagation properties [29, 32], can be invalidated during testing due to signal hazards and other factors [33]. Recent research efforts on transition fault testing, such as [32], have focused on generating high quality tests by examining various sensitization criteria for activating and propagating transition faults.

In this chapter, we present the first ever proposed method to derive a function that contains all possible tests for each transition fault. For a rising (falling) transition fault, we refer to the corresponding function as the *rising (falling) test function*. Test function generation is advantageous for a variety of reasons including test set compaction, non-enumerative critical path sensitization, and test set enrichment. Function–based ATPG methods are inherently scalable and are gaining in popularity over structural methods as the circuit size increases with the advances in deep sub–micron. They are also backed by advances in data structures and methods that store and manipulate functions which include canonical forms, such as the Binary Decision Diagrams (BDDs) [34], and non-canonical forms, such as Boolean satisfiability.

Quality tests for transition faults are generated by considering any previously proposed type of sensitization to either activate a transition at the fault site or propagate it to a circuit output. Events are also restricted to propagate along any path in a subcircuit containing paths that meet predetermined delay criteria using the fixed delay model.

The difficulty in generating a function is primarily associated with the development of a systematic methodology to incorporate the sensitization criteria required for quality tests during either fault activation or fault propagation. This inherently requires structural methods (circuit traversals) during the function formulation. The desired functions are generated by generalizing a recently proposed method in [35] which shows how to use structural methods to generate logic error as well as event propagation functions. For logic errors, the traditional boolean difference functions are generated using structural methods[1]. For events (transitions), [35] shows how to generate a Boolean function that represents all the possible primary input assignments that allow for an event (a rising or falling transition) on some line to be propagated at some primary output.

The work in this chapter generalizes the work in [35] and allows for events to propagate through gates using specific sensitization rules either during the fault activation phase or the fault propagation phase. We distinguish among robust, non-robust or function sensitization criteria [25, 26, 27], but any other sensitization classification rules that have been proposed in the literature can be incorporated in our framework. Subsequently, given a transition fault, it is shown how to generate a function that contains all patterns that allow for its activation, and a function that contains all patterns for its propagation. The product of these two functions forms the final function for the transition fault. Better quality tests are obtained by working on subcircuits whose paths meet criticality criteria as explained earlier.

Another important contribution of this work is that is shown how to reduce the required amount of computational effort for generating all transition fault test functions in a circuit. Specifically, it is shown that (i) all activation functions can be generated using a single forward topological traversal of the original circuit and (ii) all propagation functions can be generated using a backwards topological traversal on a linearly modified circuit. Thus, in contrast to traditional brute-force approaches that would require, in the worst case, circuit traversals per fault, the proposed method, which is based on dynamic programming principles, can derive all transition fault functions based only on two circuit traversals.

This work also shows how the proposed method for the function generation can be beneficial to test set enhancement techniques that can be applied to provide higher quality tests. It is first shown that events can be restricted to either sensitize paths in a subcircuit containing paths that meet predetermined delay criteria using a fixed delay model. Previous attempts for quality tests for transition faults insist that transitions are restricted along predefined paths that pass through the fault cite [32, 36]. Such research efforts essentially propose that the complexity of the PDF model is handled by examining only a subset of the PDFs. Restricting the event activation along a predetermined activation and/or propagation route is, in principle, associated with the PDF model and not the transition fault model whose original

[1]Given a circuit line a and primary output z, the boolean difference function is defined as $\frac{\partial z}{\partial a}$.

definition is path independent. In contrast to any existing approach, the proposed method generates test functions for each transition fault under any activation/propagation sensitization criterion and at the same time allows the event to sensitize any path in a subcircuit whose paths have exactly the same delay. In particular, using a dynamic programming algorithm, we define k subcircuits: the first contains all longest paths, the second all second longest paths, and so on. Then, for each transition fault, k functions are generated. The i^{th} function contains all tests to detect a transition fault where the event is propagated along any of the paths of the i^{th} subcircuit, under a parameterized sensitization criterion. This is a path independent, yet quality test oriented, method that focuses on critical routes and generates tests for a polynomial number of delay faults, which we call quality transition fault tests for the above mentioned reasons.

The second enhancement technique deals with test set compaction. The generated test functions are manipulated to generate compact test sets. The objective is to cover the same transition faults using a small subset of the original set of test functions. This amounts to performing AND operations, in a systematic manner, on the original test functions. Two simple heuristics for compaction are presented and the impact of each one is evaluated in Section 2.5. It is shown that applying both compaction methods to the original test set gives a compaction rate of 70–80% for all ISCAS'89 and ISCAS'85 circuits. Note that the studied compaction problem differs from the one in [37] where the authors also benefit by transition fault coverage by the additional (in-between) pairs of patterns that are formed when the tests generated by the ATPG tool are placed in a test set, one after the other. Future research will investigate reductions by such post-processing steps.

Finally, a novel method to enrich the compacted transition fault test set with additional tests of a certain property is presented. This property is related to the paths sensitized by the test. The rationale in this case is to add a number of new tests in the set such that they still detect the targeted transition faults but through paths that have not been sensitized by the original test set. Such test sets have higher quality since events propagate through many critical paths and, thus, are more likely to detect a delay violation in the circuit. It is expected that, on average, this procedure will allow the number of paths along which the event propagates to increase proportionally to the number of tests per transition fault.

The remainder of this chapter is organized as follows. Section 2.2 presents the formulation of event propagation functions at a single line under various sensitization criteria. Section 2.3

discusses methodologies for generating propagation functions and activation functions for all lines in a circuit. The various enhancement techniques are presented in Section 2.4. Section 2.5 presents experimental results for the ISCAS'85 and ISCAS'89 benchmark circuits, and Section 2.6 concludes Chapter 2.

2.2 Recursive Definition of Propagation Functions

An event is either a rising (R) or a falling (F) transition and is denoted by $tr \in \{R, F\}$. (The terms event and transition are used interchangeably in this context.) The event propagation function at some line l for transition tr is denoted by $P_{l,tr}()$. Set $FI(G)$ denotes the set of all fanin (immediate predecessor) lines of gate G. The controlling (non-controlling) value of a gate g is denoted by $cv(g)$ $(ncv(g)) \in \{0, 1\}$.

The event propagation function at a line l is defined recursively, with respect to the event propagation function at the line's immediate successor and the necessary sensitization criteria required at lines driving the same gate as line l. Line l is called an *on-input* and the lines driving the same gate as line l are called *off-inputs*.

This Section discusses the formulation of such a recursive event propagation function for a single circuit line. Different formulations are given in the respective subsections based on the type of sensitization criteria proposed in [25], which extends those first presented in [26, 27] by determining multiple path sensitization criteria. Table 2.1 shows the constraints (or necessary conditions) on the off-inputs of a gate g which are determined based on the type of transition tr to be propagated from the on-input of g and the type of sensitization under consideration. The *don't care value* is denoted by x. Observe that in the case of multiple path propagation (functional sensitization), the off-inputs can assume any value besides the stable-at controlling value.

Table 2.1: Off-input constraints for event propagation through gate G

Sensitization Condition	*tr* on the on-input(s)	
	cv(G) $\rightarrow$ ***ncv(G)***	***ncv(G)*** $\rightarrow$ ***cv(G)***
Robust	$x \rightarrow ncv(G)$	$stable\ at\ ncv(G)$
Non-Robust	$x \rightarrow ncv(G)$	$x \rightarrow ncv(G)$
Functional	$x \rightarrow ncv(G)$	$not\ stable\ at\ cv(G)$

2.2.1 Non-robust Sensitization

Consider a 2-input AND gate G with inputs $\{a, b\}$ and output c. An event on line a can propagate through gate G to some primary output (let c be an internal line) only if the event can propagate from line c to some primary output. Under the non-robust sensitization criterion, all of the off-input lines of G (line b for this example) must settle at a $ncv(G)$ value, irrespective of the type of the event to be propagated on a. Therefore, line b must settle to value 1 for G. (Similarly, b must settle to 0 is G is an OR gate).

Non-robust sensitization does not impose any constraints for the first vector (v_1). This is indicated by the don't care requirement for v_1 in Table 2.1. Thus, the off-input constraints for v_2 at some line x can be expressed with respect to a single boolean variable, x_2. For every line x we define a boolean variable x_2, which will represent the two logic values $\{1, 0\}$. For value 1 we use x_2 and for value 0 we use $\overline{x_2}$.

The event propagation function at line a for gate G is defined as $P_{a,tr}() = P_{c,tr}() \cdot b_2$. Similarly, if G is an OR gate we have $P_{a,tr}() = P_{c,tr}() \cdot \overline{b_2}$. In general, given an AND gate G with output line g and on-input line l, the non-robust propagation function for event tr on l is given by:

$$P_{l,tr}() \quad = \quad P_{g,tr}() \quad \cdot \prod_{\forall x \in FI(G), x \neq l} (x_2) \tag{2.1}$$

For an OR gate Equation 2.1 becomes:

$$P_{l,tr}() \quad = \quad P_{g,tr}() \quad \cdot \prod_{\forall x \in FI(G), x \neq l} (\overline{x_2}) \tag{2.2}$$

Finally, for a NOT gate an $R(F)$ event on l is propagated as an $F(R)$ event on G giving:

$$P_{l,R}() \quad = \quad P_{g,F}() \tag{2.3}$$

$$P_{l,F}() \quad = \quad P_{g,R}() \tag{2.4}$$

Observe that, in the above formulas, we have not yet defined the set of variables over which the propagation functions $P_{l,tr}()$ and $P_{g,tr}()$ are expressed. This will be presented in detail in Section 2.3.

2.2.2 Robust Sensitization

A major disadvantage when applying non-robust tests is that they may get invalidated due, primarily, to signal hazards [33]. A non-robust test assumes that all of the transitions on the off-inputs of the gates on the propagation paths arrive earlier than the transitions on the corresponding on-inputs. Only then it is guaranteed that a delay will be detected. Robust tests, on the other hand, always guarantee the detection of the delay and, thus, are preferred for high quality testing.

In contrast to non-robust sensitization, robust sensitization imposes more stringent rules on the off-inputs (see Table 2.1). When the event to be propagated settles to a non-controlling value the off-input constraints are the same as those for non-robust sensitization. However, when it settles to a controlling value only stable-at non-controlling values are allowed on the off-inputs, in order to avoid hazard excitation. In this case, both v_1 and v_2 vectors must be explicitly determined.

We define variables s_l^0 and s_l^1 for every line $l \in L$ to represent the stable-at-0 and stable-at-1 values, respectively. (We elaborate on stability values and functions in Subsection 2.3.1.) Assume the same 2-input AND gate example as in the previous subsection. The event propagation function at line a for gate G is defined based on the type of the transition to be propagated. If the transition settles to a non-controlling value, thus $tr = R$, then the robust event propagation function is identical to the non-robust event propagation function, thus $P_{a,R}() = P_{c,R}() \cdot b_2$. If the transition settles to a controlling value, $tr = F$, then the robust event propagation function is defined as $P_{a,F}() = P_{c,F}() \cdot s_b^1$. Similarly, if G is an OR gate then $tr = R$ settles to a controlling value and we have $P_{a,R}() = P_{c,R}() \cdot s_b^0$.

In general, given an AND gate G with output line g and on-input line l, the robust propagation function for event tr that settles to a controlling value on l is given by:

$$P_{l,F}() \quad = \quad P_{g,F}() \quad \cdot \prod_{\forall x \in FI(G), x \neq l} (s_x^1)$$

For an OR gate the above Eq. becomes:

$$P_{l,R}() \quad = \quad P_{g,R}() \quad \cdot \prod_{\forall x \in FI(G), x \neq l} \left(s_x^0\right)$$

Observe that for events that settle to a non-controlling value on l, the robust event propagation functions are identical to those given for non-robust propagation in Equation 2.1 – Equation 2.2. Clearly, Equation 2.3 – Equation 2.4 are the same for all types of sensitization.

2.2.3 Functional Sensitization

The proposed propagation function formulation also handles the case where an event cannot be propagated as a single event, but when propagated together with other events it can affect the delay of the circuit. According to the classification of [25], when no non-robust tests exist (in our context this means that the desired event cannot be propagated to some primary output using a non-robust test) the targeted fault is either untestable or it can be detected together with other faults as a multiple fault. Multiple faults are tested using functional sensitization.

As in the case of robust sensitization, the off-input constraints for event propagation under functional sensitization conditions differ based on the type of event to be propagated (see last row of Table 2.1). When tr settles to a non-controlling value, the off-input constraints are the same as those for non-robust sensitization and, thus, the event propagation functions for functional sensitization are identical to those for non-robust propagation given by Equation 2.1 – Equation 2.2.

When the event to be propagated settles to a controlling value, the off-inputs can take any value besides stable-at controlling since such a value would mask the propagation of the event. As in the case of robust tests, the stability variables can be used to represent the necessary values. Specifically, any value other than stable-at-1(0) on some line l is represented by $\overline{s_l^1}$ ($\overline{s_l^0}$).

Therefore, given an AND gate G with output line g and on-input line l, the functional sensitization propagation function for event tr that settles to a controlling value on l, is given by:

$$P_{l,F}() \quad = \quad P_{g,F}() \quad \cdot \prod_{\forall x \in FI(G), x \neq l} \left(\overline{s_x^0}\right)$$

For an OR gate the above Eq. becomes:

$$P_{l,R}() \;=\; P_{g,R}() \;\cdot \prod_{\forall x \in FI(G), x \neq l} \left(\overline{s_x^1}\right)$$

2.3 Test Function Generation for Transition Faults

This Section discusses the generation of transition fault test functions for all circuit lines. A transition fault test function is the product of fault activation and fault propagation functions. First, a systematic methodology is presented for generating all event (transition fault) propagation functions in the circuit based on a single traversal of a linearly modified circuit. This method is a generalization of the recently proposed method of [35], to allow for the propagation events, based on specific sensitization rules. Moreover, it is shown how considerable speedup over the algorithm of [35] can be achieved. Next, it is shown how to generate all activation functions based on a single traversal of the original circuit.

2.3.1 Propagation Function Generation

Generating Propagation Functions for all Faults

We define $L = \{i_1, i_2, ..., i_n\}$ to be the set of all lines in a circuit, excluding primary outputs, and $I \subset L$ to be the set of all primary inputs. For each line i_k we maintain two boolean variables i_k^1 and i_k^2 to form the set of all variables $V = \{i_1^1, i_1^2, i_2^1, i_2^2, ..., i_n^1, i_n^2\}$. We partition V to $V^1 = \{i_1^1, i_2^1, ..., i_n^1\}$ and $V^2 = \{i_1^2, i_2^2, ..., i_n^2\}$ to be the subsets $i_k^1 \in V$, and $i_k^2 \in V$, respectively. Additionally, we define $V_I^1 \subset V^1$ and $V_I^2 \subset V^2$ as the sets of all variables in V^1 and V^2, respectively, corresponding to only primary inputs.

Definition 2.1. The *functions* of a line $l \in L$, denoted by $F_l^1(V_I^1)$ and $F_l^2(V_I^2)$, are the functions realized at line l expressed with respect to the input variables in V_I^1 and V_I^2, respectively.

Given a logic gate G with output line l, $FI(G)$ stands for the set of all inputs (fanins) of G. The variable sets $VG^1 = \{i_k^1 \mid i_k \in FI(G)\}$ and $VG^2 = \{i_k^2 \mid i_k \in FI(G)\}$ consist of the variables corresponding to the fanins of G.

Definition 2.2. The *local functions*, $LF_l^1(VG^1)$ and $LF_l^2(VG^2)$, of output line $l \in L$ of gate G, are the functions realized at line l, expressed with respect to variables in VG^1 and VG^2, respectively.

Definition 2.3. The *stability functions* of line $l \in L$, denoted by $S_l^0(V_I^1 \cup V_I^2)$ and $S_l^1(V_I^1 \cup V_I^2)$, contain all vectors $< v_1, v_2 >$ that bring a stable-at-0 and stable-at-1 value at line l, respectively.

Stability functions were first introduced in [38] for single-input change tests, and later generalized in [39] for multi-input change tests. We use the latter method to derive multi-input global stability functions for all circuit lines. According to [39], the $S_l^0()$ and $S_l^1()$ functions are defined recursively with respect to the appropriate stability functions of the immediate predecessor lines of l, based on the type of gate that drives l. If $l \in I$, $S_l^0() = \overline{l_1} \cdot \overline{l_2}$ and $S_l^1() = l_1 \cdot l_2$. (Primary inputs are assumed to always be hazard-free.)

For example, for a 2-input AND gate with inputs $\{a, b\}$ and output c, $S_c^1() = S_a^1() \cdot S_b^1()$ and $S_c^0() = S_a^0() + S_b^0()$. If c is a primary input then, $S_c^1() = c_1 \cdot c_2$ and $S_c^0() = \overline{c_1} \cdot \overline{c_2}$. Similarly, if G is an OR gate then, $S_c^0() = S_a^0() \cdot S_b^0()$ and $S_c^1() = S_a^1() + S_b^1()$ when c is an internal line, and $S_c^0() = \overline{c_1} \cdot \overline{c_2}$ and $S_c^1() = c_1 \cdot c_2$ when c is a primary input.

We also define the set of boolean variables $SV = \{s_{i_1}^1, s_{i_1}^0, s_{i_2}^1, s_{i_2}^0, \ldots, s_{i_n}^1, s_{i_n}^0\}$ such that for every line $i_k \in L$ two variables, $s_{i_k}^0$ and $s_{i_k}^1$, are maintained. The variables in SV will be used to express the local stability requirements. Assume gate G with output line l and fanins $FI(G)$. The variable set $SVG = \{s_{i_k}^1, s_{i_k}^0 | i_k \in FI(G)\}$ contains the stability variables for the fanins of G.

Definition 2.4. The *local stability functions* of output line $l \in L$ of gate G, denoted by $LS_l^1(SVG)$ and $LS_l^0(SVG)$, are the stability functions realized at line l expressed with respect to the variables in SVG.

For the same example of the 2-input AND gate, $LS_c^1 = s_a^1 \cdot s_b^1$ and $LS_c^0 = s_a^0 + s_b^0$.

The propagation functions for all circuit lines are generated using a single topological traversal of a modified circuit, starting at the primary outputs, as in [35] where they were initially proposed. The algorithm identifies topological levels, and iterates over these levels. [35] shows that the size of the modified circuit remains linear to the size of the original one, since a line may appear in at most d topological levels, where d is the circuit depth. For clarity purposes, we present in this subsection the algorithm for generating all event propagation functions based on the topological level requirement (i.e. a line may be processed up to d times) of [35]. However, we show at the end of this subsection that this requirement can be relaxed such that each line is processed *only once*, at the expense of some additional local functions ($LF^1()$ and $LF^2()$) generation.

We also define $L_k \subset L$ to be the set of lines in a topological level k. Thus the topological level corresponding to the set of primary input lines is denoted by L_0 and the one corresponding to the set of primary output lines by L_d. Now, let set $V_k \subset V = \{i_j^1, i_j^2 \mid i_j \in L_k\}$ and set $SV_k \subset SV = \{s_{i_j}^1, s_{i_j}^0 \mid i_j \in L_k\}$ be sets of variables corresponding to the lines in level L_k.

The propagation function for each line, in every level, is expressed in terms of the necessary value assignments on all other lines in that level. This form of expressing propagation functions using topological levels is referred to as *local propagation function (LPF).*

Definition 2.5. The *local propagation function* of a line $l \in L_k$ for event tr, denoted by $LPF_{l,tr}^k(V_k \cup SV_k)$, is the function that represents all necessary conditions at the lines contained in level k that propagate event tr on line l to some primary output.

After generating the $LPFs$ for all lines in a level k, they are re-expressed with respect to the variables in level $k-1$, so that they can be used to compute the $LPFs$ for the lines in level $k-1$. Operator $Sub\_l()$, which substitutes every variable in an LPF with its corresponding local function (LF) or local stability function (LS) is used to accomplish this. $Sup_l^k \subseteq (V_k \cup SV_k)$ denotes the set of variables in the support of $LPF_{l,tr}^k()$. Operator $Sub\_l()$ is defined to perform variable substitution corresponding to lines of successive topological levels as follows:

$$Sub\_l(\ LPF_{l,tr}^k()\ ) = LPF_{l,tr}^k(\ i_j^1 \leftarrow LF_{i_j}^1(), i_j^2 \leftarrow LF_{i_j}^2(), s_{i_j}^1 \leftarrow LS_{i_j}^1(), s_{i_j}^0 \leftarrow LS_{i_j}^0(),$$

$$\mid i_j^1, i_j^2, s_{i_j}^1, s_{i_j}^0 \in Sup_l^k \setminus (V_{k-1} \cup SV_{k-1})\ )$$

Note that operator $Sub\_l()$ does not affect variables that are contained in both levels k and $k-1$.

The propagation functions given by Equation 2.1 – Equation 2.4 in Subsection 2.2.1 (for non-robust sensitization) can now be expressed completely in terms of necessary operations. For a gate G with output line $g \in L_k$, input line $l \in L_{k-1}$, and fanin lines $FI(G) \in L_{k-1}$, the local propagation function for line l for non-robust sensitization, denoted by $LPF^{k-1}_{l,tr}()$, for AND, OR, and NOT gates are given below.

$$AND: \quad LPF^{k-1}_{l,tr}() = Sub\_l(\ LPF^{k}_{g,tr}()\ ) \cdot \prod_{\forall i_j \in FI(G), i_j \neq l} (i^1_j)$$

$$OR: \quad LPF^{k-1}_{l,tr}() = Sub\_l(\ LPF^{k}_{g,tr}()\ ) \cdot \prod_{\forall i_j \in FI(G), i_j \neq l} (\overline{i^2_j})$$

$$NOT: \quad LPF^{k-1}_{l,R}() = Sub\_l(\ LPF^{k}_{g,F}()\ )$$

$$NOT: \quad LPF^{k-1}_{l,F}() = Sub\_l(\ LPF^{k}_{g,R}()\ )$$

Similar equations can be derived for the remaining types of gates as well as for the robust and functional sensitization propagation functions of all expressions given in Subsection 2.2.2 and Subsection 2.2.3.

We now give the definition of a propagation function of a stem. Assume a stem s at level $k-1$ with two branches b_1 and b_2 at level k. Let *P* be the boolean cube that represents the event to be propagated at s, namely $P = s_1 \cdot \overline{s_2}$ for the F event and $P = \overline{s_1} \cdot s_2$ for the R event. The LPF for stem s, is given by:

$$LPF^{k-1}_{s,tr}() = Sub\_l(\ LPF^{k}_{b_1,tr}(s^1_{b_1} \leftarrow s_1 \cdot s_2, s^0_{b_1} \leftarrow \overline{s_1} \cdot \overline{s_2}) + LPF^{k}_{b_2,tr}(s^1_{b_2} \leftarrow s_1 \cdot s_2, s^0_{b_2} \leftarrow \overline{s_1} \cdot \overline{s_2})\ )_{|P}$$

With this formulation, propagation of the targeted event through any one of stem's branches is allowed. Moreover, it is ensured that the stem s and all of its branches can only assume the value of the propagating event by cofactoring the LPF with respect to ***P***. Paths passing through a stem and then reconverging at some other circuit cite may bring different values on the stem's branches. If no local functions and variables were considered, and thus, propagation functions were expressed only with respect to variables corresponding to primary inputs, conflicts between stems and branches could not be identified. Also, stability variables are associated with the local line variables, in order to implicitly identify all conflicting assignments and exclude them from further consideration.

Expressing the LPF in terms of the circuit inputs gives the final form of a propagation function, referred to as *propagation function (PF)*.

Definition 2.6. The *propagation function* of a line $l \in L_k$ for event tr, denoted by $PF_{l,tr}(V_I)$, is the function that contains the complete set of primary input patterns that propagate event tr on line l, to some primary output.

To derive the PF of a line, a variable substitution operation is performed on the line LPF accordingly. Such Operator $Sub()$, is defined below:

$$PF_{l,tr}() = Sub(\ LPF^k_{l,tr}()\ ) = LPF^k_{l,tr}(i^1_j \leftarrow F^1_{i_j}(), i^2_j \leftarrow F^2_{i_j}(), s^1_{i_j} \leftarrow S^1_{i_j}(), s^0_{i_j} \leftarrow S^0_{i_j}(),$$
$$|\ i^1_j, i^2_j, s^1_{i_j}, s^0_{i_j} \in Sup^k_l)$$

PFs are the final form of a propagation function, expressed in terms of the circuit inputs. Observe that they are not used in computing the propagation functions of lines in subsequent levels.

Figure 2.1 describes the process of generating all propagation functions in a pseudocode form. Lines 1 through 6 refer to global and local function generation. Then the topological levels are derived and the algorithm iterates per such level to generate the LPF per line in a level, based on the input sensitization criterion S. The PF of a line is only computed once (as shown in lines 12-13).

```
procedure event_propagation_functions()
INPUT:   circuit $C$,
         sensitization criterion $S$
OUTPUT:  Propagation Function $PF$ per circuit line
% Propagation Function Generation
 1: Generate set of lines $L = \{1, 2, ..., n\}$, ordered topologically.
 2: for each line $l \in L,\ l \notin POs$
 3:     Declare Boolean variables $l^1$, $l^2$, $s_l^1$, and $s_l^0$
 4: for each line $l \in L$
 5:     Generate functions $F_l^1()$, $F_l^2()$, $S_l^1()$, and $S_l^0()$
 6:     Generate local functions $LF_l^1()$, $LF_l^2()$, $LS_l^1()$, and $LS_l^0()$
 7: $levels$ = Derive_topological_levels($C, d$),   % $d$ = depth %
 8: for $k = d$ down to $0$
 9:     for each $tr \in \{R, F\}$
10:         for each line $l \in level[k]$
11:             $LPF_{l,tr}^k()$ = generate_$LPF$(S)
12:             if $l \notin level[k-1]$ then
13:                 $PF_{l,tr}^k()$ = generate_$PF()$
```

Figure 2.1: Pseudocode for Propagation Function generation.

Speeding up the Generation of All Propagation Functions

The topological level based processing requirement of the algorithm presented in the previous subsection, may result in generating the local propagation function (LPF) of a line at most d times, where d is the topological depth of the circuit. In practice, [35] showed that for the ISCAS'85 and ISCAS'89 circuits, the number of local propagation function generations per line is a very small constant, between 2.1 and 12.9 (while d for these circuits is between 13 and 218). Here, we show that it is only necessary to generate the local propagation function per line once, at the expense of some additional local function generations.

A topological level is defined as a set of circuit lines that has the following three properties:

(a) Every I/O path contains a line in the level.

(b) No two lines in the level are on the same I/O path.

(c) A line belongs in a level only after all of its successor lines have been processed.

As stated before a line can be present in at most d levels. That is because of property (c) above. As an example consider a line s at topological level i that has 2 immediate successors (i.e. s is a stem with b_1 and b_2 branches), in levels $i+1$ and $i+j$, respectively (this implies that branch b_2 also belongs in levels $i+1 \ldots i+j-1$). The $LPF_s^i()$ cannot be generated unless both $LPF_{b_1}^{i+1}()$ and $LPF_{b_2}^{i+1}()$ are realized. The $LPF()$ for b_2 is generated at level $i+j$ for the first time ($LPF_{b_2}^{i+j}()$) and thus it must be re-expressed with variables in all levels $i+j-1$ downto $i+1$, before $LPF_{b_2}^{i+1}()$ can be generated (in the worst case, $LPF_{b_1}()$ and $LPF_{b_2}()$ may have to be re-expressed up to $d-2$ times, where d is the circuit depth).

Theorem 2.1. *The local propagation function for a circuit line l, $LPF_{l,tr}()$, needs to be generated (expressed with respect to local variables only in a topological level) only the first time it appears in a topological level, during the course of the algorithm.*

Proof. We first give a definition that generalizes the local functionality and local stability function definitions given so far (Definition 2.2 and Definition 2.4). Let $LF_l^{1,k}()$ and $LS_l^{1,k}()$ ($LF_l^{2,k}()$ and $LS_l^{2,k}()$) be the local function and local stability functions for line l the first (second) vector, expressed with respect to variables corresponding to lines in topological level k. Observe that for the algorithm of Subsection 2.3.1, only $LF_l^1()$ and $LS_l^1()$ ($LF_l^2()$ and $LS_l^2()$) are defined, since local functionalities and stabilities are expressed only with respect to immediate predecessors, and not also with predecessors in some arbitrary level k. Recall that levels $k=0$ and $k=d$ are the levels containing all primary inputs and all primary output lines, respectively, and that the algorithm starts at the primary outputs and terminates at the primary inputs.

Let level $i+j$ be the first topological level the algorithm visits line l and let $i+1$ be the last one to do so. Then for line l we maintain $i+1$ $LF_l^{1,k}()$ and $LS_l^{1,k}()$ ($LF_l^{2,k}()$ and $LS_l^{2,k}()$) functions with $k=0,1,2,...,i$, according to the definition given in the previous section. Clearly $LF_l^{1,0}() = F_l^1()$ ($LF_l^{2,0}() = F_l^2()$) (Definition 2.1) and $LS_l^{1,0}() = S_l^1()$ ($LS_l^{2,0}() = S_l^2()$) (Definition 2.3), which are used by the $Sub()$ operation to realize the final form of propagation function $PF_l()$. Similarly, $LF_l^{1,i}() = LF_l^1()$ ($LF_l^{2,i}() = LF_l^2()$) (Definition 2.2) and $LS_l^{1,i}() = LS_l^1()$ ($LS_l^{2,i}() = LS_l^2()$ (Definition 2.4).

We redefine operator Sub_l $(LPF_{l,tr}^{i+1}())$ to Sub_l' $(LPF_{l,tr}^{i+j}(), i)$ to apply the substitution operation on the local propagation function of line l at level $i+j$ with respect to variables in level i, by substituting every variable x in level $i+j$ with their local functionalities and local stability functions for level i ($LF_x^{1,i}(),\ LF_x^{2,i}(),\ LS_x^{1,i}(),\ LS_x^{2,i}()$).

Thus,

$Sub\_l'(LPF^{i+j}_{l,tr}(), i) = Sub\_l(LPF^{i+1}_{l,tr}()) = LPF^{i}_{l,tr}$ and $Sub\_l'(LPF^{i+j}_{l,tr}(), 0) = Sub(LPF^{i+1}_{l,tr}()) = PF_{l,tr}$. Therefore, it is only necessary to generate the LPF for line l once, that of $LPF^{i+j}_{l,tr}()$. $\square$

Calculating the local propagation functions only once, can reduce the computation time up to a factor of d, where d is the number of topological levels of the circuit. This reduction is made in the cost of maintaining all the $LF^{1,k}_{l}()$, $LF^{2,k}_{l}()$, $LS^{1,k}_{l}()$, and $LS^{2,k}_{l}()$, for all $k = 0, \ldots, i$ which however is a one time cost, obtained by pre-processing of the circuit.

2.3.2 Generating Activation Functions for all Faults

Generating a test for a transition fault involves two steps: fault *activation* and fault *propagation*. So far, we have described a methodology that generates all fault propagation functions in a circuit. Here, we describe the generation of all fault activation functions. The ultimate goal is to derive the transition fault test function $T_{l,tr}() = A_{l,tr}() \cdot PF_{l,tr}()$, where $PF_{l,tr}()$ is the propagation function as defined in the previous subsection (Definition 2.6) and $A_{l,tr}()$ is the activation function for transition tr at line l. The number of minterms in $T_{l,tr}()$ is the *total number of tests* for transition fault tr at l.

If no specific sensitization conditions are enforced from the primary inputs to the fault site l, then the following test functions can be used for a falling and rising transition fault at l, respectively:

$$T_{l,F}() = F^{1}_{l}() \cdot \overline{F^{2}_{l}()} \cdot PF_{l,F}()$$
$$T_{l,R}() = \overline{F^{1}_{l}()} \cdot F^{2}_{l}() \cdot PF_{l,R}()$$

With the above equations, it is ensured that the appropriate transition is launched at l and propagated (robustly, non-robustly, or functionally, depending on how $PF_{l,tr}()$ was generated) to some primary output. Observe that the above formulations do not allow for non-robust activation at l based on static-hazard excitation (i.e. either a $0 \to 1 \to 0$ or a $1 \to 0 \to 1$ at l) since only ”real” transitions (not hazards) can be launched at l.

For higher quality tests, one should also consider the various sensitization conditions of Table 2.1 for fault activation so that the generated tests will guarantee to sensitize *complete*

Table 2.2: Fault activation functions

Non-Robust Sensitization

AND	$A_{g,tr}() = \sum_{\forall i \in FI(G)} [A_{i,tr}() \cdot \prod_{\forall j \in FI(G), j \neq i} F_j^2()]$
OR	$A_{g,tr}() = \sum_{\forall i \in FI(G)} [A_{i,tr}() \cdot \prod_{\forall j \in FI(G), j \neq i} \overline{F_j^2()}]$

Robust Sensitization

AND	$A_{g,F}() = \sum_{\forall i \in FI(G)} [A_{i,F}() \cdot \prod_{\forall j \in FI(G), j \neq i} S_j^1()]$
OR	$A_{g,R}() = \sum_{\forall i \in FI(G)} [A_{i,R}() \cdot \prod_{\forall j \in FI(G), j \neq i} S_j^0()]$

Functional Sensitization

AND	$A_{g,F}() = \sum_{\forall i \in FI(G)} [A_{i,F}() \cdot \prod_{\forall j \in FI(G), j \neq i} \overline{S_j^0()}]$
OR	$A_{g,R}() = \sum_{\forall i \in FI(G)} [A_{i,R}() \cdot \prod_{\forall j \in FI(G), j \neq i} \overline{S_j^1()}]$

paths from the primary inputs to the primary outputs, through the fault site, in a robust, non-robust, or functional manner. We briefly describe how to generate such activation functions below.

Consider again the example of the 2-input AND gate with inputs $\{a, b\}$ and output c, and the case of non-robust fault activation. If $\{a, b\}$ are primary inputs then $A_{a,R}() = \overline{a_1} \cdot a_2$, $A_{a,F}() = a_1 \cdot \overline{a_2}$, $A_{b,R}() = \overline{b_1} \cdot b_2$, and $A_{b,F}() = b_1 \cdot \overline{b_2}$. Consequently, $A_{c,tr}() = A_{a,tr}() \cdot b_2 + A_{b,tr}() \cdot a_2$. Thus, the activation function at the output of a gate can be expressed with respect to the activation functions and the off-input constraints of its immediate predecessors. If $\{a, b\}$ are internal circuit lines then $A_{c,tr}() = A_{a,tr}() \cdot F_b^2() + A_{b,tr}() \cdot F_a^2()$, where $F_i^2()$ is the function of line i for vector v_2 (Definition 2.1).

In general, for a gate G with output line g and fanins $FI(G)$, the transition fault activation functions for $G \in \{AND, OR\}$, under the various sensitization conditions of Table 2.1, are shown in Table 2.2. Similar equations can be derived for the remaining types of gates. The activation function of a branch is identical to that of its corresponding stem. Note that for robust and functional sensitization, only the functions for transition tr settling to a controlling value are given, since the functions for tr settling to non-controlling value are identical to those of non-robust sensitization (see Table 2.1).

Observe that only variables for the primary inputs need to be used in this formulation. Therefore, only functions $F^1()$ and $F^2()$ and stability functions $S^1()$ and $S^0()$ per line, are necessary to derive the activation functions. No local variables (variables per internal line) or local functions are used. A single topological traversal, for the primary inputs to the primary outputs, on the original circuit suffices in generating all fault activation functions.

It is noted that with the proposed technique it is possible, as in [32], to combine different activation and propagation sensitization conditions.

2.4 Applications of Test Functions in Test Set Enhancement

In this section we show how test functions can be beneficial by presenting various test set enhancement techniques that can be applied to produce better quality test sets. Specifically, three such enhancement methods that benefit greatly from the use of test functions are discussed.

2.4.1 Testing Faults Through Paths of Specific Length

The method of Section 2.3 can be applied to restrict event activation/propagation through a subcircuit that contains paths that meet predetermined delay criteria under the fixed gate delay model. In this manner, specific sensitization criteria as well as path criticality can be considered for high quality test generation.

We define a subcircuit C_i of original circuit C which consists of all paths in C of length i. Let d be the size of the longest topological path in C. Then C_d contains only the paths of C of maximum length, namely the longest paths. In this manner C_{d-1} contains only second longest paths, C_{d-2} contains only third longest paths, and so on. We use a dynamic programming algorithm to generate the transition fault test functions for k such subcircuits, where k is a constant between 1 and d, defined by the desired path criticality criteria. Using the method of Section 2.3, k different test functions are generated for each fault, each of which restricts event propagation/activation through all paths of the same length i, namely subcircuit C_i. Depending on the desired qualities of the test set, the k test functions can either be used separately to derive a test for each fault in every C_i, or be combined (Boolean OR) to derive a test for all C_i subcircuits, $i = d, d-1, ..., d-k$, that allows fault activation/propagation through paths of any length between d and $d-k$. This circuit decomposition can be run in parallel, if necessary, so as to minimize the computation time.

Next we focus on how to derive a C_i subcircuit. We use a simple algorithm, based on circuit traversals and linear number of node replications, which guarantees that the generation of C_i is done in linear, to the size of circuit C, time. First, a forward topological traversal is performed in C. At each line l in the circuit, a list of buckets is maintained and each bucket B is labelled with the *length*(L) of sub-paths ending at that line. Each such bucket B_L holds a list of identifiers for all *immediate* predecessor of l that reach l with sub-paths of length L. As an example, consider the graph of Figure 2.2. Graph nodes represent circuit lines and $L[p_1, p_2, ...]$ for node l denotes that $p_1, p_2, ...$ are immediate predecessors of l that reach l with sub-paths of length L (bucket B_L and corresponding identifiers kept). For instance, node e is labelled with $3[d], 2[d], 1[i_3]$ which implies that it has 3 buckets, $\{B_3, B_2, B_1\}$, one for every sub-path length up to that node. Node d is the only identifier in the list for buckets B_3 and B_2 of node e, since sub-paths of length 3 and 2 reach e from d. In the same manner, the primary input i_3 is the only identifier contained in the list for bucket B_1 of line e, since i_3 is the only immediate predecessor of e that reaches e with length 1. In general, the number of buckets kept per line is no more than d, where d is the length of the longest path in the circuit.

A backward traversal follows the forward traversal, starting from primary outputs that have a bucket labelled with the desired path length, let this be i. When at some bucket B_L, of a node l, the list of identifiers kept for B_L is used to determine the nodes to be visited next. For example, consider again the graph of Figure 2.2 and suppose that we are interested in deriving C_5. For simplicity, let's focus only on paths of length 5 ending at primary output k, namely i_2adfhk, i_2bdfhk and i_1defhk. Then, for B_5 at k we know that h is to be visited next. Moreover, bucket B_4 of h is selected, since in order to reach k with paths of length 5, it is necessary to reach h with paths of length 4. In this manner, the paths of the desired length are identified when the primary inputs are reached, based on the visited nodes and edges. During this process, it is essential to ensure that no paths of any length other than i are included in

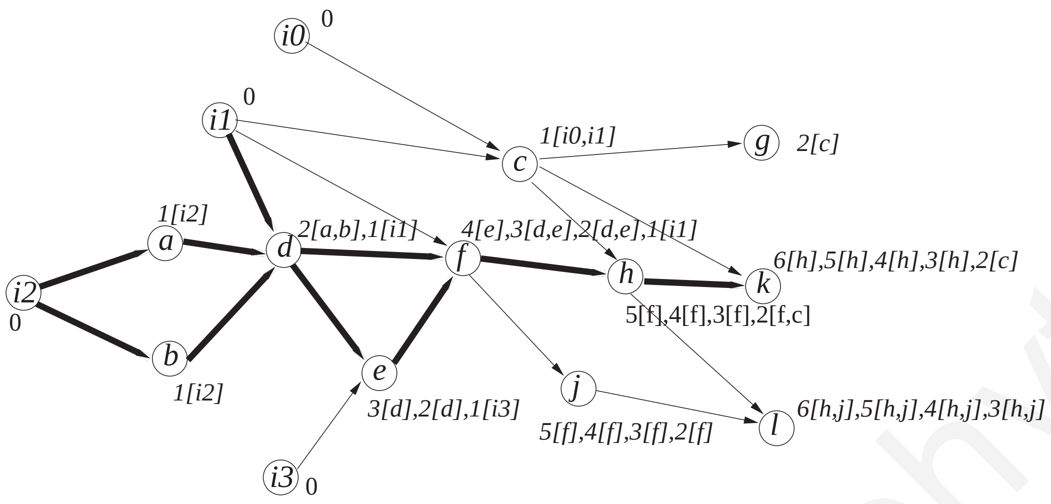


Figure 2.2: Attempting to derive C_5 on original Graph C.

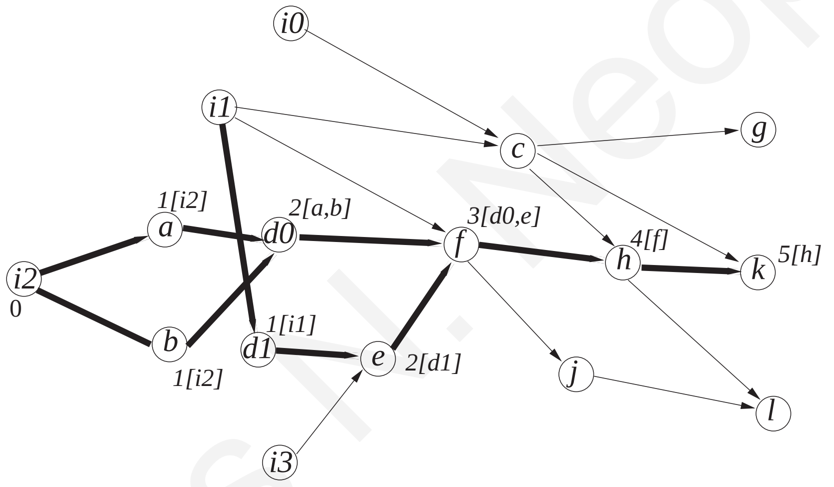


Figure 2.3: Replication of node d to derive C_5.

C_i. Observe again Figure 2.2. If the process of backtracking from B_5 of k is applied till all primary inputs are reached, then the identified paths are shown in bold lines in Figure 2.2. In this case, C_5 (bold part of the Figure 2.2) also contains paths of length 4 (i_1dfhk) and length 6 ($i_2adefhk$ and $i_2bdefhk$). In order to avoid such cases, our algorithm modifies the induced subcircuit by replicating nodes to ensure the properties of C_i. Visited nodes (other than PIs) that have sub-paths to POs, of different size are replicated with connections to appropriate successors and predecessors. In the graph of Figure 2.3 (only used buckets per node are shown), node d is replicated in C_5 (consider again only paths of length 5 to PO k) since it is reached from f via B_3 and from e via B_2. Node d_0 reaches nodes a and b via B_2 and node

d_1 reaches i_1 via its B_1 bucket. Bold lines in Figure 2.3 show all visited nodes and edges. In general, if d is the length of the longest path in circuits C then a node may be replicated, in the worst case, as many as $d - i + 1$ times in order to derive subcircuit C_i. Clearly, the circuit modification is linear to the size of the original circuit C.

Note that a subcircuit C_i does not contain lines which are not on a path of length i and are needed by the method of Section 2.3 to ensure off input constraints presence along targeted paths. We call these lines *side lines* of C_i. Formulation of test functions is trivially modified in a way such that the side input constraints are met, but no fault sensitization through subpaths consisting of side lines are allowed.

2.4.2 Test Set Compaction

The second test set enhancement method proposed examines the problem of decreasing the size of the test set without any compromise in fault coverage, known as test set compaction. By insisting on maintaining test functions per fault we have a major advantage over structural-based techniques, that find only one test per fault (or small set of tests if don't-cares are allowed), when trying to compact a test set. Each test function can implicitly represent an enormous number of tests per fault. This gives a much greater flexibility in finding tests that detect many faults.

Consider fault f with transition fault test function $T_f()$. Function $T_f()$ contains all possible tests for *f*, under the specified sensitization criterion. Any minterm of $T_f()$, let m_f, can be selected to cover fault f. Consider now a second fault *f'*, other than *f*, with corresponding test function $T_{f'}()$. One can again derive a minterm, let $m_{f'}$, from $T_{f'}()$ to cover fault f'. Alternatively, one may attempt to find a common test for the two faults, i.e., a single test that will detect both faults simultaneously. This amounts to computing the intersection of the two sets of minterms represented by each test function. When dealing with boolean functions, set intersection is equivalent to Boolean AND. Thus, $T_f() \cdot T_{f'}()$ will contain all possible tests that can detect faults f and f' simultaneously. If $T_f() \cdot T_{f'}() = 0$ then faults f and f' are said to be incompatible, which means that they can never be detected together by a single test. In general, given a list of n faults, $\{f_1, f_2, \ldots, f_n\}$ and their corresponding list of test functions, $\{T_{f_1}(), T_{f_2}(), \ldots, T_{f_n}()\}$, the function $T_N()$ that contains all tests that detected all n faults simultaneously is given by $T_N() = \prod_{i=1\ldots n} T_{f_i}()$.

The goal here is to select test functions that can be combined, i.e. have at least one common minterm. Ideally, one would like $T_N()$ to have as many minterms as possible, since this may also indicate that further reduction of the test set is possible. In general, the main objective for achieving good test set compaction using functions relies on finding a combination of the faults in the targeted fault list such that each of generated $T_N()$ combines a large number of functions such that the number of $T_N()$ functions necessary to detect all faults is minimized.

We use two simple heuristics to demonstrate the effectiveness of test function generation in test set compaction. The first method starts with a test set that contains one test function per fault and checks for local compatibilities, i.e. attempts to compact faults in the same locality. Using a backward circuit traversal we check if the test function of a fault f at some line l can be combined with the appropriate test functions for faults at one or more of the immediate predecessor lines of l. For instance, consider a 2-input NAND gate with inputs a and b and output g. There are 6 transition faults for lines $\{a, b, g\}$ (3 rising and 3 falling faults) with 6 corresponding test functions, denoted by $\{T_a^R(), T_a^F(), T_b^R(), T_b^F(), T_g^R(), T_g^F()\}$. The compaction algorithm will attempt first to compute functions $T_1() = T_a^R() \cdot T_b^R() \cdot T_g^F()$ and $T_2() = T_a^F() \cdot T_b^F() \cdot T_g^R()$. Function $T_1()$ ($T_2()$) attempts to combine all the rising (falling) faults on the inputs with the falling (rising) fault at the output of the gate. If any of these two functions is 0, the algorithm will select subsets of the inputs (instead of all) and try to see if their corresponding faults can be tested together. Once a test function is considered in one compacted set of functions, it is dropped from further consideration. We repeatedly apply this scheme at each line in the circuit. At fanout stems the rising (falling) test function for a fault at the stem is combined with all rising (falling) test functions for the faults at its branches. If a single function that detects all does not exist (=0), then the algorithm again selects subsets of the branches to be combined. This scheme is very effective, as it guarantees to reduce the test set by at least 50% (except in the case where the stem fault is redundant), since the test function of a gate's output (fanout stem) can always be successfully combined with the corresponding test function of at least one of the gate's inputs (fanout branches). We call this heuristic compaction method A.

A second compaction heuristic, we call it method B, is also applied on top of method A. It considers pair-wise compaction of all remaining functions in the test set. This greedy approach can be applied in an iterative manner until the size of the test set cannot be further reduced. We start with the set of test functions obtained by compaction method A (one can also apply method B independently by considering one test function per fault in the input test set) and perform pair-wise compaction. Once a test function is considered in one compacted pair of functions, it is dropped from further consideration. Any function $T_f()$ that cannot be combined with any other function is also added in the new list of test functions to guarantee that fault coverage is not compromised. The above process is repeated on the new list of test functions until no further compaction is possible (size of starting list is equal to the size of the new list). Although, this approach selects functions to combine greedily, our experimental results for the ISCAS'85 and ISCAS'89 circuits show that it can achieve up to 80% of test set compaction of the original test set (when applied on top of compaction method A).

The pseudocode for the compaction heuristic is shown in Figure 2.4. Method A is given in lines 1–13 and method B in lines 14–26. Fault f_i^{tr} denotes transition fault at line i with $tr = \{R, F\}$. Operator ! denotes the change of transition $R(F)$ to transition $F(R)$. For example, if $tr = R$ then $!tr = F$. Operators $\cdot$ and $+$ denote Boolean AND and OR, respectively.

2.4.3 Test Set Enrichment

Here we discuss an additional application of high quality transition fault test functions for test set enhancing. We present a novel method to enrich the compacted test set with more tests by adding a number of new tests in the set such that they detect the same transition faults but only through paths that have not been sensitized by the initial test set. Such test sets have higher quality since events propagate through many different paths and, thus, are more likely to detect a delay violation in the circuit. It is expected that, on the average, this procedure will allow the number of paths sensitized by all transition fault tests to increase proportionally to the number of tests per transition fault.

Any minterm of a transition fault test function guarantees to detect transition fault by activating and propagating the transition through at least one circuit path. Using the method described in Subsection 2.4.2 the test set is compacted as a first step. If the test set size drops below some required upper bound (maximum test set size), one can select to enrich the test set with new tests that guarantee detection of already detected faults, but via sensitization of new (not already sensitized) paths, thus, improving the quality of the test set. (Note that the method described in this section can be also applied independently to the compaction method of Subsection 2.4.2.)

```
procedure compact()
INPUT:  circuit $C$,
        test function array $TA$,
        corresponding fault array $FA$
OUTPUT: compacted test function array $new\_TA$
% Compaction method $A$
 1: for each fault $f_i^{tr}$ in FA
 2:     $G$ = gate driving line $i$
 3:     if $G$ is inverting gate then $tr' = !tr$ else $tr' = tr$
 4:     $FI(G)$ = fanin list of gate $G$ with output line $i$
 5:     $comp\_TA = \emptyset$
 6:     $temp\_T_{comp}^{tr}() = T_{f_i}^{tr}(),\ \ T_{comp}^{tr}() = logic\_one(1)$
 7:     for each fanin $j \in FI(G)$, where $f_j^{tr} \in FA$
 8:         $temp\_T_{comp}^{tr}() = T_{F_j}^{tr'}() \cdot T_{comp}^{tr}()$
 9:         if $temp\_T_{comp}^{tr}() = logic\_zero(0)$ then break
10:         else $T_{comp}^{tr}() = temp\_T_{comp}^{tr}()$
11:             Drop fault $f_j^{tr'}$ from $FA$
12:     Add $T_{comp}^{tr}()$ to $comp\_TA$
13: for each fault $f_j^{tr} \in FA$, Add $T_j^{tr}()$ to $comp\_TA$
% Compaction method $B$
14: $k = 0$, $new\_TA = temp\_TA = \emptyset$
15: do
16:     if $k = 0$ then $temp\_TA = comp\_TA$ else $temp\_TA = new\_TA$
17:     Increment $k$
18:     for each $T_{f_i}^{tr}() \in temp\_TA$
19:         for each $T_{f_j}^{tr}() \in temp\_TA$, $T_{f_j}^{tr}() \neq T_{f_i}^{tr}()$
20:             $T_{comp}^{tr}() = T_{f_i}^{tr}() \cdot T_{f_j}^{tr}()$
21:             if $T_{comp}^{tr}() \neq logic\_zero(0)$ then
22:                 Add $T_{comp}^{tr}()$ to $new\_TA$
23:                 Drop functions $T_{f_i}^{tr}()$ and $T_{f_j}^{tr}()$ from $temp\_TA$
24:                 break
25:     for each $T_{f_i}^{tr}() \in temp\_TA$, Add $T_{f_i}^{tr}()$ to $new\_TA$
26: while ($|new\_TA| < |temp\_TA|$)
```

Figure 2.4: Pseudocode for Compaction Algorithm

The enrichment algorithm starts with a compacted (or not) list of test functions. For every function $T_i()$ in the list, a minterm m_{T_i} (test) is randomly selected and simulated to identify all sensitized paths, under the given sensitization criterion that the function was originally generated. Paths through which the targeted fault(s) is(are) sensitized, as well as other paths of the circuit detected incidentally, are identified. Then, for each identified sensitized path, the path test function (again, under the given sensitization criterion) is generated. Path test

```
procedure enrich()
INPUT:   test function array $TA$, Test_Set_Bound
OUTPUT: enhanced test set $S$
 1: $S = \emptyset$
 2: for each test function $T_i() \in TA$
 3:     $m_{T_i}$ = pick_minterm($T_i()$)
 4:     Add test $m_{T_i}$ to $S$
 5: while ($|S| \leq$ Test_Set_Bound)
 6:     Pick test $m_{T_i} \in S$
 7:     Simulate $m_{T_i}$
 8:     $\mathcal{P}$ = set of paths sensitized by $m_{T_i}$
 9:     $T_P = logic\_zero(0)$
10:     for each path $p_i \in \mathcal{P}$
11:         $T_{p_i}()$ = test function for path $p_i$
12:         $T_P() = T_P() + T_{p_i}()$
13:     $T_i' = T_i() \cdot \overline{T_P()}$
14:     $m_{T_i}'$ = pick_minterm($T_i'()$)
15:     Add test $m_{T_i}'$ to $S$
```

Figure 2.5: Pseudocode for Enrichment Algorithm

functions, denoted here by $T_p()$ for some path p, are discussed and generated in [38] and [40], among others. Let $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$ be the set of n paths sensitized by test m_{T_i} and $\mathcal{T}_{\mathcal{P}}() = \{T_{p_1}(), T_{p_2}(), \ldots, T_{p_n}()\}$ be the set of their corresponding path test functions. The set of all possible tests that can sensitize any path in $\mathcal{P}$, denoted by $T_P()$ is given by $T_P() = \sum_{i=1\ldots n} T_{p_i}()$, where $\sum$ denotes the Boolean OR operation. To find the set of all tests in $T_i()$ that sensitize paths not in $\mathcal{P}$, it suffices to compute function $T_i'() = T_i() \cdot \overline{T_P()}$. We can then enrich our test set with any minterm $m_{T_i'}$ of $T_i'()$, which guarantees to detect fault i through paths not in $\mathcal{P}$.

The diagram in Figure 2.6 illustrates the above process in a Venn diagram. It assumes that test function T_i covers 2 faults. It is shown how the set of minterms (tests) of function $T_i'() = T_i() \cdot \overline{(T_{p_j}() + T_{p_k}())}$ is defined. U is the universe of possible patterns. The test function $T_i()$, for fault i, is the intersection of activation and propagation functions, $A_i()$ and $P_i()$, respectively. Any minterm m_{T_i}, selected from $T_i()$, sensitizes paths whose corresponding test functions can have some or complete overlap with function $T_i()$. These two cases are illustrated in Figure 2.6. Let m_{T_i} sensitize two paths, p_j and p_k. Their corresponding test functions, denoted by $T_{p_j}()$ and $T_{p_k}()$, are shown in Figure 2.6. Observe that $T_{p_j}() \subseteq T_i()$, which implies that path p_j contains the line of transition fault i. On the other hand, path p_k

may be an incidental path sensitized by m_{T_i}. In any case, $T_i'()$ (shown in the patterned set in Figure 2.6) contains those minterms of $T_i()$ that will detect the fault i through any path other than p_j or p_k.

The number (or $\%$) of new tests added to the original test set can be user defined. The pseudocode of the proposed enrichment algorithm is shown in Figure 2.5. *Test_Set_Bound* denotes the maximum number of tests allowed in the enriched test set. Operators $\cdot$ and $+$ denote Boolean AND and OR, respectively.

2.5 Experimental Results

The proposed methods were implemented in C language and run on a 1GHz SunBlade 1500 with 4GB of RAM. All Boolean functions were represented and manipulated using Binary Decision Diagrams (BDDs)[34]. The package of [41] was used to generate the BDDs as well as optimal initial ordering of variables, that is also provided along with [41]. No dynamic variable reordering was allowed, even though it could have been invoked to provide space saving at the expense of computation time. All circuits of the ISCAS'85 and ISCAS'89 benchmarks, with the exception of c6288, were considered. Circuit c6288 is a 16 x 16 multiplier that cannot be represented using BDDs. We note here that no comparison between

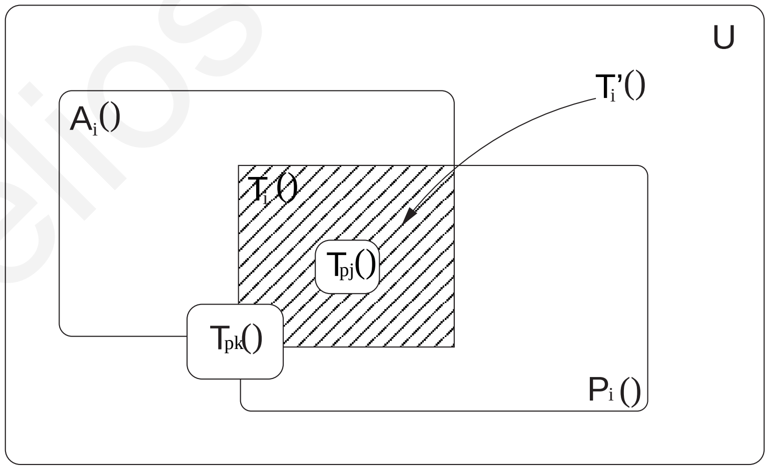


Figure 2.6: Finding new test function $T_i'()$.

our results and other methods is possible since, to our knowledge, no other existing work considers such types of quality tests for transition faults.

Table 2.3: Test Function generation for the ISCAS'85 and ISCAS'89 circuits.

Circuit	Subcircuit C_{max}			Subcircuit C_{max-1}			Subcircuit C_{max-2}		
	Paths	Faults	Red.	Paths	Faults	Red.	Paths	Faults	Red.
s382	24	82	0	4	44	0	18	88	0
s420.1	9	64	0	4	86	0	13	156	0
s444	32	122	29	16	90	0	8	102	19
s641	2	200	113	2	198	113	4	386	38
s713	32	232	184	32	230	183	76	442	186
s1423	4	204	8	6	224	10	4	216	2
s9234.1	5632	1668	1475	512	704	652	10240	1858	1677
s13207.1	240	322	235	840	830	197	3184	1340	1252
s15850.1	1024	442	396	24292	1254	0	91432	2060	1042
s38417	64	238	192	112	322	259	96	498	397
s38584.1	896	598	524	128	338	293	7147	1176	1028
c880	108	196	0	270	244	0	334	388	0
c1355	196608	1588	894	65536	1508	801	298560	2664	1599
c1908	32	360	180	618	888	180	3328	1498	0
c2670	448	346	334	1248	528	516	5568	916	904
c3540	32	202	163	460	678	291	5014	1318	30
c5315	12	328	164	816	692	386	7388	1208	601
c7552	1	124	64	136	492	138	1743	980	204

We first present the results obtained for generating the transition fault test functions for non-robust propagation, without enforcing any specific sensitization rule for fault activation. As already mentioned in Subsection 2.3.2, the computational effort to generate all fault activation functions for various sensitization criteria is considerably less than the one required to generate all fault propagation functions, since in the former case no local variables or functions are involved. Table 2.3 shows the results when faults are activated and propagated along any path in subcircuits C_{max}, C_{max-1}, and C_{max-2}, where max denotes the size of the longest circuit path. For each subcircuit, the number of paths (Columns 2, 5, and 8),

Table 2.4: Time and space requirements for test function generation for the ISCAS'85 and ISCAS'89 circuits.

Circuit	Subcircuit C_{max}		Subcircuit C_{max-1}		Subcircuit C_{max-2}	
	CPU (secs)	Mem. (MBs)	CPU (secs)	Mem. (MBs)	CPU (secs)	Mem. (MBs)
s382	0.04	1.94	0.01	1.94	0.04	2.05
s420.1	0.05	7.45	0.71	16.06	1.01	17.85
s444	0.04	2.02	0.02	2.04	0.0	2.05
s641	0.07	3.28	0.06	3.33	0.07	3.49
s713	0.06	3.33	0.04	3.35	0.07	3.35
s1423	0.75	22.85	1.11	22.50	10.91	30.36
s9234.1	12.75	172.7	0.26	81.99	19.91	213.74
s13207.1	0.38	30.83	0.54	31.45	0.41	31.45
s15850.1	0.26	34.02	0.19	34.02	0.09	34.02
s38417	24.27	4.0	31.13	5.01	38.11	4.97
s38584.1	37.15	57.99	25.18	55.98	41.51	63.99
c880	7.78	28.51	9.11	29.36	9.99	35.42
c1355	110.01	213.72	55.73	83.99	174.11	228.98
c1908	74.90	188.47	71.29	252.02	79.18	312.02
c2670	0.14	11.47	0.90	13.42	5.15	121.67
c3540	60.89	43.61	110.86	52.77	90.32	50.75
c5315	93.11	30.71	80.13	51.17	85.13	65.37
c7552	8.45	31.85	7.08	95.32	13.44	115.30

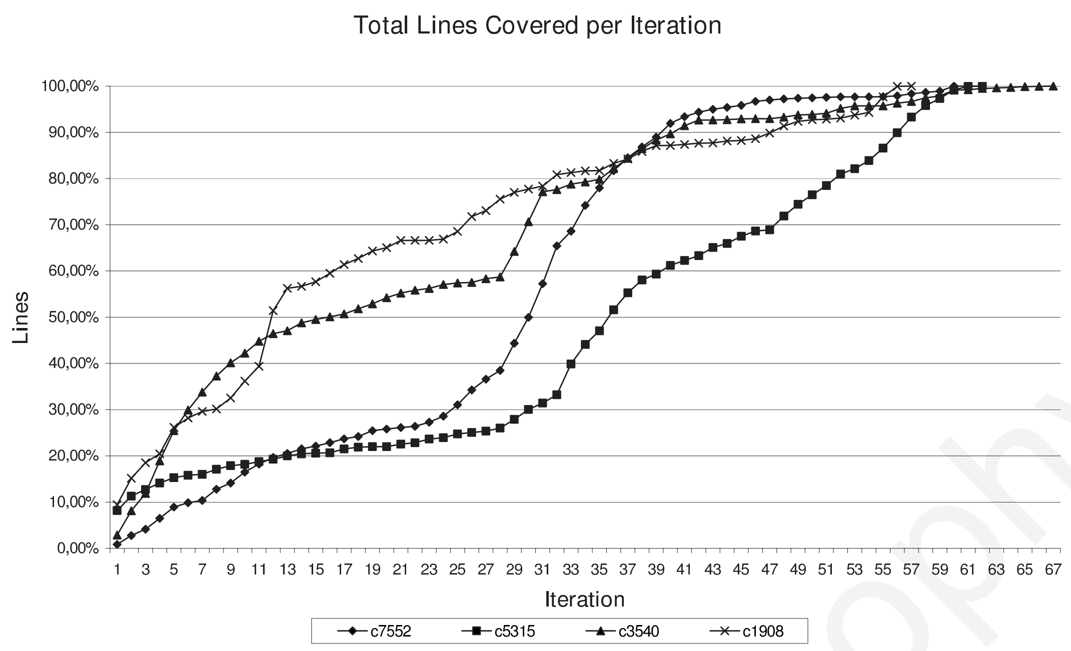
the number of faults (Columns 3, 6, and 9), and the number of identified redundant faults (Columns 4, 7, 10) are reported. Time and space requirements are given in a similar manner in Table 2.4. All faults were processed, i.e., there were no aborted faults in any of the three subcircuits. An important observation from the obtained results is that a very large percentage of the faults, for many subcircuits, are redundant (cannot be activated or propagated to some primary output under the considered sensitization criterion). A fault is identified as redundant when its test function is the constant zero function. It is expected that when specific sensitization criteria are enforced also for fault activation, the number of redundant faults will increase further. Note that the number of faults considered and the number of redundant faults are reported per subcircuit in Table 2.3. To determine the total number of redundant faults after the three iterations of the approach (for subcircuits C_{max}, C_{max-1}, and C_{max-2}), one should compute the sum (logic OR) of the test function for the same faults that appear in any subcircuit. This information is not provided here, however, it could be trivially obtained.

Next we concentrate on presenting some static information per circuit, with regard to subcircuits C_i. One might want to know how many iterations of test function generation are necessary (one iteration finds C_i containing only paths of length i and applies the method of Section 2.3 to find the transition fault test functions), given some user defined threshold for transition fault coverage as well as % of considered paths of high quality (high length).

Figure 2.7 and Figure 2.8 show information for some of the ISCAS'85 and some of the ISCAS'89 circuits, respectively. Each figure contains two plots. The bottom plot in each figure shows the percentage of circuit lines (analogous to the percentage of transition faults covered with high quality tests) considered, as the number of iterations increases. The maximum number of iterations per circuit is equal to the size of the longest path in the circuit (under the fixed gate delay model). Again, let max be the size of the longest path in a circuit. Thus, for some iteration i this plot shows the percentage of transition faults covered through paths of at least length $max - i + 1$. For example, for iteration 1 this plot will show what percentage of transition faults can be covered through only longest paths (iteration 1 will consider subcircuit C_{max}). Observe that the fault coverage reported here is pessimistic (it can be higher) since the total number of the redundant faults is not considered in the calculation.

The top plot in each of the Figure 2.7 and Figure 2.8 shows the percentage of high quality paths (in terms of length) considered for transition fault activation/propagation as the number of iterations increases. For example, for iteration 1 this plot will give the percentage of the total paths that are the longest. For iteration i these plots show the percentage of paths of length $max - i + 1$ or greater from the total number of paths in the circuit. Up to iteration i, the method considers subcircuits $C_{max}, C_{max-1}, \ldots, C_{max-i+1}$ and the reported percentage of paths shows the ratio of the sum of paths among all considered subcircuits over the total number of paths in the circuit.

Consider, for example, circuit c3540 in the bottom plot of Figure 2.7. With approximately 21 iterations the targeted transition faults (around 55% of total number of transition faults as given in the top plot of Figure 2.7) will be detected though only 40% of the paths of



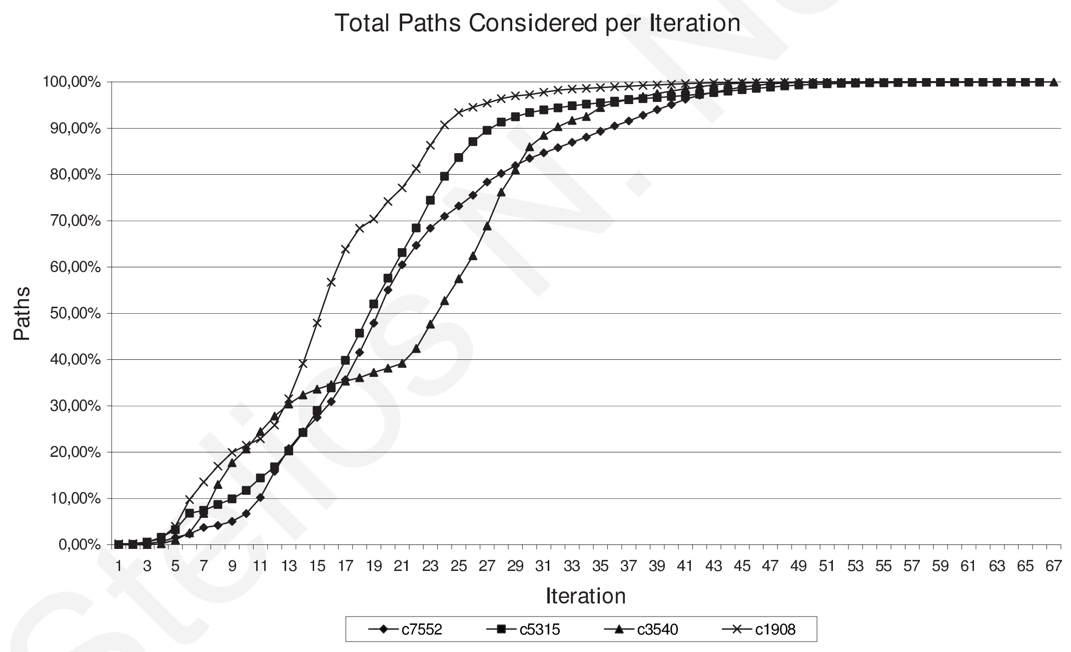
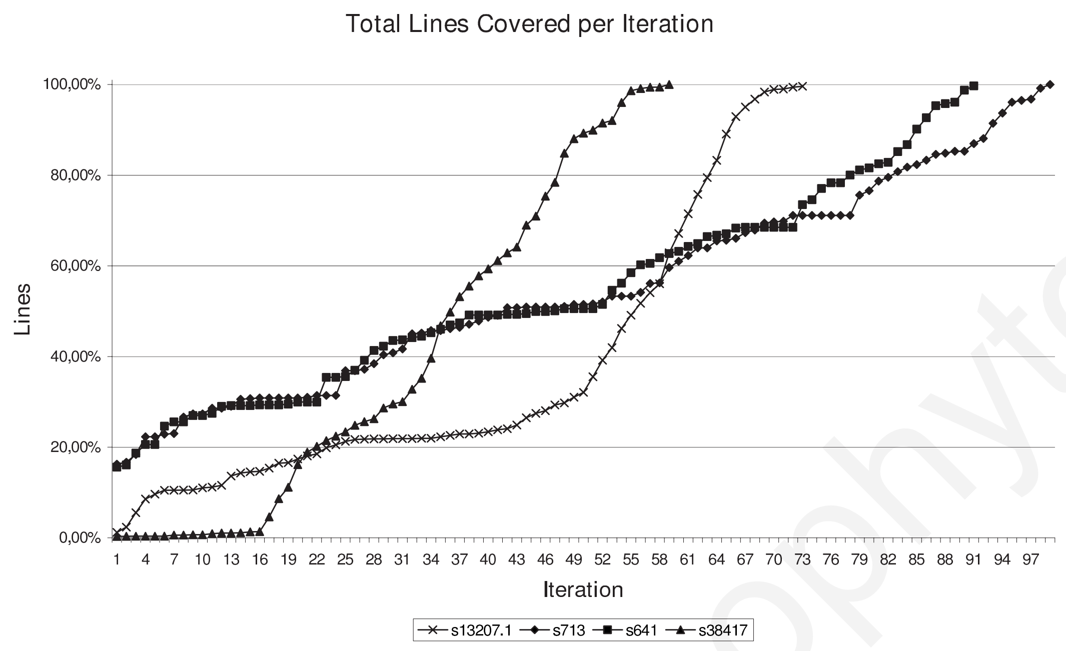


Figure 2.7: Subcircuit C_i details for some ISCAS'85 circuits.

the circuit. Each path in this 40% has length $max - 20$ or more. Thus, around 55% of the transition faults can be detected through paths of size $max - 20$ or more. As another



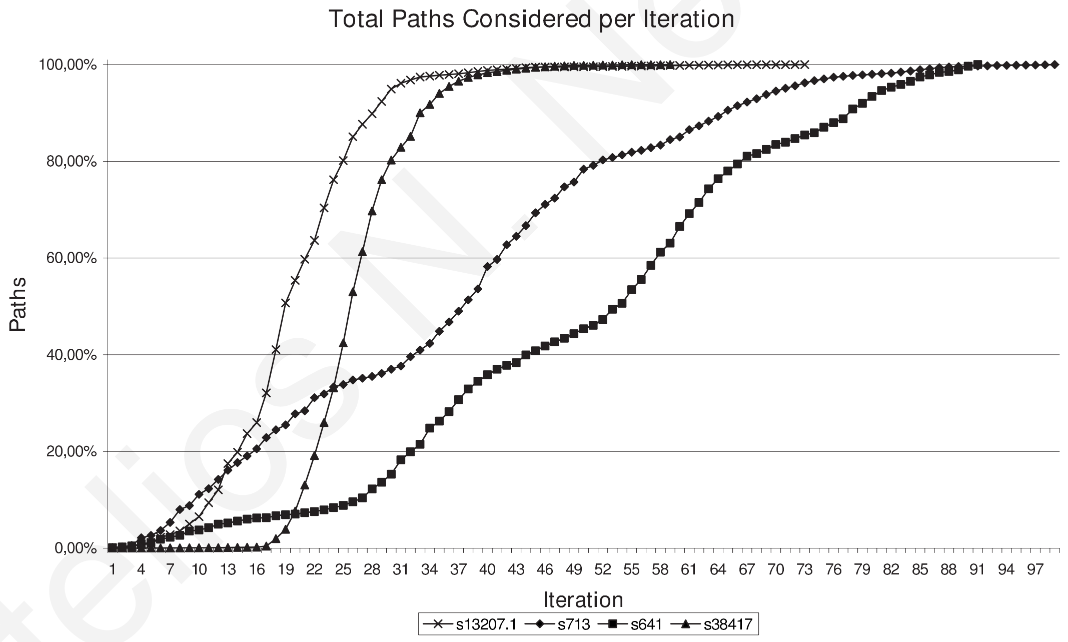


Figure 2.8: Subcircuit C_i details for some ISCAS'89 circuits.

example, consider circuit s641 shown in Figure 2.8. It shows that with around 30 iterations, approximately 45% of the transition faults can be detected through 20% of the total paths, which have length $max - 30$. Such information is very useful in guiding the overall ATPG process.

Table 2.5: Test Set Compaction for ISCAS'85 and ISCAS'89 circuits

Circuit	Subcircuit C_{max}				Subcircuit C_{max-1}				Subcircuit C_{max-2}			
	No. Tests	Comp. A(%)	Comp. B(%)	Comp. Tot.(%)	No. Tests	Comp. A(%)	Comp. B(%)	Comp. Tot.(%)	No, Tests	Comp. A(%)	Comp. B(%)	Comp. Tot.(%)
s382	82	56	50	78	44	52	50	76	88	58	50	78
s420.1	64	55	47	77	86	48	50	74	156	61	50	81
s444	93	57	48	78	90	49	44	72	83	62	47	80
s526n	28	52	50	76	58	59	50	80	83	60	50	80
s641	87	56	50	78	85	53	50	77	348	55	47	76
s713	48	56	50	78	47	57	50	79	256	57	50	79
s1423	196	57	41	75	214	59	50	80	214	52	49	76
s9234.1	193	59	50	80	52	41	50	71	181	53	48	76
s13207.1	87	56	45	75	633	48	48	73	88	57	50	79
s15850.1	46	56	46	76	1254	51	47	74	1018	59	47	78
s38417	46	55	41	73	63	53	50	77	101	46	50	79
s38584.1	74	56	45	76	45	53	45	74	148	59	48	79
c880	196	55	48	77	244	51	50	76	388	59	48	79
c1355	694	55	48	77	707	46	46	71	1065	60	47	79
c1908	180	68	50	84	708	61	47	79	1498	57	50	79
c2670	12	58	50	79	12	53	50	77	12	53	50	77
c3540	39	61	47	79	387	43	50	72	1288	56	47	77
c5315	164	54	40	72	386	52	50	76	607	55	49	77
c7552	60	56	50	78	354	54	48	76	776	56	50	70
Avg. Comp. %		**56.74**	**47.16**	**77.16**		**51.74**	**48.68**	**75.47**		**56.58**	**48.79**	**78.27**

Next, we report compaction results, obtained by applying the compaction method described in Subsection 2.4.2. Table 2.5 shows the compaction results for the test sets presented in Table 2.3. Thus, Columns 2, 6, and 10 show the size of the original test set (equal to the number of test functions generated) for subcircuits C_{max}, C_{max-1}, and C_{max-2}, respectively. Columns 3, 7, and 11 show the reduction rate (%) achieved for Method A of Subsection 2.4.2, and Columns 4, 8, and 12 show the reduction rate achieved for Method B of Subsection 2.4.2. The total reduction rate when Method B is applied on top of Method A is shown in Columns 5, 9, and 13. The proposed compaction technique is very effective since the total reduction rates are between 70% and 84% for all circuits. The last row of Table 2.5 reports the average reduction rates for all listed circuits. The total time performance (CPU secs), shown in Table 2.6, demonstrate that the proposed scheme is very efficient.

Table 2.6: Time requirements for Test Set Compaction (CPU secs)

Circuit	Subcircuit C_{max}	Subcircuit C_{max-1}	Subcircuit C_{max-2}
s382	0.01	0.01	0.01
s420.1	0.01	0.01	0.02
s444	0.01	0.01	0.01
s526n	0.01	0.02	0.02
s641	0.02	0.02	0.08
s713	0.02	0.02	0.03
s1423	0.03	0.02	0.03
s9234.1	0.23	0.01	0.02
s13207.1	0.03	0.04	0.02
s15850.1	0.01	8.17	5.11
s38417	0.02	0.02	0.03
s38584.1	0.02	0.01	0.04
c880	0.04	0.02	0.05
c1355	0.08	0.11	0.19
c1908	27.54	39.11	51.99
c2670	0.01	0.01	0.02
c3540	0.05	0.19	1.53
c5315	0.02	0.05	0.06
c7552	0.01	0.02	0.04

2.6 Conclusions

This chapter presented a novel methodology for efficient generation of transition fault test functions for high quality tests. First we showed how to generate a function that contains all possible tests to detect a transition fault. Moreover, a systematic methodology is presented, that derives the functions for all transition faults based on only two circuit traversals. Quality tests are generated by requiring that the function formulation considers established sensitization criteria to either activate a transition at the fault site, or propagate it to a circuit output. Experimental results on the ISCAS'85 and ISCAS'89 circuits demonstrate the promise of the method and that the examined test functions can be generated quickly and with reasonable memory requirements. In addition, they show that, in many cases, only a small percentage of the faults, for the examined problem, can have high quality tests. Moreover, it is shown how the proposed method can be beneficial for further test set enhancement techniques which can be applied to provide better quality test sets. One such method restricts events propagation along any path in a subcircuit containing paths that meet predetermined delay criteria using the fixed delay model. Another, manipulates the derived test functions to generate compact test sets. Experimental results show a compaction rate of the order of 70% to 84%, with no compromise in fault coverage. Finally, a novel method to enrich the compacted test set with additional vectors so that transition faults are tested through different activation and propagation paths. Such test sets have higher quality, compared to traditional transition fault test sets, since events can propagate through many critical paths.

The test generation methodology proposed in this chapter can be easily extended to produce test sets with don't care bits (incompletely specified test sets), on top of the quality criteria specified in this work. Such "flexible" test patterns may be desirable for many reasons. Appropriate don't care bit fixing can be used to benefit different applications such as low power testing, test set embedding, test set enrichment and test set compaction, among many others. The subsequent chapters explicitly examine the problem of generating flexible test sets. Even though the stuck-at fault model is used in the experiments, any linear fault model, such as the enhanced transition delay fault model presented in this chapter, can be considered.

CHAPTER 3

STATIC TEST SET RELAXATION

3.1 Introduction

A number of digital circuit testing problems can benefit when using flexible test sets, i.e., test sets with a large number of unspecified bits, as their starting point. In this chapter we approach quality from a different direction from that of Chapter 2. Instead of generating test sets of increased detection capabilities, we generate "flexible" test sets which under appropriate manipulation benefit different test-related applications and problems. Such applications include, but are not limited to, adding to a given test set extra properties such as fixing unspecified bits appropriately for low power dissipation during test [11, 42] or additional fault type detection [43]. Flexible test sets are also extremely crucial in various compression schemes for on-chip or off-chip test set embedding, given in [44, 45, 46] among many others, as well as in test set compaction as it will be demonstrated in this dissertation. The majority of existing test generation tools produce fully specified test sets since fixing the values of test bits is essential for the test set compaction methods they employ. Most of the popular compaction techniques rely on having fully specified bits in order to take advantage of coincidental detection of faults and removal of tests that do not target any new faults. Examples of such compaction methods include the static technique of reverse order fault simulation as well as the popular dynamic method of [47] which identifies and merges compatible tests. Test generation techniques that allow the existence of don't cares (unspecified bits) in the generated test set usually result in larger test sets. Moreover, one drawback that is common to all ATPG or dynamic compaction methods is that they cannot take advantage of random test generation when don't care values need to be considered.

Therefore, it is often necessary to relax an already generated test set so that it contains many unspecified bits. The work in this chapter examines this problem, which is defined as *replacing a fully or partially specified test set with a new partially specified test set such that the total number of unspecified bits is maximized while fault coverage remains the same and test set size does not increase*. In this context, test set relaxation does not imply that the specified bits of the new test set are a subset of the specified bits in the original test set, as it is the case with the existing test relaxation methods in [48, 49]. These methods rely on various ATPG concepts in order to identify specified bits in the test set that can be replaced by don't care values. [48] proposes a method for identifying don't care bits in a test pattern using ATPG concepts such as implication and justification. [49] uses a similar rationale, taking into consideration testability measures in the justification process. Moreover, this latter method proposed some heuristic to improve the accuracy of the relaxation process.

The methods proposed here are essentially test replacement techniques, which ensure that each new test pattern has fewer care bits than the one being replaced, and detects at least a subset of the faults detected by the original test. In order to maintain fault coverage, each fault is guaranteed to be detected at least once. Detecting a fault additional times is not essential and can be eliminated in favour of decreasing the specified bits. This is actually a major observation that is implicitly enforced by all previous methods that solve this problem [48, 49]. Here, this observation is explicitly explored in the proposed techniques. Without any loss of generality, the work in this chapter concentrates on single time stuck-at fault detection, as in [48, 49]. However, the presented methodologies apply to any other linear, to the size of the circuit, fault model, such as the transition fault model considered in Chapter 2.

The first method investigated proceeds in a test-oriented manner. It replaces each test with one that has a larger number of unspecified bits and targets only those faults detected by the original test but are not explicitly or coincidentally detected by already relaxed tests. This is essentially a simple test dropping process, coupled with test generation for maximizing unspecified bits, which in practice performs very well (as it is demonstrated by the obtained experimental results). The second method presented is more sophisticated and proceeds towards a different direction than the first one by considering one fault at a time. For each fault, this method determines the one test to detect the fault (among all of the original tests that detected the fault) that gives the maximum benefit in terms of specified bits savings in the entire test set. Thus, it selects the "best" test to detect the fault and drops the fault from the remaining tests in order to reduce the total number of specified bits in these tests.

Both of the proposed methods are different from the previously proposed techniques since they use test set replacement instead of relaxing each test pattern. The latter can limit the number of specified bits that can be relaxed. On the other hand, test set replacement implies explicitly invoking a test generator, capable of generating tests with large number of unspecified bits. However, this is not very different from what is required by the existing methods [48, 49], which rely on modified ATPG routines (such as justification) or have to deal with standard ATPG related problems (such as fault masking due to multiple path activation).

All test generation related routines can be implemented with any previously proposed method that generates tests with many unspecified bits. Essentially, structural approaches like those used in [48, 49] can be employed. Alternatively, symbolic techniques in a function-based framework (e.g. BDD based, SAT based) can be used for all the proposed techniques, as well as for the techniques we propose in the next chapter. Since the impact of the test generation process is important for all the proposed techniques, we propose a function-based approach based on Binary Decision Diagrams (BDDs) [34]. Beyond test generation, this approach allows for efficient representation and manipulation of tests or group of tests. Moreover, we take advantage of the canonical form of BDDs in order to generate tests with a large number of specified bits. We postpone this discussion until the next chapter (Section 4.5) in order to give a complete framework that can be used with all the proposed techniques of this as well as the next chapter.

The rest of this chapter is organized as follows. Section 3.2 gives the problem formulation together with some necessary definitions. Section 3.3 and Section 3.4 describe in detail the two proposed methods. In Section 3.5 we present a post-processing compaction heuristic for reducing the number of tests in the relaxed test set. Section 3.6 gives the obtained experimental results and necessary comparisons for the proposed methods. Section 3.7 concludes Chapter 3.

3.2 Problem Formulation and Notation

Consider a given test set $\mathcal{T}$=$\{t_1, t_2, ..., t_N\}$ for a combinational or a fully-scanned sequential circuit-under-test $\mathcal{C}$. Each of the N test patterns consists of a string of 3-valued bits $\in$ {0,1,x}, thus, the test set $\mathcal{T}$ can be fully or partially specified. Consider also a fault model $\mathcal{M}$, based on which the list of faults detected by $\mathcal{T}$, denoted by F, is derived. For a test set $\mathcal{T}$, we denote the ratio of the bits having a specified value $\{$0,1$\}$ over the total number of test set bits by $K(\mathcal{T})$. This ratio gives a test set property that indicates how flexible a test set is. Clearly, $0 \leq K(\mathcal{T}) \leq 1$, for any test set. The closer $K(\mathcal{T})$ is to 0, the more flexible $\mathcal{T}$ is. For fully specified test sets, $K(\mathcal{T}) = 1$.

The test set relaxation process refers to replacing test set $\mathcal{T}$=$\{t_1, t_2, ..., t_N\}$ with a test set $\mathcal{T}'$=$\{t'_1, t'_2, ..., t'_N\}$ such that each of the following is satisfied:

- $\mathcal{T}'$ has the same fault coverage as $\mathcal{T}$ (under fault model $\mathcal{M}$)
- $K(\mathcal{T}') < K(\mathcal{T})$, i.e. $\mathcal{T}'$ has more unspecified bits than $\mathcal{T}$
- Every $t'_i \in \mathcal{T}'$ detects a subset of the faults detected by some $t_j \in \mathcal{T}$

Since there is no constraint on the place of specified bits that become unspecified at a test, the replacement method may give tests that have no specified bits at all. Depending on the targeted application, such tests can be either preserved or eliminated.

3.3 Test-Based Replacement Method

The optimal approach for solving the problem of minimizing the number of specified bits in a test set is to identify a minimal input assignment to test each fault. In other words, the target is to find the test that detects each fault with the minimum number of specified bits. Since, for this problem, we focus on modifying a given test set to have as few specified bits as possible, the optimal solution is to identify the pattern in the test set that contributes less in specified bits and then convert the specified bits to don't care in all other tests that detect the fault under examination. The former step, however, can become problematic because it involves test generation, not only for each one of the examined faults but for all the combinations of faults, as well. Hence, since the number of combinations can be exponential, the methods proposed in this subsection (as well as in the next subsection) try to identify the optimal test (the one that gives the fewer specified bits), in an iterative manner. Of course the results are not optimum; however, experimentation shows that they are better than other similar methods.

The first method proposed in this subsection iterates among tests examining the faults that are targeted by the pattern considered at each iteration. Thus, the order by which the faults are examined is defined by the order of the test patterns in the test set. All faults that have been detected by a previous test are removed form the detect list of each test and a new test is formed that has fewer specified bits than the given one. Thus, the new test, that replaces a test in the initial test set $\mathcal{T}$, detects the faults detected by the initial test excluding any faults already detected. The proposed algorithm is given in Figure 3.1. The input parameters are the circuit-under-test $\mathcal{C}$, the test set to be relaxed $\mathcal{T}$, and the considered fault model $\mathcal{M}$ based on which the targeted fault list F is derived (lines 1-2 of Figure 3.1). Consequently, for each given test $t_i \in \mathcal{T}$ the algorithm first finds the faults detected by t_i and not already detected by tests $t'_1, t'_2, ..., t'_{i-1}$ (this corresponds to lines 5-6 in Figure 3.1), and then generates a test replacement t' that only targets these faults. If all the faults detected by t_i are already detected by at least one of $t'_1, t'_2, ..., t'_{i-1}$ then t_i is a fully unspecified test (i.e. $K(t') = 0$ which is the maximum relaxation possible for t_i) and is dropped from test set $\mathcal{T}'$. The process terminates when all tests in $\mathcal{T}$ have been processed or when 100% fault coverage is achieved for $\mathcal{T}'$ (line 14 of Figure 3.1). In this manner, the relaxed test set $\mathcal{T}'$ is guaranteed to have no more tests than the initial test set $\mathcal{T}$.

```
procedure **Test_Based_Replacement**

Inputs:   circuit $\mathcal{C}$, test set $\mathcal{T}$, fault model $\mathcal{M}$
Outputs: relaxed test set $\mathcal{T}'$
01: Fault Simulate $\mathcal{T}$ based on fault model $\mathcal{M}$
02: $F$ = list of faults detected by $\mathcal{T}$
03: $\mathcal{T}' = \emptyset$, $F' = \emptyset$
04: for each $t \in \mathcal{T}$
05:       $F_i$ = list of faults detected by $t_i$
06:       if $F_i - F' \neq \emptyset$
07:             generate test $t'$ that detects all faults in $F_i - F'$
08:             if $t'$ has more unspecified bits than $t_i$
09:                   $t'_i = t'$
10:             else
11:                   $t'_i = t_i$
12:             add $t'_i$ in $\mathcal{T}'$
13:             $F' = F' + F_i$
14:       if $F' = F$ break
15: return $\mathcal{T}'$
```

Figure 3.1: Static Test-based Replacement Method.

All static test set relaxation methods rely, implicitly or explicitly (as in the proposed methods), on removing multiple detections of faults in the given test set. Let $sp(t_i)$ denote the number of specified bits in test t_i. The effectiveness of the proposed static method, relies on the following:

Theorem 3.1. *If a test t_i detects a number of faults $F_i = \{f_j, j = 1, 2..., n | t_i$ detects $f_j\}$, then $\forall\ F'_i \subset F_i\ \exists\ t'_i$ that detects all faults in F'_i and such that $sp(t'_i) \leq sp(t_i)$.*

Proof. Let T_j denote the set of all tests that detect fault f_j. Then the set of tests that detect all faults in F_i is given by $T_{F_i} = \bigcap T_j, f_j \in F_i$. Clearly, test $t_i \in T_{F_i}$. Then, for $F'_i \subset F_i$ and $T_{F'_i} = \bigcap T_j, f_j \in F'_i$ it holds that $T_{F_i} \subseteq T_{F'_i}$. The latter suggests that $t_i \in T_{F'_i}$ which implies that t'_i cannot have more specified bits than t_i. This occurs because, t'_i can be any test in $T_{F'_i}$ then it can also be t_i. Moreover, in the case where $T_{F_i} \subset T_{F'_i}$, t'_i can even have fewer specified bits than t_i since a test in $T'_{F_i} - T_{F_i}$ may have fewer specified bits in T_{F_i}. Therefore, $sp(t'_i) \leq sp(t_i)$. Since, F'_i was arbitrarily selected and has no constraints except that of $F'_i \subset F_i$, then this statement holds for all proper subsets of F_i. □

Although, Theorem 3.1 can only guarantee that when replacing a test t_i with another t'_i detecting only a subset of the faults, the number of specified bits in t'_i cannot be more than those in t_i, we expect that removing faults from F_i will give tests with fewer specified bits. Thus, the test generation process should be able to find a test for the faults in F'_i that has fewer specified bits than t_i. Actually, the effectiveness of the proposed method depends greatly on the ability of the test generation process (line 7 of Figure 3.1) to derive tests with a large number of unspecified bits. Several existing methods can be used to solve this problem effectively. Both of the structural methods of [48, 49] propose specific ATPG-like routines (using implications, justifications, and testability measure concepts) to find a large test cube (test with a large number of unspecified bits) that detects a number of faults. Alternatively, the function-based routine that we describe in Section 4.5 can derive a large cube by extracting the shortest path in a BDD-based implementation. Any of these techniques can be used by the proposed method whose main contribution is not on this specific single test generation problem but on finding a systematic method to replace an entire test set such that the total number of unspecified bits is maximized.

The proposed algorithm is quite effective, as it can be concluded from the experimental results, yet is of the same order with other comparable techniques, in terms of time complexity. We give a brief worst-case time complexity analysis of the algorithm, expressed in necessary typical operations such as number of fault simulations and number of test generations preformed. The proposed algorithm requires, in the worst case, $|\mathcal{T}|$ fault simulations, and $|\mathcal{T}|$ test generations. This is in the same order as the complexity of [48], which requires, in the worst case, $3 \cdot |\mathcal{T}|$ fault simulations and $2 \cdot |\mathcal{T}|$ test-generations.

	Initial Test Set $\mathcal{T}$					Test Set $\mathcal{T}'$ (ord. 1)					Test Set $\mathcal{T}'$ (ord. 2)					Test Set $\mathcal{T}'$ (ord. 3)				
Fault	t_1	t_2	t_3	t_4	t_5	t_1	t_2	t_3	t_4	t_5	t_2	t_4	t_1	t_3	t_5	t_3	t_4	t_2	t_1	t_5
f_1	●	●				●	–				●		–					●	–	
f_2		●					●				●							●		
f_3	●		●			●		–					●	–		●			–	
f_4			●		●			●		–				●	–	●				–
f_5		●	●				●	–			●			–		●		–		
f_6	●		●	●		●		–	–			●	–	–		●	–		–	
f_7				●					●			●					●			
f_8				●	●				●	–		●			–		●			–
Sp. Bits.	**23**	**20**	**29**	**22**	**19**	**23**	**17**	**22**	**19**	**0**	**20**	**22**	**18**	**21**	**0**	**29**	**19**	**19**	**0**	**0**
Total Bits	**113**					**81**					**81**					**67**				
	(a)					(b)					(c)					(d)				

Table 3.1: Test-based Replacement Example: (a) Initial Test Set (b) Relaxed Test Set with ordering $t_1 < t_2 < t_3 < t_4 < t_5$, (c) Relaxed Test Set when considering essential faults first, (d) Relaxed Test Set with ordering $t_3 < t_4 < t_2 < t_1 < t_5$.

Next we present an illustrative example for the method of Figure 3.1. Assume an initial test set $\mathcal{T} = \{t_1, t_2, t_3, t_4, t_5\}$ and a circuit with 8 faults given in $F = \{f_1, f_2, ..., f_8\}$. Table 3.1(a) gives the fault simulation results for $\mathcal{T}$. The rows of the table correspond to the faults in the fault list F and the columns correspond to the tests in $\mathcal{T}$. A dot in a table cell indicates that the fault in the corresponding row is detected by the test in the corresponding column. For instance, test t_1 detects faults f_1, f_3 and f_6. The cells in the last row show the number of specified bits in the corresponding test. In the first iteration of the method, a test is generated detecting faults f_1, f_3 and f_6 at the same time. This new test, let that be t'_1, has as many unspecified bits as possible. Faults f_1, f_3 and f_6 are then dropped from further consideration. Next, t'_2 is generated for faults f_2 and f_5 only, since f_1 has already been covered by test t'_1. Test t'_2 is guaranteed to have at least as few specified bits as t_2, yet removing the constraint of detecting f_1 suggests the possibility of fewer specified bits in t'_2. In the following iteration t'_3 is generated and detects only f_4, whereas for the faults f_7 and f_8 a new test t'_4 is generated in the subsequent iteration. Finally, since t_5 only detects already detected by previous tests, it is no more necessary in the test set. Thus, the complete fault coverage termination condition (line 14 of Figure 3.1) determines the process which returns the relaxed test set $\mathcal{T}'$ (given in Table 3.1(b)). According to Theorem 3.1, no test in $\mathcal{T}'$ can have a larger number of specified bits than its corresponding tests in the initial test set $\mathcal{T}$. Moreover, it is expected that tests t'_2, t'_3 and t'_4 will have fewer specified bits than t_2, t_3 and t_4, respectively, since each of them targets fewer faults. In the case of t'_5 where no faults are left, the test is removed from the test set. In this example, the total specified bits reduction in the entire test set is 32 bits, giving a test set with only 81 specified bits instead of the 113 specified bits of $\mathcal{T}$.

The observation that targeting essential faults (i.e. faults that are targeted only by one test in the test set) first, made by the authors of [48], might be beneficial for a number of compaction techniques as well as the test set relaxation technique of [48]. To the contrary, it gives no advantage in this test-based replacement method, since targeting essential faults first does not guarantee that the resulting test will have larger number of specified bits. In Table 3.1(c) we give the relaxed test set when we consider essential faults first. Since this is a test-based method, considering essential faults first translates to examining the tests that target essential faults first. Despite the fact that the initial test set $\mathcal{T}$ of this example was randomly chosen to illustrate the algorithm, observe that no difference in the number of specified bits in $\mathcal{T}'$ was recorded. Actually, changing the ordering in which the tests are considered by our algorithm, can affect the number of specified bits in the resulting test set, yet no clear benefit was reported among all circuits during experimentation.

The results of this method clearly depend on the processing ordering of the tests in $\mathcal{T}$. For example, if we repeat the example of Table 3.1(a) using a decreasing test ordering on the number of faults detected by each fault, i.e. $t_3 < t_4 < t_2 < t_1 < t_5$, (since t_3 detects 4 faults, t_1, t_2 and t_4 detect 3 faults, and t_5 detects 2 tests) we have a different $\mathcal{T}'$ shown in Table 3.1(d). Here, $\mathcal{T}'$ contains 3 tests and a total of 67 specified bits. In this case more relaxation was achieved than applying the method with the different ordering (Table 3.1(b)). Our implemented tool allows running the method under various ordering heuristics including:

i. increasing ordering on the number of faults detected by each test,

ii. decreasing ordering on the number of faults detected by each test,

iii. considering tests detecting essential faults first,

iv. considering tests detecting essential faults last,

v. existing test ordering in the initial test.

However, our extensive experimentation using all these orderings shows no consistent advantage for one of the orderings considered. In short, all orderings give larger numbers of unspecified bits in some circuits and smaller numbers in the other circuits. Based, on that the only conclusion that can be extracted, is that this test-based replacement method is highly biased on the test ordering and that less biased methods should be investigated.

3.4 Fault-Based Replacement Method

This section discusses an alternative method for static test set relaxation which proceeds in a different direction to that of the method described in the previous section. Essentially, the method proposed in Section 3.3 is greedy, in the sense that the decision on which test should detect each fault is taken based on the ordering under which the tests are examined. As shown previously, this makes the test-based replacement method biased on the ordering of the tests. Here, we describe a more intelligent approach that gives well-defined criteria based on which the test that should target each fault is chosen. Thus, instead of concentrating on one test at a time, this method pivots on one fault at a time. For some fault f_i the algorithm determines the test $t_j \in \mathcal{T}$ to be the only one detecting the fault such that the number of bits that can be relaxed in the entire test set is maximized. Put differently, the algorithm determines the test to explicitly target the detection of the fault and relaxes the bits in the remaining tests required to detect the fault. We formulate the problem as an optimization problem were the proposed algorithm tries to identify the appropriate test for each based on a closed-form equation.

Consider a fault f_i detected by one or more tests in $\mathcal{T}$. Let $\mathcal{T}_i \subseteq \mathcal{T}$ denote the set of tests in $\mathcal{T}$ that detect fault f_i. Consider a test $t_k \in \mathcal{T}_i$.

Definition 3.1. The contribution of a fault f_i in a test t_k denoted by c_{ik}, is the number of specified bits in t_k that can be unspecified if t_k no longer detects f_i.

The total number of specified bits in $\mathcal{T}$ that can become unspecified if fault f_i is only detected by some test $t_j \in \mathcal{T}_i$ (and not by any other test in $\{\mathcal{T}_i - t_j\}$), is given by:

$$G_{ij} = \sum c_{ik}, \quad k \in \{\mathcal{T}_i - t_j\} \tag{3.1}$$

Thus, G_{ij} denotes the gain in unspecified bits if fault f_i is only explicitly targeted during the test generation of test t_j. Of course, coincidental detection of f_i by other tests may occur but this is done with no extra cost in terms of specified bits. Observe that, based on theorem Theorem 3.1, $sp(t_k) - c_{ik} \leq sp(t_k)$. Thus, the range of values of c_{ik} is $0 \leq c_{ik} \leq sp(t_k)$, since by definition c_{ik} cannot be more than $sp(t_k)$. In turn, the range of the values of G_{ij} is given by $0 \leq G_{ij} \leq \sum sp(t_k), \ \ k \in \{\mathcal{T}_i - t_j\}$.

In order to determine t_m, i.e. the test in $\mathcal{T} = \{t_1, t_2, ..., t_N\}$ that should explicitly target fault f_i, it suffices to calculate:

$$\hat{G}_{im} = max\{G_{ij}\}, \quad t_j \in \mathcal{T}_i \subseteq \mathcal{T} \tag{3.2}$$

Figure 3.2 shows the proposed algorithm. First, fault simulation is performed to derive the complete fault list F as well as the fault lists F_j for each test $t_j \in \mathcal{T}$. Then, the algorithm iterates over each fault $f_i \in F$, to determine the "best" test to detect f_i. This is done by examining only tests in $\mathcal{T}$ that detect f_i, that is $\mathcal{T}_i$ (lines 06-16). For every test $t_j \in \mathcal{T}_i$ the contribution of f_i in t_j (c_{ij}) is calculated (line 08). This is a crucial step which invokes a similar test generation routine to that of the method in Section 3.3 (line 07 of Figure 3.1). Specifically, to find c_{ij} for a fault f_i and a test t_j detecting the faults in F_j, we generate a test cube t' targeting faults in $F_j - f_i$ and calculate $c_{ij} = sp(t_j) - sp(t')$. This is the number of specified bits savings if test t_j no longer detects fault f_i. Once c_{ij} is calculated for every test $t_j \in \mathcal{T}_i$, the total gain G_{ij} in unspecified bits (meaning f_i is detected by t_j but not by $\{\mathcal{T}_i - t_j\}$) for every test t_j is easily computed (line 10). Consequently, the maximum gain is found (line 11) indicating the test $t_m \in \mathcal{T}_i$ selected to detect f_i.

The next step (lines 13-16) convey the dynamic nature of the algorithm. Once test t_m is determined as the most appropriate to detect f_i, it is no longer necessary for tests $\{\mathcal{T}_i - t_m\}$ to detect f_i. Therefore, the fault list F_j for each of the remaining tests in $\mathcal{T}_i$ is updated. In this manner, fault f_i will never be targeted in any subsequent test generation step (line 08). Observe that if a test's fault list becomes empty at any point, the test can be fully relaxed which means it can be dropped since all of the faults it detected are now detected by some

Procedure Fault-based_Replacement

```
Inputs:   circuit $\mathcal{C}$, test set $\mathcal{T}$, fault model $\mathcal{M}$
Outputs:  relaxed test set $\mathcal{T}'$
01: Fault Simulate $\mathcal{T}$ based on fault model $\mathcal{M}$
02: $F$ = list of faults detected by $\mathcal{T}$
03: for each test $t_j \in \mathcal{T}$
04:     $F_j$ = list of faults detected by $t_j$
05: for each fault $f_i \in F$
06:     $\mathcal{T}_i$ = list of tests detecting $f_i$
07:     for each test $t_j \in \mathcal{T}_i$
08:         use $F_j$ to calculate $c_{ij}$
09:     for each test $t_j \in \mathcal{T}_i$
10:         calculate $G_{ij} = \sum c_{ik}, \;\; k \in \{\mathcal{T}_i - t_j\}$
11:     $\hat{G}_{im} = \max\{G_{ij}\}, \;\; t_j \in \mathcal{T}_i$
12:     % test $t_m$ keeps fault $f_i$, tests $\{\mathcal{T}_i - t_m\}$ drop $f_i$
13:     for each $t_j \in \{\mathcal{T}_i - t_m\}$
14:         $F_j = F_j - f_i$
15:         if $F_j = \emptyset$
16:             $\mathcal{T} = \mathcal{T} - t_j$ % drop test $t_j$
17: $\mathcal{T}' = \emptyset$
18: for each test $t_j \in \mathcal{T}$
19:     generate test $t'_j$ that detects all faults in $F_j$
20:     add $t'_j$ to $\mathcal{T}'$
21: return $\mathcal{T}'$
```

Figure 3.2: Static Fault-based Replacement Method.

other test(s). The fault coverage of $\mathcal{T}$ is maintained since every fault f_i is guaranteed to be detected by some test t_m with $F_m \neq \emptyset$.

Once all faults are examined, the relaxed test set $\mathcal{T}'$ is generated based on the updated fault list F_j for each test t_j that has remained in $\mathcal{T}$ (lines 17-20). Each new test $t'_j \in \mathcal{T}'$ is guaranteed to detect a subset of the faults detected by the corresponding test $t_j \in \mathcal{T}$, since the size of the updated fault list per test is reduced (in most cases) or, in the worst case, remains the same.

The worst-case time complexity of this approach in terms of fault simulations and test generations performed is $|\mathcal{T}|$ fault simulations plus $|\mathcal{T}| \cdot |F|$ + $|\mathcal{T}|$ test generations. In practice, however, the factor $|\mathcal{T}| \cdot |F|$ is much smaller since each fault $f_i \in F$ is examined only against the small number of tests in $\mathcal{T}_i \subseteq \mathcal{T}$ that detect the fault, and not for the entire test set $\mathcal{T}$.

Moreover, in practice, the initial test set $\mathcal{T}$ is small since it is required to be compact to reduce the test application time.

The following Lemma is used to show the effectiveness of the fault-based replacement method. Let us first assume that there exists a test generation process that identifies the test that detects a number of faults with a minimal number of specified bits. We elaborate on this assumption later in this section.

Lemma 3.1. For a given test set $\mathcal{T}$, the fault-based replacement method (Figure 3.2) identifies a test $t \in \mathcal{T}$ that detects fault f_i with the minimum overhead in specified bits, under some fault ordering in F.

Proof. When $|\mathcal{T}_i| = 1$ then test $t_j \in \mathcal{T}_i$ is clearly the one with the minimum overhead in specified bits for targeting f_i. When $|\mathcal{T}_i| > 1$, the values for the G_{ij} for fault f_i are given by $G_{ij} = \sum c_{ik},\ t_k \in \{\mathcal{T}_i - t_j\},\ \forall t_j \in \mathcal{T}_i$ (by Equation 3.1). That is the total cost in specified bits of testing f_i with tests in $\mathcal{T}$ $\big($i.e., $\sum c_{ik},\ t_k \in \mathcal{T}\ \big)$ minus the cost for testing f_i with t_j (i.e. c_{ij}). Recall that by assumption the contribution in specified bits of each fault at a test (i.e. c_{ik}), can be computed optimally. This occurs since the test generation process gives the test with the minimal number of specified bits for a group of faults. Since the decision on which test explicitly targets f_i in $\mathcal{T}'$ is taken by selecting the maximum over the G_{ij}, then this number is the total cost in specified bits for testing f_i with tests in $\mathcal{T}$ minus the minimum cost for testing f_i with a test in $\mathcal{T}$ i.e., $\hat{G}_{im} = \max_j\{G_{ij}\} \Rightarrow \hat{G}_{im} = \max_j\{-c_{ij} + \sum c_{ik},\ t_k \in \mathcal{T}\} \Rightarrow \hat{G}_{im} = \sum c_{ik},\ t_k \in \mathcal{T} - \min_j\{c_{ij}\}$. The total cost in specified bits for testing f_i with tests in $\mathcal{T}$ is constant and all c_{ij} have non-negative integer values; thus, the selected test gives the minimum overhead in specified bits for testing f_i with a test in $\mathcal{T}$. By considering different ordering of the faults may change the values for the c_{ij}. To prove this, consider two tests f_y and f_x that are both tested by a specific test t_u, i.e., $t_u \in \mathcal{T}_x$, $t_u \in \mathcal{T}_y$ and $f_x, f_y \in F_u$. If we first consider fault f_x (i.e. iteration for fault f_x is before iteration for fault f_y) and the selected test to target f_x is not t_u, then $F_u = F_u - f_x$ (step 14 in Figure 3.2). Thus, calculation of c_{yu} in the iteration for fault f_y (step 8 of Figure 3.2) may give a different value than the case where the original F_u was used. That is, if we first consider fault f_y (i.e. iteration for fault f_y is before iteration for fault f_x), calculation of c_{yu} will be done using F_u and not $F_u - f_x$. Since changing the ordering the faults are considered may change the values of c_{ij}, this statement holds only for the ordering considered and does not give a global minimum for the testing overhead in specified bits for fault f_i. $\square$

Table 3.2: Fault-based Replacement Example

Initial Test Set $\mathcal{T}$						**Relaxed Test Set $\mathcal{T}'$**				
t_1	t_2	t_3	t_4	t_5	**Fault list**	t_1	t_2	t_3	t_4	t_5
●	●				f_1	●	–			
	●				f_2		●			
●		●			f_3	–		●		
		●		●	f_4			●		–
	●	●			f_5		●	–		
●		●	●		f_6	–		–	●	
			●		f_7				●	
			●	●	f_8				●	–
23	**20**	**29**	**22**	**19**	**Spec. Bits**	**16**	**16**	**20**	**22**	**0**
113					**Total Spec. Bits**	**74**				

(a) (b)

	Iteration for f_1					**Iteration for f_2**					**Iteration for f_3**					**Iteration for f_4**				
Fault list	t_1	t_2	t_3	t_4	t_5	t_1	t_2	t_3	t_4	t_5	t_1	t_2	t_3	t_4	t_5	t_1	t_2	t_3	t_4	t_5
f_1	3	4				●	–				●	–				●	–			
f_2							8					●					●			
f_3											5		3			–		●		
f_4																		4		5
f_5																				
f_6																				
f_7																				
f_8																				
G_{ij}	*4*	*3*									*3*		*5*					*5*		*4*
Spec. Bits	**23**	**20**	**29**	**22**	**19**	**23**	**16**	**29**	**22**	**19**	**23**	**16**	**29**	**22**	**19**	**18**	**16**	**29**	**22**	**19**

(c)

	Iteration for f_5					**Iteration for f_6**					**Iteration for f_7**					**Iteration for f_8**				
Fault list	t_1	t_2	t_3	t_4	t_5	t_1	t_2	t_3	t_4	t_5	t_1	t_2	t_3	t_4	t_5	t_1	t_2	t_3	t_4	t_5
f_1	●	–				●	–				●	–				●	–			
f_2		●					●					●					●			
f_3	–		●			–		●			–		●			–		●		
f_4			●		–			●		–			●		–			●		–
f_5		4	5				●	–				●	–				●	–		
f_6						5		4	3		–		–	●		–		–	●	
f_7														1					●	
f_8																			4	14
G_{ij}		*5*	*4*			*7*		*8*	*9*										*14*	*4*
Spec. Bits	**21**	**16**	**24**	**22**	**14**	**21**	**16**	**22**	**22**	**14**	**16**	**16**	**20**	**22**	**14**	**16**	**16**	**20**	**22**	**14**

(d)

Obviously, the assumption on the test generation is not realistic since it implies full exploration of a circuit's test space, which is known to be an NP-hard problem. Yet, together with Lemma 3.1 it ensures that if the test generation process is effective in generating tests with a large number of unspecified bits, the resulting test set will have a large number of unspecified bits, optimizing the test set relaxation method.

Next we give an illustrative example for the method of Figure 3.2. Consider an initial test set $\mathcal{T} = \{t_1, t_2, t_3, t_4, t_5\}$ detecting the 8 faults in $F = \{f_1, f_2, ..., f_8\}$ shown in Table 3.2(a). The rows of the table correspond to the faults in the fault list F. The columns of the table correspond to tests in the test set under consideration ($\mathcal{T}$). A bullet on a cell denotes that the fault of that row is detected by the test of the corresponding column, in the initial test set. The last two rows report the number of specified bits for the test of the corresponding column and the total number of specified bits in the entire test set, respectively.

Tables Table 3.2(c) and (d) summarize the execution of the proposed algorithm. Each subtable corresponds to an iteration of the algorithm, noted at the header. Recall that each iteration examines one fault and so iteration 1 is for f_1, iteration 2 for f_2 and so on. The test and fault orientation is as in the Table 3.2(a). An entry between a test t_k and a fault f_i, for some iteration, denotes the number of specified bits that can be relaxed if fault f_i is no longer detected by the test that will replace test t_k in the final test set (i.e., c_{ik}). For instance, consider fault f_1 during the first iteration, which is detected by tests t_1 and t_2. The entry between f_1 and t_1 (t_2) gives $c_{11} = 3$ ($c_{12} = 4$), meaning that if f_1 is no longer detected by t_1 (t_2) the number of bits that can be relaxed is 3 (4). Consequently, $G_{11} = 4$ and $G_{12} = 3$, given in the row labeled G_{ij} in the table. The last row reports the number of specified bits in a test t_j, when all the faults in its fault list for a specific iteration (faults in F_j) are explicitly targeted. An empty entry indicates that the fault is not detected by the corresponding test. We only show the values of c_{ij} for the fault corresponding to the iteration considered.

During the first iteration, enforcing detection of f_1 by t_1 and not by t_2 results in more unspecified bits than the opposite ($G_{11} > G_{12}$). So the decision is to keep f_1 in F_1 and discard it from F_2. This is shown in the next subtable (iteration for f_2) with a dot under t_1 and a hyphen under t_2. The number of specified bits in t_2 has been reduced by 4 as the effect of not considering f_1 when generating a test for the remaining faults in F_2. Moving to the second iteration (targeting fault f_2) the gains for the tests that detect f_2 are computed, in order to reflect the new situation after removing f_1. This computation gives 8 bits gain if we exclude

f_2 from t_2. Since f_2 is only detected by t_2, no decision needs to be made and the algorithm moves on to the next iteration for f_3. Here, f_3 is selected to be detected by t_3 and not by t_1, resulting in 5 specified bits reduction for test t_1. In the same manner, after iterations for fault f_4 and f_5, t_3 and t_2 are selected for faults f_4 and f_5, respectively. The process continues until all faults are examined. The Table 3.2(b) shows the final distribution of the faults between the tests. Each column determines the targeted fault list for the corresponding test. For each list a test is generated to replace the one originally present in the given test set (lines 22-23 of Figure 3.2). Tests with empty lists (t_5 in this example) are removed from the test set. The proposed method gives a test set with 74 specified bits, instead of 113 in the original test set. Note that, for the proposed fault-based method the ordering under which the tests are considered is not important for the number of specified bits in the resulting test set. However, the ordering under which the faults are examined may affect the effectiveness of the proposed method. This happens since the computation of G_{ij} is based on the list of faults that are detected by a test (i.e. F_i for test $t_i \in \mathcal{T}$). If we change the ordering under which the faults are examined, the elements of the lists F_i will be different for the same iteration of the algorithm. Experimentation shows that the fault ordering does not considerably affect the results of this fault-based method.

3.5 Post-Processing Compaction Step

The performance of the presented methodologies is biased on the order in which the tests in $\mathcal{T}$ (for the test-based method) as well as the faults in F (for the fault-based method) are considered. Thus, as in the case of all existing methods that examine the same problem, the algorithms are not optimal and it is possible to further reduce the number of tests in the relaxed test set $\mathcal{T}'$. We propose a test set compaction algorithm that formulates the problem as a system of constraint equations, similar to the unate covering problem. Based on information obtained by the fault simulation the constraint equations are formed in such a way that they correspond to all fault considered. Each variable of the Boolean equation represents a pattern in the test set and the optimal solution will give a single variable becoming true at each equation. However, this problem is known to be NP-hard and, thus, we use heuristic algorithm to solve the problem, giving a near-optimal solution.

For each fault in the fault list F, a constraint Boolean equation representing all tests that detect the fault is given by:

$$C_{f_i} = \bigvee_{t_j \ detects \ f_i} t_j \tag{3.3}$$

where f_i and t_j are boolean variables corresponding to faults and tests, respectively. Satisfying each of these equations ensures detection of the corresponding fault by at least one test. In order to preserve the fault coverage of $\mathcal{T}$ it is necessary to also find a satisfying solution for:

$$1 = \bigwedge_{f_i \in F} f_i \tag{3.4}$$

The solution for the system of equations formed by Equation 3.3 and Equation 3.4 results in a logic value assignment on the variables t_i and f_i. All variables f_i get a logic one value (due to Equation 3.4). A test variable t_i given a logic zero variable implies, that the test corresponding to t_i can be safely dropped from the test set. The system can be solved using any two-level logic minimization procedure such as Quine-McCluskey or Espresso Exact. Our experimentation showed test set size reduction for uncompacted or moderately compacted test sets.

3.6 Experimental Results

The proposed methods were implemented in C language and run on a SunBlade 1500 machine, running Solaris with 4GB of RAM. We experimental with the ISCAS'85 and the full-scan versions of the ISCAS'89 benchmark circuits. The initial test sets were derived from ATALANTA, for stuck-at faults. Two types of initial test sets were used; one fully-specified and optimized in terms of compaction and the other optimized in terms of unspecified bits. Furthermore, we experimented with the compact test sets provided by the authors of [48]. The fault simulation and test generation steps were implemented using an in-house function-based tool for single stuck-at faults, based on BDDs.

Table 3.3 lists the results obtained by the proposed methodologies, for a number of circuits, for each of the two initial test sets obtained by ATALANTA. We show results for the larger ISCAS'89 as well as all the ISCAS'85 benchmark circuits. We do not report results for circuit c6288, since it is known that its representation with BDDs is not feasible. Partitioned

Table 3.3: Results of the proposed methods for two different initial test sets.

		Initial Test Set			**Test-based Replacement**					**Fault-based Replacement**				
Circuit	**PIs**	$\|\mathcal{T}\|$	Sp. Bits	$K(\mathcal{T})$	$\|\mathcal{T}'\|$	Sp. Bits	$K(\mathcal{T}')$	**Red. (%)**	CPU(secs)	$\|\mathcal{T}'\|$	Sp. Bits	$K(\mathcal{T}')$	**Red. (%)**	CPU(secs)
Initial Test Sets Optimized for Test Set Size														
c2670	233	113	32853	1	60	3259	0.099	**90.1**	63.1	59	3384	0.103	**89.7**	62.1
c7552	207	238	49266	1	171	9444	0.192	**80.8**	6.1	75	6645	0.135	**86.5**	5.4
s838.1	66	149	9834	1	141	3406	0.346	**65.4**	0.3	140	3189	0.324	**67.6**	0.2
s1196	32	144	4608	1	125	1701	0.369	**63.1**	0.8	123	1657	0.360	**64.0**	0.8
s1238	32	158	5056	1	138	1845	0.365	**63.5**	0.9	124	1697	0.336	**66.4**	0.8
s1423	91	71	6461	1	24	1215	0.188	**81.2**	0.3	24	1174	0.182	**81.8**	0.2
s9234.1	247	365	90155	1	122	8512	0.094	**90.6**	3.4	123	8418	0.093	**90.7**	3.1
s13207.1	700	472	330400	1	275	11957	0.036	**96.4**	12.9	265	11453	0.035	**96.5**	12.4
s15850.1	611	441	269451	1	99	13003	0.048	**95.2**	7.6	100	13185	0.049	**95.1**	7.1
s38584.1	1464	637	932568	1	115	33215	0.036	**96.4**	23.1	114	33401	0.036	**96.4**	23.2
					Average Reduction:			**82.3**					**83.5**	
Initial Test Sets Optimized for Unspecifed bits														
c2670	233	141	14620	0.45	81	3489	0.106	**76.1**	63.6	58	3413	0.104	**76.7**	63.8
c7552	207	258	31866	0.65	111	8054	0.163	**74.7**	6.2	110	7737	0.157	**75.7**	5.9
s838.1	66	184	6569	0.67	140	3312	0.337	**49.6**	0.3	144	3312	0.337	**49.6**	0.3
s1196	32	155	3546	0.77	126	1708	0.371	**51.8**	1.0	124	1677	0.364	**52.7**	0.9
s1238	32	164	3724	0.74	133	1801	0.356	**51.6**	0.9	134	1779	0.352	**52.2**	0.7
s1423	91	83	5955	0.92	41	1449	0.224	**75.7**	6.9	44	1448	0.224	**75.7**	6.1
s9234.1	247	495	65931	0.73	124	8568	0.095	**87.0**	3.7	126	8601	0.095	**87.0**	3.2
s13207.1	700	692	256333	0.78	274	11956	0.036	**95.3**	12.9	278	12125	0.037	**95.3**	11.2
s15850.1	611	519	163698	0.61	102	13241	0.049	**91.9**	7.9	103	13541	0.050	**91.7**	7.5
s38584.1	1464	840	868090	0.93	114	33860	0.036	**96.1**	26.1	114	34012	0.036	**96.1**	25.2
					Average Reduction:			**75.0**					**75.3**	

BDD techniques [50] can be used in order to represent this circuit and other larger and/or more complex circuits. Columns 1-2 list the circuit name and the number of Primary Inputs, respectively. Columns 3-5 give information regarding the initial test set. Column 3 lists the number of tests in the initial test set; Column 4 lists the total number of specified bits, and Column 5 gives $K(\mathcal{T})$ which is the ratio of the specified bits over the total number of bits in the test set. For the fully specified test sets $K(\mathcal{T}) = 1$. Columns 6-8 and 10-12 show the same information for the relaxed test set $\mathcal{T}'$ obtained after applying the methods of Section 3.3 and Section 3.4, respectively. Columns 9 and 13 give the reduction (%) achieved by the two methods, calculated by 1-$K(\mathcal{T}')/K(\mathcal{T})$. In all results the postprocessing step described in Section 3.5 has been applied on top of the two methods.

The average reduction in specified bits obtained by the test-based replacement method is 82.3 % for the initial compact test sets and 75 % for the initial partially-specified test sets. For the fault-based replacement this reduction is 83.5 % and 75.3 %, respectively. Obviously, the fault-based method gives better results for most of the circuits. However, the reduction rates are very similar between the two methods, especially when the initial test sets have unspecified bits. Moreover, the reduction is higher when the initial test set to be relaxed is fully specified. However, the proposed methods also give significant reduction for the partially specified test sets. Of course, the final values of $K(\mathcal{T}')$ (Columns 8 and 13) are very similar for the two different types of test sets, indicating a saturating behavior of the proposed methods for the given test sets. Another interesting observation is that the size of the relaxed test set $\mathcal{T}'$ is also significantly reduced (compacted), irrespective of the initial test set, as it can be concluded by comparing Column 3 with Columns 6 and 11.

Table 3.4: Comparison with existing work.

		Initial Test Set			**[48]**				**Proposed**			
Circuit	**PIs**	**Tests**	S_{max}	**Sp. Bits**	S_{max}	S_{avg}**/PI (%)**	**Sp. Bits**	**Red. (%)**	S_{max}	S_{avg}**/PI (%)**	**Sp. Bits**	**Red. (%)**
c880	60	21	60	**1260**	60	65.6	**827**	34.4	50	62.7	**790**	37.3
c1355	41	84	41	**3444**	41	100.0	**3444**	0.0	41	82.3	**2836**	17.7
c1908	33	106	33	**3498**	33	83.0	**2903**	17.0	32	53.0	**1853**	47.0
c2670	233	45	233	**10485**	233	29.3	**3072**	70.7	183	26.4	**2767**	73.6
c3540	50	93	50	**4650**	50	46.7	**2172**	53.3	41	42.6	**1982**	57.4
c5315	178	46	178	**8188**	178	38.6	**3161**	61.4	132	36.0	**2948**	64.0
c7552	207	75	207	**15525**	207	45.5	**7064**	54.5	163	42.6	**6613**	57.4
s1238	32	125	32	**4000**	32	43.5	**1740**	56.5	23	42.5	**1698**	57.6
s1423	91	24	91	**2184**	91	58.0	**1267**	42.0	73	53.5	**1168**	46.5
s1494	14	100	14	**1400**	14	72.6	**1016**	27.4	14	72.0	**1008**	28.0
s9234	247	111	247	**27417**	247	31.0	**8499**	69.0	218	30.2	**8293**	69.8
s13207	700	235	700	**164500**	700	8.0	**13160**	92.0	672	7.3	**12031**	92.7
s15850	611	97	611	**59267**	611	22.7	**13454**	77.3	408	20.6	**12186**	79.4
s38417	1664	84	1664	**139776**	1664	25.2	**35224**	74.8	1145	22.9	**32016**	77.1
s38584	1464	114	1464	**166896**	1464	18.9	**31543**	81.1	1321	18.0	**29984**	82.0

Table 3.4 provides a comparison between the proposed methodologies and that of [48], which reports the highest reduction among all existing techniques that examined the considered problem. For a fair comparison, we experimented with the same initial test sets of [48], which are fully specified and very compact [1]. Column 3 reports the size of the initial test set and Column 5 gives the number of specified bits in the test set. The results of [48] are given in Columns 6-9 and those of the proposed methodologies in Columns 10-13 (the best results among the two methods are considered). Based on the reduction percentage for each of the methods (Columns 9 and 13) we note that the proposed methods always outperform that of [48]. In some cases the increase in our reduction is marginal; however, in some other cases this increase is considerable (such as for circuits C1355 and C1908). For both methods, the size of the relaxed test set $\mathcal{T}'$ equals that of the initial test set $\mathcal{T}$. This occurs because the initial test sets are very compact and all their tests detect at least one essential fault. In these

[1]We would like to thank the authors of [48] for kindly providing us their input test patterns

cases, the post-processing step of Section 3.5 gives no improvement. Columns 8 and 12 show the average number of specified bits among tests as a percentage of the number of primary inputs, i.e. the portion of unspecified bits at each test over the number of primary inputs. Another useful observation is that the maximum number of specified bits on a test, denoted by S_{max} (Columns 4,6,10), is significantly reduced by the proposed method, in many cases. This value is very important in some applications, such as deterministic LFSR-based BIST, because it determines the size of the LFSR hardware. Chapter 5 elaborates further on this issue.

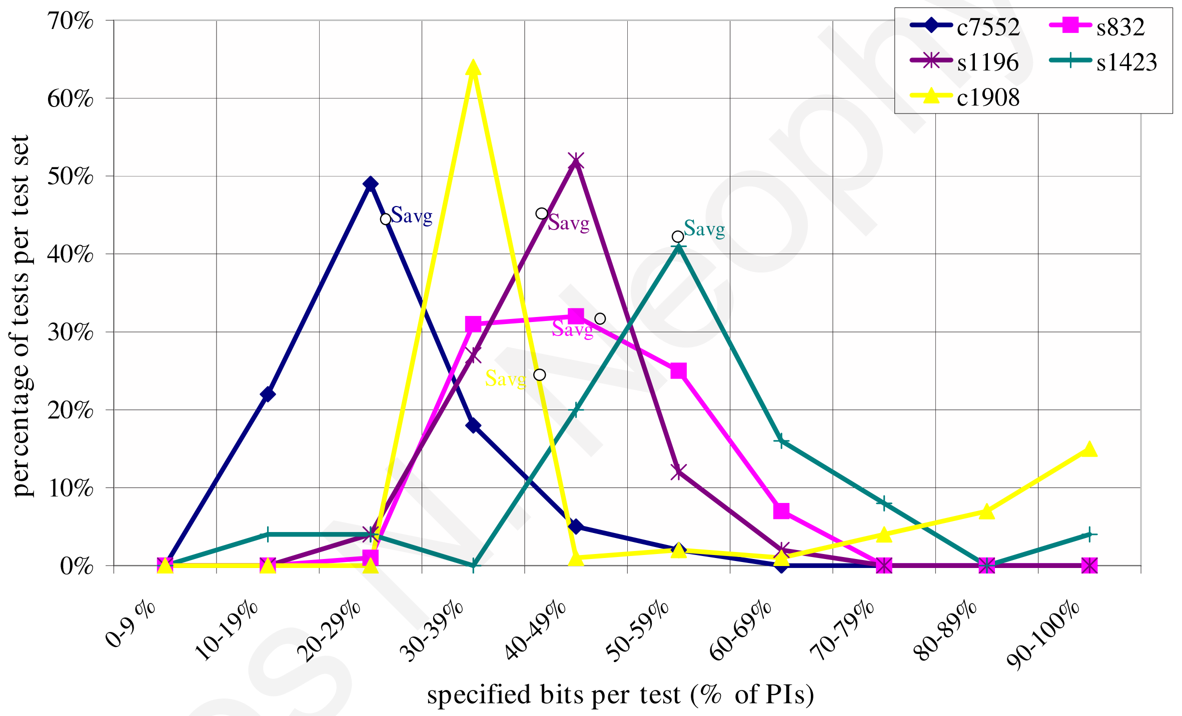


Figure 3.3: Specified bits distribution among generated tests of Table 3.4

The values reported in the Columns 8 and 12 of Table 3.4 show the average number of specified bits (S_{avg}) as a percentage on the maximum size of a test pattern. Since our experimentation is done using full-scan versions the maximum size of a pattern is identical to the number of primary inputs of the circuit examined. This number, together with the S_{max} value, give an indication on the distribution of the specified bits. Of course, these two numbers alone may result in misleading conclusions regarding the specified bits distribution among test patterns. Yet, as it can be concluded from our experimentation, the test sets resulting from both the proposed static methods tend to have a lot of patterns whose numbers of unspecified bits are close to the value of S_{avg}. Figure 3.3 shows the distribution of specified bits, in the test patterns, for 5 indicative circuits. The horizontal axis shows the percentage of specified bits per test, while the vertical axis shows the distribution of tests in the test set based on the number of specified bits per test. Observe that in all cases the number of specified bits in the majority of tests is close to the value of S_{avg}. This kind of distribution can benefit certain applications like test pattern concatenation used in the test set embedding scheme of [44]. We investigate this application in Chapter 5. Analogous specified bits distribution is recorded for all the circuits examined.

3.7 Conclusions

Two systematic methodologies for increasing the unspecified bits in a static test set have been presented. The first method is test oriented. It iterates among tests, under a given ordering, removing fault detections that have been covered by an already examined test. Essentially, this is a straight forward, easy to implement method that gives high specified bits reduction ratio. The second method proceeds on a fault oriented rationale. It iterates among faults in order to identify the test in the test set, that gives the minimum cost in specified bits, when detecting the fault examined. The test selection for each fault is done based on a gain factor which is dynamically calculated, depending on the faults that remain undetected at each iteration. Finally, we propose a post-processing compaction technique that reduces the test size of the relaxed test set. This step can be applied on top of both the techniques proposed here, and gives small reduction in the number of tests in the resulting tests when the given test set is not highly compacted.

The reported experimental results demonstrate the effectiveness of the proposed methods in achieving high specified bit reduction rates, for various types of test sets under the stuck-at fault model (additional, linear fault models, such as the one discussed in Chapter 2 also apply). Moreover, they give a smooth distribution of specified bits among the tests in the test set, as well as small test set sizes. While the fault-based method gives slightly better results than the test-based method, both methods outperform all the previously proposed relaxation techniques. Nevertheless, since these two techniques are based on a given test set, the obtained reduction of specified bits is limited by the initial test set. Thus, at this point, it would be interesting to study if better results can be obtained when there is no constraint on an initial test set. Thus, the following chapter, presents dynamic techniques that generate compact test sets with many unspecified bits.

CHAPTER 4

DYNAMIC TEST GENERATION WITH LARGE NUMBER OF DON'T CARE BITS

4.1 Introduction

While test set relaxation is highly desired when we want to maintain some properties of a test set obtained by a certain test generation technique, starting from an initial test set may limit the unspecified to specified bits ratio that can be achieved. This chapter investigates test generation with many unspecified bits. Our motivation is analogous to compact test set generation, yet in this context we impose extra constraint to keep the number of specified bits in the test set as small as possible. Thus, this additional constraint makes the problem essentially different from the well studied problem of test set compaction. Test set compaction is a traditional problem in the VLSI testing area. Its goal is, mainly, to reduce the test application time by reducing the number of tests in a test set. A number of other parameters also benefit from small test sets, such as area requirements in test set embedding as well as the usage of ATE resources. Various algorithms for compact test set generation have been proposed in the literature, for various applications [51, 47, 52].

At the same time, test sets with many unspecified bits (inputs with don't care values) have become desirable for a number of applications. A large number of unspecified bits in the test set gives the flexibility to fix values appropriately for low power dissipation during test application [11, 53, 42]. Also, don't care bits are typically beneficial in increasing the encoding efficiency of many deterministic test set embedding schemes, such as [46, 44, 45]. For example, in LFSR-based deterministic ATPG ([44, 45]), it is desirable to keep the number of specified bits per pattern bounded to reduce the LFSR overhead, in addition to minimizing the total number of bits stored on the chip. Having many unspecified bits can also be advantageous when enriching a test set with some additional desired property, like the n-detect property (also see Chapter 7) or detecting additional fault types (also see Chapter 2).

Existing methods for test set compaction are categorized as static or dynamic. Static techniques [51, 47] are applied on top of the ATPG process, to compact the given test set while maintaining the fault coverage. Dynamic compaction [54, 55] involves test replacement through test generation, to explicitly consider test reduction during the test generation procedure. Existing methods for identifying don't care bits in a test set (compact or not) are mainly static, although they use some ATPG concepts such as implications and justifications [48] and testability measures [49], in order to modify the given test set. The two static methods proposed in Chapter 3 outperform existing methods, mainly because they do not rely on the structure of the circuit and, thus, can take decisions in a less localized manner. The existing dynamic methods for generating test sets with a large number of don't care bits are usually application oriented. For instance, the work of [44] invokes a test generation process which gives test sets with large numbers of unspecified bits, prior to the application of their proposed test set embedding technique. However, this process is tuned toward the specific application of deterministic test set embedding for LFSR-reseeding based BIST and, therefore, may result in a larger number of tests while trying to minimize the LFSR size. ATPG tools that allow for unspecified inputs can also be used; however, their resulting test sets are usually too large since compaction methods using reverse-order fault simulation [56] or double detection [57] do not apply to test sets with unspecified bits.

This chapter presents two methodologies for generating small test sets containing a small number of specified bits, in a dynamic manner. Thus, test generation and compaction are performed in a unified single phase, which is optimized with respect to three measures:

(i) small number of specified bits per test,

(ii) small number of tests in the test set, and

(iii) small number of total specified bits in the test set.

The first two measures come from the motivation constraints discussed in the previous paragraph, while the latter is a natural outcome of the first two, essentially defining the overall goal of the dynamic methods proposed here. Whereas static methods attempt to relax the specified bits in a given test set to don't care values, the proposed techniques proceed toward an opposite rationale, trying to minimize the number of specified bits during test generation. The idea behind both methods is to identify a large number of compatible faults (faults that can be mutually tested by a single test) at a time. This is a concept that has been traditionally used in compaction methods (see [47], among others). Here we consider a constrained version of compatible faults, where *two (or more) faults are considered compatible iff they can be mutually detected by a test with a small number of specified bits*. The proposed approaches employ ATPG methods to identify groups of such compatible faults. Faults are placed in the same compatibility group (and, thus, will be detected by a single test), only if there exists a test whose number of specified bits does not exceed a user defined threshold and detects all faults of the group. The considered threshold is bounded and explored systematically. This problem is modeled as an undirected graph, which we call *constraint fault-compatibility graph* and use in both methods.

The first method is essentially a straight forward match-and-merge algorithm. The method is iterative on the graph's vertices and proceeds in two phases. In the first phase the algorithm identifies compatible faults, corresponding to graph vertices, considering the entire fault compatibility graph. A local optimization selection criterion is used to identify pairs of vertices of minimal cost in terms of specified bits. In the second phase the vartices of each compatibility pair are merged into a single vertex, and the constrained compatibility graph is reconstructed to reflect the new relationships between faults. The second method is essentially a hierarchical refinement of the previous method. At each iteration it considers only a small part of the graph (i.e. a subgraph) to identify the best matching among compatible faults. The size of the subgraph can be explored in order to determine the value that gives the best results in terms of specified bits. Moreover, this exploration can help handle large circuits and, thus, give extra importance to the hierarchical nature of the second method.

Both the processes of identifying fault compatibility and of merging vertices, corresponding to faults, involve test generation. Since test sets with a large number of unspecified bits are desired, the test generation routine must have the ability of generating tests with many don't cares. Ideally thes test that has the fewer specified bits among all tests that detect a fault or a group of faults is desired. The two approaches we propose here, can be combined with any structural or function-based test generation procedure that gives test sets with many don't care bits. However, this chapter includes a function-based framework based on Binary Decision Diagrams (BDDs) [34] which is used by the proposed methods, as well as with the methods of Chapter 3.

As in Chapter 3 we experimented with the popular stuck-at fault model, yet any other fault model with a linear number of faults can be considered. Both of the proposed dynamic methods give small number of specified bits per generated test set, as it can be observed from the experimental results. Moreover, the results show higher specified bit reduction ratios than any static test relaxation technique. At the same time, they give high fault coverage while keeping the test set size small.

In the remainder of this chapter, we first give the necessary notation and preliminaries (Section 4.2). Next, we describe the match-and-merge dynamic algorithm (Section 4.3), while in Section 4.4 we describe the hierarchical merging dynamic algorithm. Section 4.5 provides necessary implementation details for the ATPG process, including a function-based BDD framework, which we have developed for the purpose of this dissertation. Finally Section 4.6 presents the experimental results and relevant discussion, whereas Section 4.7 concludes the chapter.

4.2 The Constrained Fault Compatibility Graph

This section defines the constrained fault compatibility graph that is used by both of the algorithms we propose in this chapter.

Let $sp(t)$ denote the number of specified bits in some test t. Given a set of tests $\mathcal{T}$, $S_{max} = max\{sp(t)|t \in \mathcal{T}\}$, i.e. S_{max} is the maximum number of specified bits in a test of $\mathcal{T}$. Also, let $\mathcal{T}_i$ denote the set of all possible tests that detect some fault f_i.

Consider a fault list $F = \{f_1, f_2, ..., f_n\}$ with corresponding sets of tests $\mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_n$. A weighted undirected graph $G(V, E)$, where V is the set of vertices and E is the set of edges, is defined as the *fault compatibility graph*. In this graph, every vertex $\upsilon_i \in V$ initially corresponds to a fault $f_i \in F$. Thus, $|V| = |F|$. An edge $e_{ij} = (\upsilon_i, \upsilon_j) \in E$ exists if and only if the two corresponding faults f_i and f_j can both be detected by at least one test. In general, compatibility can be expanded to more than two faults. For example, consider faults, f_1, f_2 and f_3. These faults are compatible iff $\mathcal{T}_1 \cap \mathcal{T}_2 \cap \mathcal{T}_3 \neq \emptyset$. If the latter condition holds, then it can be concluded that there is at least one test that detects all three faults. Thus, every vertex may correspond to a group for faults, instead of a single fault, in the case, for example, that two vertices are merged into one. The weight at some vertex $\upsilon_i \in V$ denoted by $w(\upsilon_i)$, is the smallest number of specified bits in some test t that detects fault f_i corresponding to vertex υ_i, i.e., $w(\upsilon_i) = \min\{sp(t)|t \in \mathcal{T}_i\}$. The weight of a vertex gives the minimum cost, in terms of number of specified bits, to detect the corresponding fault or group of faults. The weight at some edge $e_{ij} \in E$, denoted by $W(e_{ij})$, is the smallest number of specified bits in some test t that detects both faults (group of faults), f_i and f_j, associated with e_{ij}, i.e., $W(e_{ij}) = \min\{sp(t)|t \in \mathcal{T}_i \cap \mathcal{T}_j\}$. In the case of an edge, the weight gives the minimum number of specified bits required to detect the two associated faults (group of faults) by a single test.

Now, consider two sets of tests F_i and F_j, with corresponding sets of tests $\mathcal{T}_i$ and $\mathcal{T}_j$. Each test in $\mathcal{T}_i$ ($\mathcal{T}_j$) detects all faults in F_i (F_j). $\mathcal{K}$-compatibility between the sets of faults F_i and F_j is defined as follows.

Definition 4.1. Two sets of faults F_i and F_j are $\mathcal{K}$-compatible if and only if all of the following hold:

i. There exists at least one test that detects all faults in F_i and F_j , i.e. $\mathcal{T}_i \cap \mathcal{T}_j \neq \emptyset$.

ii. At least one test t can be found in $\mathcal{T}_i \cap \mathcal{T}_j$ that contains no more than $\mathcal{K}$ specified bits, i.e. $\exists\, t \in \mathcal{T}_i \cap \mathcal{T}_j : sp(t) \leq \mathcal{K}$.

The *constrained fault compatibility graph* is a subgraph of a fault compatibility graph where all vertices corresponding to set of faults that are not $\mathcal{K}$-compatible have been removed. Stated otherwise, the constrained fault compatibility graph has edges only between sets of faults that are $\mathcal{K}$-compatible, i.e. can be both tested by at least one test that has no more than $\mathcal{K}$ specified bits. Like compatibility, $\mathcal{K}$-compatibility can be expanded to more than two sets of faults. For example, consider the sets of faults, F_1, F_2 and F_3. These set of faults are $\mathcal{K}$-compatible iff $\mathcal{T}_1 \cap \mathcal{T}_2 \cap \mathcal{T}_3 \neq \emptyset$ and $\exists\, t \in \mathcal{T}_1 \cap \mathcal{T}_2 \cap \mathcal{T}_3$ such that $sp(t) \leq \mathcal{K}$. If the two conditions hold, then it can be concluded that there is at least one test that detects all faults in all three sets and does so with no more than $\mathcal{K}$ specified bits. Of course, $W(e_{ij}) \leq \mathcal{K}$ if

F_i and F_j are $\mathcal{K}$-compatible and, thus, in the constrained fault compatibility graph no edge weight is higher than $\mathcal{K}$. The same holds for the vertex weights, only if $\mathcal{K}$ is greater than or equal to S_{max}.

4.3 Match-And-Merge Algorithm

The first dynamic method is essentially a two phase match-and-merge procedure, employing the constrained compatibility graph described in Section 4.2. The ideal solution for this problem need to have complete information of (i.e. have all the tests targeting) both each fault considered and all the possible combinations between faults. If this information was available then the optimal solution can be obtained by finding a set of fault combinations that gives the overall minimum number of specified bits and, at the same time, cover all the considered faults. The latter can be reduced to the infamous *set covering problem* which is known to be an NP-complete problem [58], i.e., has no polynomial time solution. This is in addition to the test generation problem that is necessary for obtaining the complete information about the tests described earlier. Specifically, it has been shown that the general problem of test generation for digital circuits is NP-hard [59]. Hence, this section, as well as the next one, propose heuristic algorithms for obtaining test sets with a large number of unspecified bits using the constrained fault compatibility graph described in Section 4.2.

First, pairs of faults are identified and merged (via vertex merging) based on a minimum-weight-neighbor criterion. The algorithm takes a decision on which neighbor to choose based on local information, yet the minimization goal is global. The algorithm terminates when no edge is left on the graph and the final vertices correspond to tests that detect all the modeled faults considered. Initially each vertex v_i in the graph corresponds to a fault f_i in the fault list, under the fault model considered. The edges between vertices are placed according to the $\mathcal{K}$ constraint which is upper bounded by a user defined threshold S_h. The value of parameter S_h is important to the performance of the method since it directly impacts the maximum number of specified bits allowed per generated test. The size of S_h can be experimentally determined in a systematic manner.

In the first phase, the algorithm identifies pairs of compatible faults that give the minimum testing cost in terms of specified bits. We formulate this problem as a modified min-cost max-matching; instead of focusing on minimizing the overall weight (sum of all node weights in the graph), as in the traditional min-cost max-matching problem, we concentrate on max matching with individual node weights not greater than S_h. This procedure is applied iteratively, with pair-wise matching per iteration. The solution to this problem will give us a maximal merging of the faults (max–matching) with a minimal number of specified bits in each generated test.

To solve this problem, we use a Path Growing Algorithm for the weighted matching problem, similar to the one proposed in [60]. The algorithm is linear to the size of the input graph and guarantees a performance ratio of 1/2. More sophisticated algorithms could always be used, at the expense of extra complexity in both implementation and running cost. Initially, an arbitrary vertex v_1 is selected and a growing path P_{v_1} is constructed by following the minimum weight edge e_{min}. All other edges adjacent to the current vertex are dropped from consideration. The other endpoint of the selected edge (e_{min}) becomes the current vertex, and the path is grown in the same manner, until the path can grow no further. Then, an unvisited vertex v_2 is selected to be the starting vertex of a new path. The algorithm terminates when no more edges exist. In each iteration, two edge lists are maintained. The graph edges are inserted in the appropriate list so that no adjacent edges are in the same list. The list with the minimum size is a valid matching for the graph examined.

Once the weighted matching algorithm terminates, the match-and-merge procedure enters its second phase, i.e. the one performing the vertex merging. For each edge e_i in the matching obtained, the algorithm merges its adjacent vertices v_1 and v_2 to form a new vertex $v_{1,2}$. Merging, essentially, refers to the union of the corresponding set of faults (F_1 and F_2) and, consequently, to the intersection of the corresponding set of tests ($\mathcal{T}_1$ and $\mathcal{T}_2$). In this way $v_{(1,2)}$ corresponds to a new set of faults containing all faults in both F_1 and F_2 (i.e., $F_1 \cup F_2 = F_{1,2}$). Each one of the faults in $F_{1,2}$ can be detected by any test that belongs to both $\mathcal{T}_1$ and $\mathcal{T}_2$, (i.e., in $\mathcal{T}_1 \cap \mathcal{T}_2 = \mathcal{T}_{1,2}$). In the next iteration of the algorithm a new constraint compatibility graph is constructed using the merged vertices. The weights on the vertices are computed again using test generation for the corresponding set of faults, in order to determine the set of tests $\mathcal{T}_{i,j}$, for each vertex $v_{i,j}$.

Lemma 4.1. The weight on a vertex $v_{i,j}$ emerged after the merging of two vertices v_i and v_j, denoted by $w(v_{i,j})$ always falls within the following bounds:

$$\max(w(v_i), w(v_j)) \leq \; w(v_{i,j}) \; \leq \; w(v_i) + w(v_j)$$

Proof. Both bounds are determined by the intersection operation performed in the set of tests corresponding to v_i and v_j in order to obtain the merged vertex $v_{i,j}$. Recall that, the weight on a vertex is essentially the minimum number of specified bits per test among all tests that detect the corresponding faults. Since $\mathcal{T}_{i,j}$ consists only of tests that are in both $\mathcal{T}_i$ and $\mathcal{T}_j$, then the test with the minimum number of specified bits in $\mathcal{T}_{i,j}$ is the test with the minimum number of specified bits either in $\mathcal{T}_i$ or in $\mathcal{T}_j$. From these two tests only the one with the maximum number of specified bits can be in both $\mathcal{T}_i$ and $\mathcal{T}_j$. Thus, $\max(w(v_i), w(v_j)) \leq \; w(v_{i,j})$. Furthermore, the test with the minimum number of specified bits in $\mathcal{T}_{i,j}$ cannot be more than the sum of the number of specified bits in the minimum test of $\mathcal{T}_i$ and $\mathcal{T}_j$. This happens because a test having specified bits in the same positions as the minimum test in $\mathcal{T}_i$ and the minimum test in $\mathcal{T}_j$ together, is an element of both $\mathcal{T}_i$ and $\mathcal{T}_j$ and, thus, an element of $\mathcal{T}_{i,j}$. Such test always exists since sets $\mathcal{F}_i$ and $\mathcal{F}_j$ are compatible and, so it holds that $w(v_{i,j}) \; \leq \; w(v_i) + w(v_j)$. □

This match-and-merge algorithm iterates until all vertices in the constrained compatibility graph are fully disjoint, indicating that no further matching/merging can be done for the given value of S_h.

An example of the iterative match-and-merge procedure is shown in Figure 4.1. Figure 4.1(b) shows the initial weighted constraint compatibility graph corresponding to the fault list in Figure 4.1(a). For simplicity, the graph vertices are denoted by their corresponding sets of faults. Let us set $S_h = 32$, so that we consider a 32-compatibility constraint in the graph. The procedure starts from the randomly selected vertex F_2 and selects the edge (F_2, F_1) which is placed in the first edge list, let that be denoted by L_1 =$\{(F_2, F_1)\}$. All other edges adjacent to F_2 are dropped from further consideration. In the next step the edge (F_2, F_3) is selected and inserted in the second list, i.e. L_2 =$\{(F_2, F_3)\}$. The path cannot grow any more so a new vertex is selected; let that be F_6, which is matched with F_4 and (F_4,F_6) is placed in L_1. At this point the matching phase of the algorithm ends. The maximum matching is given in L_1 containing the edges (F_1, F_2) and (F_4, F_6) which are merged and become the graph's vertices in the next iteration (second phase). All vertices not considered in the matching are also included in the new graph. The starting graph in the second iteration is shown in Figure 4.1(d) and the corresponding fault list in Figure 4.1(c). Note, that the test generation

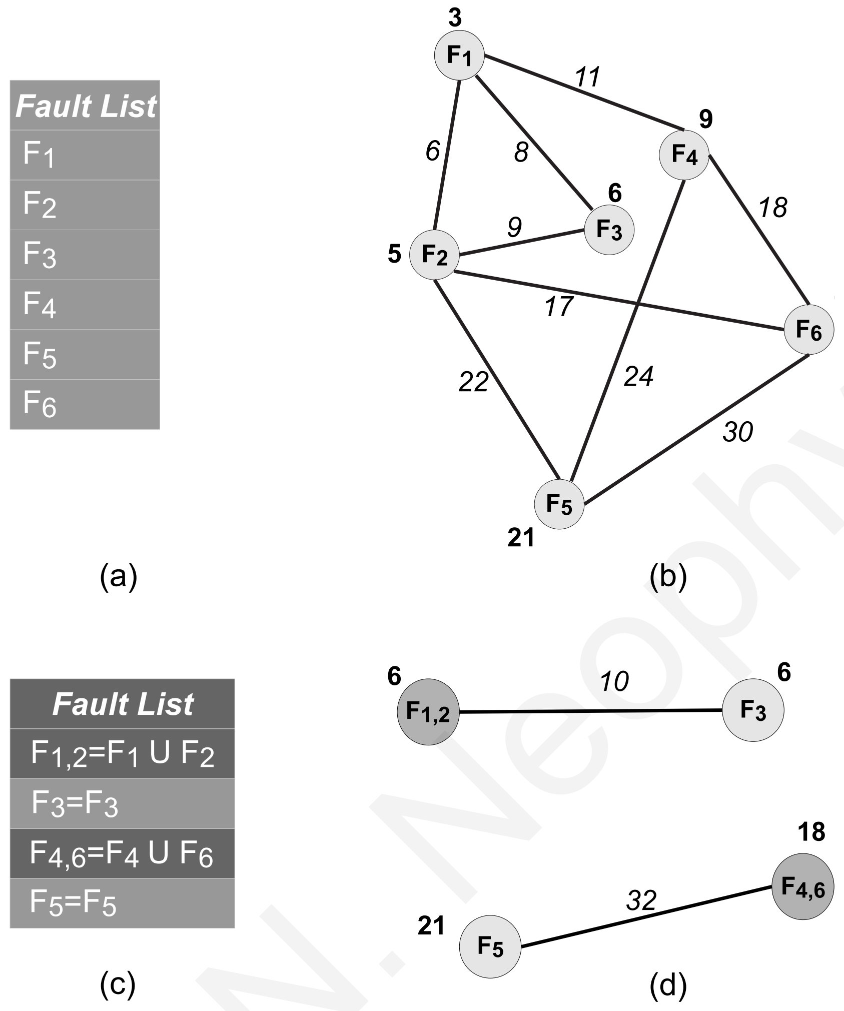


Figure 4.1: An iteration of the match-and-merge algorithm

procedure was used in order to calculate the new weight on the graph's edges. The weights on the graph's vertices are equal to the weight of the corresponding edges, before the merging phase. The second iteration is the final one and matches vertex $F_{1,2}$ with F_3 and vertex $F_{4,6}$ with F_5. Thus, tests are generated for the sets of faults $F_{1,2,3}$=$F_{1,2} \cup F_3$ and $F_{4,5,6}$ $=$$F_{4,6}$$\cup$$F_5$ with as few specified bits as possible.

4.4 Hierarchical Fault Compatibility Identification Algorithm

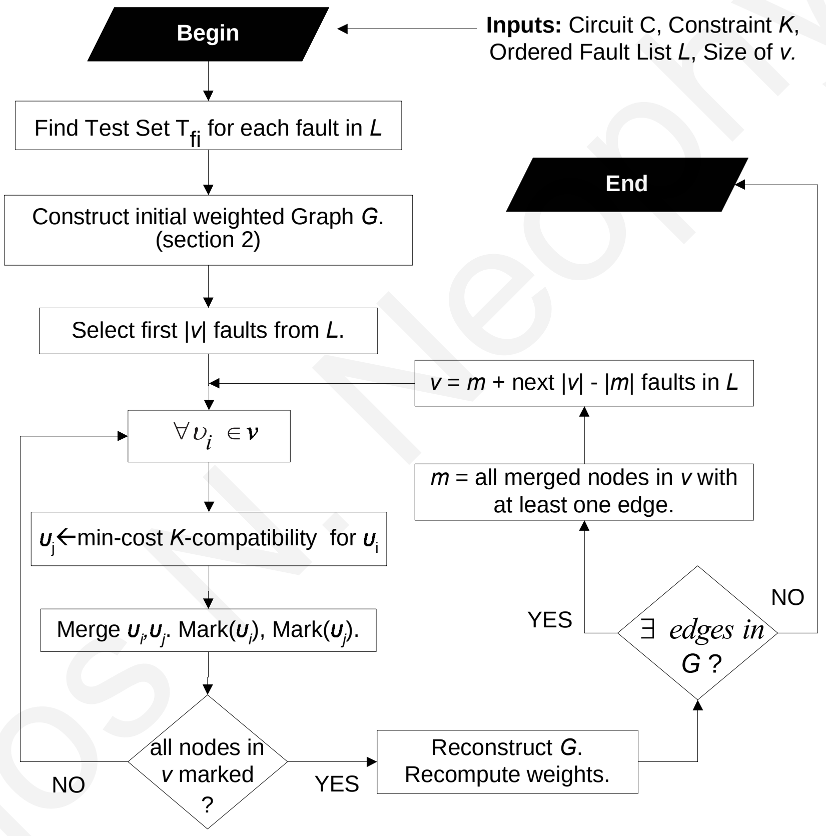
The second method that we propose in this chapter is essentially an improvement over the algorithm of Section 4.3. It generates the desired test set by appropriate manipulations and modifications on a constraint compatibility graph that corresponds to a given targeted fault list. Essentially, the matching phase in this second dynamic method is totally different. The decision is taken on a more localized rationale, considering only a small part of the graph at each iteration. This makes the method more efficient and, at the same time, applicable to large circuits for which the first dynamic method blows up.

The flowchart of the algorithm of the method is given in Figure 4.2. The inputs are the circuit-under-test C, an ordered fault list F and a value for constraint $\mathcal{K}$. The ordering of the faults in F can be significant to the overall performance of the method. Section 4.6 elaborates further on this issue, presenting and evaluating various ordering methods.

The algorithm begins by constructing the corresponding test sets per fault in F, and the constrained fault compatibility graph G, as given in Section 4.2. Then, it enters a sequence of iterations, where at each iteration a set of vertices in G, let that be denoted by ν, is examined for minimum-cost $\mathcal{K}$-compatibility. At the end of each iteration, the pairs of $\mathcal{K}$-compatible vertices are merged into one and the vertex and edge weights in G are recomputed, based on the new vertices in G. The algorithm is hierarchical in the sense that it does not examine all the vertices in G for min-cost $\mathcal{K}$-compatibility at once. Instead, it selects a small number of vertices (faults) and attempts to find the best way to pairwise merge (detect) them with some other vertex in G, such that the overall cost on the number of specified bits is minimized. Merging of two vertices corresponds to finding the set of tests that detect all faults represented by the two vertices. As explained in Section 4.2, $\mathcal{K}$-compatible vertices can always be merged. Next we discuss min-cost $\mathcal{K}$-compatibility. Consider a vertex $\upsilon_i \in \nu \subset V$. Vertex υ_i is $\mathcal{K}$-compatible with all vertices in G that are connected to υ_i via an edge. The minimum cost incurred to merge vertex υ_i with one of its $\mathcal{K}$-compatible neighbors, referred to as min-cost $\mathcal{K}$-compatibility, is given by:

$$\min_j \left\{ W(e_{ij}) + \sum_{e_{ik}\in E, k\neq j} w(\upsilon_k) \right\}$$

The idea behind min-cost $\mathcal{K}$-compatibility is that fault (or faults) corresponding to some vertex υ_i is selected to be detected with the fault(s) of some vertex υ_j by a single test such that the number of specified bits in this test ($W(e_{ij})$) plus the number of specified bits in the remaining $\mathcal{K}$-compatible vertices of υ_i is minimized.



Begin
Inputs: Circuit C, Constraint K, Ordered Fault List L, Size of v.
Find Test Set T_fi for each fault in L
Construct initial weighted Graph G. (section 2)
Select first |v| faults from L.
∀υ_i ∈ ν
u_j←min-cost K-compatibility for u_i
Merge u_i,u_j. Mark(u_i), Mark(u_j).
all nodes in v marked ?
NO
YES
Reconstruct G. Recompute weights.
∃ edges in G ?
YES
NO
m = all merged nodes in v with at least one edge.
v = m + next |v| - |m| faults in L
End

Figure 4.2: Flowchart of proposed dynamic methodology.

After the min-cost $\mathcal{K}$-compatibility is computed for all vertices in ν, G is reconstructed to reflect the new merged vertices as well as the new vertex and edge weights. In the next iteration the set ν includes the vertices merged in the previous iteration which have at least one neighbor. For every such vertex with no neighbors, a new vertex that corresponds to the next fault in F is added to ν. Thus $|\nu|$ is kept constant among iterations. The algorithm terminates when there are no more edges in the new G which indicates no $\mathcal{K}$-compatibility and, thus, no test that can detect all faults represented by two vertices of G and have at most $\mathcal{K}$ specified bits.

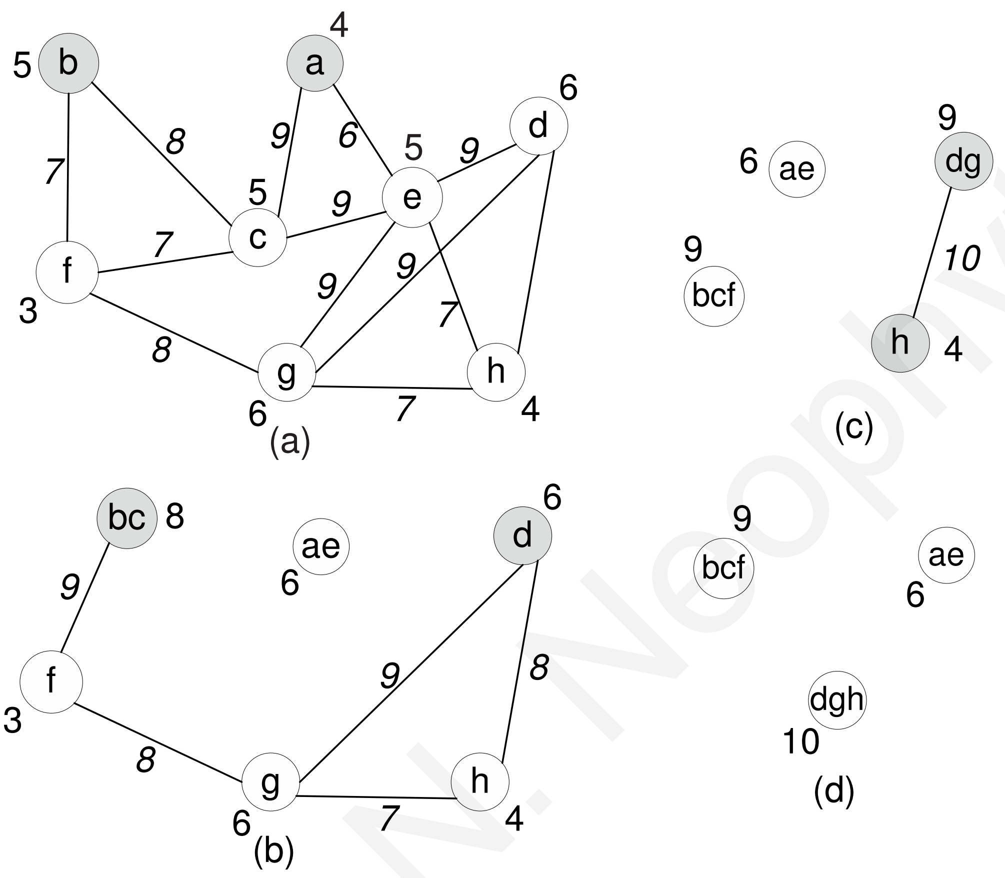


Figure 4.3: Hierarchical fault merging example.

Figure 4.3 gives an illustrative example. Consider an ordered fault list $F = \{a < b < c < d < e < f < g < h\}$, $\mathcal{K} = 12$ and $|\nu| = 2$. Figure 4.3a shows the initial graph G (first iteration) with some assumed vertex and edge weights. Here each vertex corresponds to a unique fault in F and no edge or vertex weight is greater than $\mathcal{K}$. During the first iteration $\nu = \{a, b\}$. Vertex a is $\mathcal{K}$-compatible with vertices c and e. The min-cost $\mathcal{K}$-compatibility for a is 11, that is 6 for the edge (a, e) plus 5 for the cost of vertex c. Thus, vertex e is selected to be merged with vertex a which implies that faults a and e can be tested by a single test with at most 6 specified bits. The other choice here would have been merging a with c which, however, gives a higher cost of 14. In a similar manner, vertex b is selected to be merged with vertex c with a cost of 11. Next, graph G is reconstructed to reflect the new merged vertices and corresponding weights, as shown in Figure 4.3b. Observe that only vertex and edge weights involving the merged vertices need to be recomputed. In the next iteration, only the merged vertex bc remains in ν, since it is still connected. The unconnected vertex ae implies that no more $\mathcal{K}$-compatible vertices exists for this vertex, thus will contribute with 6 specified bits in the final test set. Vertex d is the next in the order in F (c is part of a merged vertex, already in ν), so it is added to ν. Vertices bc and f are merged since no other option is left for the two vertices, during the second iteration. During this iteration vertices d and g are merged together (cost=13 over 14 for merging d with h) as illustrated in Figure 4.3c. The previously merged vertex bcf is removed from ν and h is added. The algorithm terminates after the fourth iteration with three disconnected vertices, ae, bcf and dgh (Figure 4.3d). Thus, the resulting test set will contain three tests with a total number of 25 specified bits. Observe that if we want to explicitly detect each fault with a dedicated test, the total cost in specified bits is 38, as it can be concluded by adding all the vertex weights in Figure 4.3(a).

In order to demonstrate the effectiveness of the proposed dynamic method and give some theoretical bounds on the execution time of the algorithm, we make the following statements.

Theorem 4.1. *If two vertices, υ_x and υ_y of the fault compatibility graph corresponds to faults (or group of faults) that are $\mathcal{K}$-compatible, then* $\max\{w(\upsilon_x), w(\upsilon_y)\} \leq W(e_{xy}) \leq min\{\mathcal{K},\ w(\upsilon_x) + w(\upsilon_y)\}$.

Proof. Let us first denote by t_i^{min} the test in T_i that has the minimum number of specified bits among all tests in T_i. From definition $w(\upsilon_x) = \min\{sp(t) \mid t \in T_x\} = sp(t_x^{min})$ and $w(\upsilon_y) = \min\{sp(t) \mid t \in T_y\} = sp(t_y^{min})$, where T_x and T_y are the sets of tests detecting the faults (or group of faults) corresponding to vertices, υ_x and υ_y, respectively. Since, the weight on an edge of the fault compatibility graph corresponds to the specified bits of a new test that detects all faults that corresponds to both the merged vertices, this test must come from the intersection of the corresponding sets of tests. In the case considered here, $W(e_{xy}) = \min\{sp(t) \mid t \in T_x \cap T_y\}$. Clearly, this new test cannot have less specified bits than either t_x^{min} or t_y^{min}. More precisely, it cannot have fewer specified bits than the maximum of these two tests since, since only the maximum of t_x^{min} and t_y^{min} can be common in T_x and T_y, since t_x^{min} and t_y^{min} are both minimum, in terms of specified bits, for each set. Thus, $\max\{w(\upsilon_x), w(\upsilon_y)\} \leq W(e_{xy})$. In order to prove the upper bound of this theorem, we consider the case where the two minimum tests (i.e. t_x^{min} and t_y^{min}) have no common bits that have a specified logic value. In this case the new test consists of all the specified bits of t_x^{min} and all the specified bits of t_y^{min}. Stated differently, since this new test is minimum in the intersection of T_x and T_y, it cannot have more than $sp(t_x^{min}) + sp(t_y^{min})$ since this test exist in the intersection (no conflicting bits) and combining the bits in both t_x^{min} and t_y^{min} gives a test that detects all faults previously detected by each of the two tests. Thus, $W(e_{xy}) \leq w(\upsilon_x) + w(\upsilon_y)$. Furthermore, by hypothesis the two tests are $\mathcal{K}$-compatible, and so $W(e_{xy})$ cannot be greater than $\mathcal{K}$. Thus, only if $\mathcal{K}$ is greater than $sp(t_x^{min}) + sp(t_y^{min})$ the proven upper bound holds. This implies that $W(e_{xy}) \leq min\{\mathcal{K},\ w(\upsilon_x) + w(\upsilon_y)\}$. □

Lemma 4.2. The number of iterations I that the algorithm can perform is bounded by: $0 \leq I \leq |F| - 1$, where $|F|$ is the number of faults considered.

Proof. The *lower bound* indicates that it is possible not to enter any iteration, in the case where no two faults in F can be detected by a single test. The *upper bound* is met when at every iteration, the number of merged vertices is exactly one. □

Definition 4.2. For one iteration, we refer to the ratio of the number of edges selected (indicates selection of vertices/faults to be detected by a single test) to the maximum of the edges that could be selected as *Merging Efficiency,* abbreviated as ME.

Lemma 4.3. The range of ME per iteration is given by $\frac{1}{2} \leq ME \leq 1$.

Proof. The *upper bound* is when at every iteration all vertices in ν are merged with vertices outside ν, resulting in merging the maximum number of vertices at that iteration. The *lower bound* comes when no vertex in ν is merged with a vertex outside ν, i.e. when $\frac{|\nu|}{2}$ merged pairs consist only of vertices in ν. □

The proposed algorithm guarantees that, after merging, the produced set of tests contains at least one test that is of less specified bits than the tests in the merged sets. Actually, from Theorem 4.1, it is guaranteed that if we are able to find the minimum test for detecting a fault or a group of faults, vertex merging will not increase the number of specified bit of the underlying test set, while the test set size is systematically reduced. In the worst case vertex merging gives a test set with the same number of specified bits, and with fewer tests. Experimental results in Section 4.6, show that the total number of specified bits in the test set is also reduced because of the vertex merging. Further experimentation shows that the upper bound of Theorem 4.1 is rarely met, especially in the first iterations of the algorithm where the tests have very few specified bits. Moreover, from Lemma 4.2 we conclude that the number of iterations are bounded. In practice, as our experimental results demonstrate, the number of iterations decreases as $|\nu|$ increases, at the expense of more computation per iteration. Another important attribute for the proposed algorithm is the number of vertices merged at each iteration. In Definition 4.2 we have defined this attribute as the merging efficiency. This number gives the rate by which our algorithm approaches the final solution. In most cases the higher this rate is, the better the solution is. The latter is because ideally our approach should expand each test as much as possible by combining it with as much compatible faults as possible, before the corresponding fault is removed from set ν. If two faults inside ν are merged together, then this expansion is constrained to one of the faults. This can limit the efficiency of this dynamic method, especially when the fault ordering considered, gives priority to the easy-to-detect faults.

4.5 Implementation Overview

While the methods described in Chapter 3 and Chapter 4 could be implemented using any test generation and fault simulation procedures, for the purpose of this dissertation, we have implemented all methods using a function-based framework. In this section we give some details for this implementation, as well as some preliminaries on function-based implementation. We choose a function-based implementation for the test generation and the fault simulation procedures because they provide an easy way to identify various desirable characteristics of a test or a group of tests. Specifically, they provide an efficient way of finding a minimum specified bits test among a set of tests, as well as an elegant systematic way of identifying compatibility and $\mathcal{K}$-compatibility between two or more tests.

4.5.1 Function-Based Test Generation and Fault Simulation

For all test generation and fault simulation proposed in this paper we have used a function-based framework. Although any of the previously proposed function-based frameworks can be used, in this work we have used an in-house tool that uses Binary Decision Diagrams (BDDs) [34].

Test generation is performed by XORing the fault free functionality of the circuit with the functionality of the circuit after the considered fault is injected. For the results obtained we have used the stuck-at fault model. Given one or more faults, the corresponding test function *implicitly represents all the test patterns detecting the fault(s)* [59]. Essentially, every minterm of this function is a valid test for the fault(s). By considering a large number of tests for each fault a great advantage over structure-based techniques, is obtained. That is because the large number of tests gives more flexibility in selecting the appropriate one, based on the targeted application. This advantage is more important in the dynamic method where no initial test restriction exists and, thus, flexibility in the test selection process is highly desired. For the test generation in all methods proposed in this work, we extract a test function for each targeted fault in the collapsed stuck-at fault list of the circuit. Fault collapsing in this phase is done using the static arguments of the Checkpoint Theorem [59]. Then, functional fault equivalence and functional fault dominance are applied to further reduce the number of faults that need to be considered.

Fault simulation is performed by checking if a test cube corresponding to a given test is contained in the function generated for each fault considered. Containment here implies that the cube consists only of minterms of the function examined and can be justified if the logic ANDing between the test cube and the function corresponding to the fault gives a non-zero function.

The process of *retrieving a test with a large number of unspecified bits* from a test function is identical to the process of finding a large cube in that function. If the largest cube is selected, then a test is found which is guaranteed to have the fewer specified bits among all tests detecting the corresponding fault or group of faults. While the process of identifying the largest cube in a generic test function is a hard process, it can be done efficiently when BDDs are used. Because of their canonical form, the largest cube can be found by following the shortest path in the diagram which is a linear time operation, with respect to the size of the BDD. Obtaining the shortest path is done with respect to the BDD variable ordering and, thus, it does not guarantee that it coincides with the function's actual largest cube. Improvements can be achieved using a heuristic described in the next subsection.

Fault compatibility, which is necessary for the dynamic method of Chapter 4, can be examined by ANDing the test functions corresponding to the faults checked for compatibility. The result of the AND operation gives a test function whose each minterm can detect all the faults considered, at the same time. If this function is zero then the faults are not compatible; otherwise, faults can be tested together and the ANDing operation corresponds to the merging operation of the method described in Chapter 4. Consequently, the test function resulting from this AND operation, implicitly holds all the tests that can detect all the merged faults. In order to determine whether two faults meet the $\mathcal{K}$-compatibility constraint, a largest cube query on the merged test function is performed, as explained in the previous paragraph. If the size of the largest cube is lower than, or equal to, $\mathcal{K}$ then the two faults are $\mathcal{K}$-compatible.

Using BDDs for representing test functions may lead to high memory requirements, and, consequently may be an issue for the scalability of the methods using BDDs. Both the proposed methods essentially consist of two parts regarding the usage of the BDDs. The first part is a pre-processing step that generates a test function for each fault considered. The second part includes all the BDD-related operations of the methods. For the proposed methods, the second part does not affect scalability, since the proposed methods perform BDD operations just for obtaining temporal information used for locally taken decisions. The BDDs constructed for this reason are released after the decision is made, and, thus memory is not accumulated. The first part (i.e. generating a test function for each fault considered) is the part that can give undesired memory increase. Stated differently, this is the only part of the algorithm that can make the algorithm implementation blow-up and, thus, hurt the scalability of the proposed methods. Based on our experience on the ISCAS benchmark circuit, this happens mainly due to the complexity of a circuit and not proportional to its size. It is well known that out of all ISCAS circuits, only c6288 from the ISCAS'85 suite cannot be represented using BDDs, despite the fact that other circuits with much larger size and much more paths can be represented.

A number of solutions can be used for handling more complex and/or larger circuits like modern industrial circuits:

• Instead of building a BDD for the entire test function for each fault, construct a number of simplified BDDs for each fault by systematically fixing a small number of the circuit's primary inputs. The inputs to be fixed should be selected so that they affect as small portion of the circuit as possible. Different BDDs are generated for all possible value combinations for these inputs. Appropriate merging of these BDDs gives a reduced size diagram which, however, has smaller number of tests and, thus, smaller test space. Naturally, this affects the obtained test set that may have more specified bits (sub-optimal solution). This approach has been used in [61] and manages to handle even the problematic benchmark c6288. Naturally, this solution gives partial information and, thus, the decisions taken at different steps of the proposed methods are not the best possible. Moreover, this rationale can be extended in generating test functions at the input-output fault cones, and, hence, drastically reduce the memory requirements of the methods.

• Constraint versions of the BDDs can be used in order to avoid blowing up. In [50] a framework is proposed, where each test function is partitioned into k different partitions so that each partition is represented by a different BDD. This way the total memory requirements are reduced canceling any scalability problem that may occur. These new constraint BDDs are called Partitioned BDDs and have been used in design verification [50, 62] and show great reduction in the BDD size. The framework, also, proposes procedures for efficient manipulation of Partitioned BDDs. Yet, the efficiency of this solution should be verified, with respect to the final test sets obtained by the two methods.

• Using heuristic techniques for the Boolean Satisfiability problem (SAT-solvers) in conjunction with the BDD structures in order to reduce the BDD size. A number of recent work (see [63, 64] among others) has been proposed which describe some very fast implementations of SAT-solvers, based on the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [65]. Previous work on using SAT-solvers for test generation (e.g. the work in [66, 67]) as well as in a number of design automation problems [68]. Essentially, SAT-solvers can be used together with BDDs in cases where BDDs blows-up in order to give a sub-optimal solution for the corresponding function or can be used as a standalone solution in order to completely replace BDDs for function representation and manipulation. This approach has been widely used for Symbolic Model Checking ([69, 70, 71] among others) and shows to drastically reduce the process response time.

4.5.2 An Optimization Heuristic for Obtaining a Larger Cube from a BDD

We now present an optimization heuristic to further reduce the size of a largest cube in a BDD-based function. This is a post-processing step, that can be optionally applied. The size of the largest cube in a BDD depends on the BDD variable order. Thus, theoretically, the largest cube derived from a BDD can be further enlarged. The heuristic tries to further enlarge the largest cube LC_i of a BDD-based test function TF_i by appropriately relaxing fixed variables in LC_i. It iterates on the variables in the support of LC_i and performs Existential Abstraction on LC_i with respect to each variable, i.e. removes the impact of the variable from the corresponding function TF_i. The resulting cube is then checked against function TF_i for satisfiability. If the resulting cube satisfies TF_i the new cube is set as the largest cube and the heuristic continues with the next variable, until all variables are considered. The operations of existential abstraction and satisfiability are standard BDD operations and can be done very efficiently due to the canonical structure of the BDDs. Figure 4.4 shows the pseudocode of the heuristic using standard BDD operations.

procedure **identify_extra_xs( )**

```
INPUT:    function $TF_i$
OUTPUT: largest cube $LC_i$ of function $TF_i$,
 1: $LC_{temp}$=$LC_i$=Shortest_Path($TF_i$)
 2: $S$=Support($TF_i$)
 3: for each variable $v \in S$
 4:        $LC_{new}$ = Existential_Abstract($LC_{temp}$)$_{|v}$
 5:        if ( $TF_i$($LC_{new}$) = 1 ) then
 6:              $LC_{temp} = LC_{new}$
 7: return $LC_i$=$LC_{temp}$
```

Figure 4.4: Identifying additional unspecified bits per test in BDD-based implementations.

Next, we give some measurements that demonstrate the impact of this optimization procedure. Table 4.1 shows this information for an indicative number of circuits when this optimization is used together with the dynamic method of Section 4.4. Column 2 lists the average gain (percentage against number of primary inputs) that can be obtained each time the optimization is called. This gain is considerable for certain cases. For example, C880 has 60 inputs. An average gain of 6.75% leads to approximately 4 new unspecified bits per test, on the average. Columns 3 and 4 list the number of tests and the number of specified bits in the test set, when no optimization was activated. Columns 5 and 6 list similar information, when the optimization was applied every time a largest cube query in the BDD was performed. Columns 7 and 8 report the optimization's gain when the optimization is applied only once, after the algorithm terminates and the test set has been derived. These measurements show that applying the optimization just once at the end of the algorithm gives a noticeable benefit on the total number of specified bits. Moreover, applying the optimization throughout the algorithm's execution does not always lead to a smaller number of specified bits. This occurs since modifying the number of specified bits per vertex in the compatibility graph can guide the fault merging algorithm to a different direction. The behavior of the optimization is analogous for all the static and dynamic methods proposed in this dissertation.

Table 4.1: Impact of the cube optimization heuristic

Circuit	**Avg. Gain**	**No optimization**		**Always Activated**		**Activated at the End**	
	(% PIs)	**Test Func.**	**Sp. Bits**	**Test Func.**	**Sp. Bits**	**Test Func.**	**Sp. Bits**
c880	6.75	20	815	21	764	20	734
c1355	8.02	84	3120	84	2844	84	2985
c1908	2.72	107	1762	107	1701	107	1666
c2670	1.15	56	2945	55	2798	56	2834
c3540	3.08	100	2159	101	2075	100	2005
c5315	1.11	51	3106	52	3145	51	3005
c7552	0.59	78	6140	78	6044	78	6044
s953	4.33	78	1124	79	1003	78	972
s1196	3.23	118	1740	117	1634	118	1618
s1238	4.21	123	1875	123	1709	123	1709
s1423	4.22	25	1341	24	1186	25	1245
s1494	7.06	102	1171	103	1081	102	1063
s9234	0.02	145	8399	147	8419	145	8367
s13207	1.17	267	12173	267	11480	267	11596

4.6 Experimental Results for the two Dynamic Methods

The proposed methods were implemented in C language and run on a SunBlade 1500 machine, running Solaris with 4GB of RAM. We experimented with the ISCAS'85 and the full-scan versions of the ISCAS'89 benchmark circuits. All the test generation related procedures were implemented using the framework described in Section 4.5.

In Table 4.2 we show results for the first dynamic method proposed in this chapter. Column

Table 4.2: Results for the match-and-merge method

Circuit	PIs	Faults	Tests	S_{max}	Total Bits	Sp. Bits	% Sp. Bits	# Iter.	CPU (s)
c880	60	895	76	43	4560	2266	49.7	8	18.1
c1355	41	1536	200	40	8200	6844	83.5	9	14.6
c1908	33	1981	161	31	5313	3778	71.1	9	10.6
c2670	233	2681	69	188	16077	7536	46.9	10	75.7
c3540	50	3634	141	44	7050	3924	55.7	9	98.6
c5315	178	5623	N/A	N/A	N/A	N/A	N/A	N/A	N/A
c7552	207	7620	N/A	N/A	N/A	N/A	N/A	N/A	N/A
s510	25	551	70	11	1750	541	30.9	6	0.1
s713	54	514	125	26	6750	1453	21.5	6	0.1
s832	23	833	136	15	3128	1287	41.1	7	0.2
s953	45	1063	129	19	5805	1664	28.7	6	0.2
s1196	32	1334	182	18	5824	2577	44.2	8	0.8
s1238	32	1334	258	18	8256	2648	32.1	9	1.2
s1423	91	1532	212	20	19292	3233	16.8	7	2.0
s1494	14	1694	155	14	2170	1450	66.8	9	0.7
s9234	247	9402	N/A	N/A	N/A	N/A	N/A	N/A	N/A
s13207	700	13287	N/A	N/A	N/A	N/A	N/A	N/A	N/A
s15850	611	10278	N/A	N/A	N/A	N/A	N/A	N/A	N/A
s38417	1664	31183	N/A	N/A	N/A	N/A	N/A	N/A	N/A
s38584	1464	36301	N/A	N/A	N/A	N/A	N/A	N/A	N/A

2 lists the number of Primary Inputs and Column 3 lists the number of total faults in the collapsed fault list, per circuit. The collapsed fault list was realized after functional equivalence and structural dominance were applied to the given fault list. No faults were aborted. Column 4 gives the number of tests by the obtained method and Column 5 reports the maximum number of specified bits per test for each test set. Observe that the sizes of the test sets are large, which was expected, while the S_{max} attribute is much lower than the number of primary inputs, indicating a significant reduction for this measure. Columns 6 and 7 report the number of total test bits and the number of specified bits, respectively, while Column 8 gives the percentage of specified bits in the test set. The last two columns show the method's requirements in terms of iterations and CPU time (in secs) needed. The N/A indicates that the results for the corresponding circuit could not be obtained, since the BDD-based implementation could not handle the huge number of BDD operations. As we show next we

overcome this problem in the hierarchical method.

Table 4.3 gives our results for the hierarchical fault compatibility method. Column 2 lists the number of Primary Inputs and Column 3 lists the number of total faults in the collapsed fault list, per circuit. Column 4 reports the number of tests in the generated test set. The value for constraint $\mathcal{K}$ is given in Column 5 which is actually the S_{max} parameter per test set, i.e., the maximum $sp(t)$ among all tests in the set. The value of $\mathcal{K}$ ranges between the maximum largest cube for each individual fault and the number of primary inputs of the circuit. Columns 6 and 7 give the total number of bits and the number of specified bits per test set, respectively. The percentage of the specified bits with respect to the total bits in the resulting test sets is given in Column 8. As expected, circuits with a large number of primary inputs tend to have a small number of specified bits, whereas circuits with a small number of primary inputs do not allow for a large number of unspecified bits in compact test sets. Column 9 shows the cardinality of ν considered for each benchmark and Column 9 reports the CPU time for the proposed method in seconds.

In Table 4.4 we compare the two proposed approaches. To our knowledge no dynamic technique that explicitly targets the same problem exists and, thus, we compare with the best previously proposed static technique [48], for completeness. We intend to compare with application specific dynamic techniques in Chapter 5. Since we are interested in maintaining compact test sets, we compare with the results reported in [48] for compact test sets, which also include information on the S_{max} parameter.

The results of the proposed dynamic methods are given again in Table 4.4. Columns 2-5 and Columns 10-13 of the table show the results for the hierarchical method and the match-and-merge method, respectively. The experimental results reported in [48] are listed in Columns 6-9. The N/A value corresponds to non-reported circuits. In this comparison, the key measure is the total number of specified bits in the final test. We focus on the hierarchical fault compatibility method, since it shows to give better results in terms of absolute numbers of specified bits. Observe that in all cases, except for s1494, the hierarchical method reports a reduction in the total number of specified bits per test set (Columns 4 and 8), which is considerable in some cases such as for circuits c1908 and s13207. Also, despite the fact that the method of [48] uses very compact initial test sets (derived from a compaction tool), the proposed method was able to derive test sets with very comparable cardinality, with some exceptions (s9234, s38417). This shows that the hierarchical method was able to generate

Table 4.3: Test generation results of the hierarchical dynamic method.

Circuit	PIs	Faults	Tests	S_{max}	Total Bits	Sp. Bits	% Sp.Bits	$\|\nu\|$	CPU (s)
c880	60	895	20	55	1200	734	61.2	2	23.4
c1355	41	1536	84	41	3444	2844	82.6	1	18.6
c1908	33	1981	107	31	3531	1666	47.2	1	12.3
c2670	233	2681	55	149	12815	2798	21.8	2	72.6
c3540	50	3634	100	33	5000	2005	40.1	4	89.4
c5315	178	5623	51	127	9078	3005	33.1	4	101.5
c7552	207	7620	78	160	16146	6044	37.4	2	38.1
s510	25	551	56	11	1400	447	31.9	2	0.2
s713	54	514	23	33	1242	632	50.9	2	0.3
s832	23	833	127	16	2921	1056	36.2	1	0.3
s953	45	1063	78	22	3510	972	27.7	1	0.4
s1196	32	1334	115	19	3680	1599	43.5	4	0.9
s1238	32	1334	123	20	3936	1709	43.4	1	1.1
s1423	91	1532	24	80	2184	1186	54.3	2	1.9
s1494	14	1694	102	14	1428	1063	74.4	4	0.7
s9234	247	9402	145	205	35815	8367	23.4	4	5.3
s13207	700	13287	267	458	186900	11480	6.1	2	7.9
s15850	611	10278	160	541	97760	11871	12.1	4	21.2
s38417	1664	31183	88	1012	146432	31731	21.7	4	95.1
s38584	1464	36301	126	1314	184464	29768	16.1	4	121.9

tests with a large number of unspecified bits, and at the same time, maintain a small number of total tests. Based on this, we conclude that the highest relaxation achieved by the method does not come because we have not imposed any constraint on the size of the test set. Relaxation is not even inversely proportional to the size of the test set (see results for c1355 and c1908). On the contrary, the large percentage of unspecified bits is attributed mainly to the constraints enforced on the fault compatibility graph, which guide the algorithm in choosing groups of compatible faults to be detected by a single test with small number of specified bits. In all cases, the S_{max} parameter is lower than those reported in [48], in some cases considerably (e.g. s1238, s13207). A low value for the S_{max} parameter is desirable for many test encoding/embedding methods, such as in deterministic BIST with LFSR-reseeding, giving the proposed technique a clear advantage for such applications. Despite the fact that the method of [48] did not target the problem of test set embedding specifically, it is quite common for these techniques to start from highly compacted test set, in order to reduce the area of the embedding overhead. For this specific application, however, the results obtained by the match-and-merge method may be of greater interest. Observe that, in most cases, while the absolute number of specified bits is larger, the percentage of specified bits is lower, mainly because the test set sizes are too large. Moreover, the values for the S_{max} attribute is, in most of the cases, lower for the match-and-merge method. This occurs because the method of Section 4.3 gives priority in minimizing the S_{max} parameter in disfavor of the total number of tests. These two test set characteristic can become very desirable in test set embedding techniques using compression, like those in [72, 73, 74]. More importantly, the technique of Section 4.3 cannot handle large circuits for the problem considered. We investigate how these test sets can be used in test embedding techniques in Chapter 5.

Table 4.4: Comparison between existing static and the proposed dynamic methods.

	Hierarchical Method (Section 4.4)				**[48]**				**BIST-guided (Section 4.3)**			
Circuit	**Tests**	S_{max}	**Sp. Bits**	**% Sp. Bits**	**Tests**	S_{max}	**Sp. Bits**	**% Sp. Bits**	**Tests**	S_{max}	**Sp. Bits**	**% Sp. Bits**
c880	20	55	**734**	61.2	21	60	**827**	65.6	76	43	**2266**	49.7
c1355	84	41	**2844**	82.6	84	41	**3444**	100.0	200	40	**6844**	83.5
c1908	107	31	**1666**	47.2	106	33	**2903**	83.0	161	31	**3778**	71.1
c2670	55	149	**2798**	21.8	45	233	**3072**	29.3	69	188	**7536**	46.9
c3540	100	33	**2005**	40.1	93	50	**2172**	46.7	141	44	**3924**	55.7
c5315	51	127	**3005**	33.1	46	178	**3161**	38.6	N/A	N/A	**N/A**	N/A
c7552	78	160	**6044**	37.4	75	207	**7064**	45.5	N/A	N/A	**N/A**	N/A
s510	56	11	**447**	31.9	N/A	N/A	**N/A**	N/A	70.	11	**541**	30.9
s713	23	33	**632**	50.9	N/A	N/A	**N/A**	N/A	125	26	**1453**	21.5
s832	127	16	**1056**	36.2	N/A	N/A	**N/A**	N/A	136	15	**1287**	41.1
s953	78	22	**972**	27.7	N/A	N/A	**N/A**	N/A	129	19	**1664**	28.7
s1196	115	19	**1599**	43.5	N/A	N/A	**N/A**	N/A	182	18	**2577**	44.2
s1238	123	20	**1709**	43.4	125	32	**1740**	43.5	258	18	**2648**	32.1
s1423	24	80	**1186**	54.3	24	91	**1267**	58.0	212	20	**3233**	16.8
s1494	102	14	**1063**	74.4	100	14	**1016**	72.6	155	14	**1450**	66.8
s9234	145	205	**8367**	23.4	111	247	**8499**	31.0	N/A	N/A	**N/A**	N/A
s13207	267	458	**11480**	6.1	235	700	**13160**	8.0	N/A	N/A	**N/A**	N/A
s15850	160	541	**11871**	12.1	97	611	**13454**	22.7	N/A	N/A	**N/A**	N/A
s38417	88	1012	**31731**	21.7	84	1664	**35224**	25.2	N/A	N/A	**N/A**	N/A
s38584	126	1314	**29768**	16.1	114	1464	**31543**	18.9	N/A	N/A	**N/A**	N/A

In Figure 4.5 we show the specified bits distribution among the tests in the resulting test set, for the second dynamic method proposed (hierarchical). The distribution is similar to

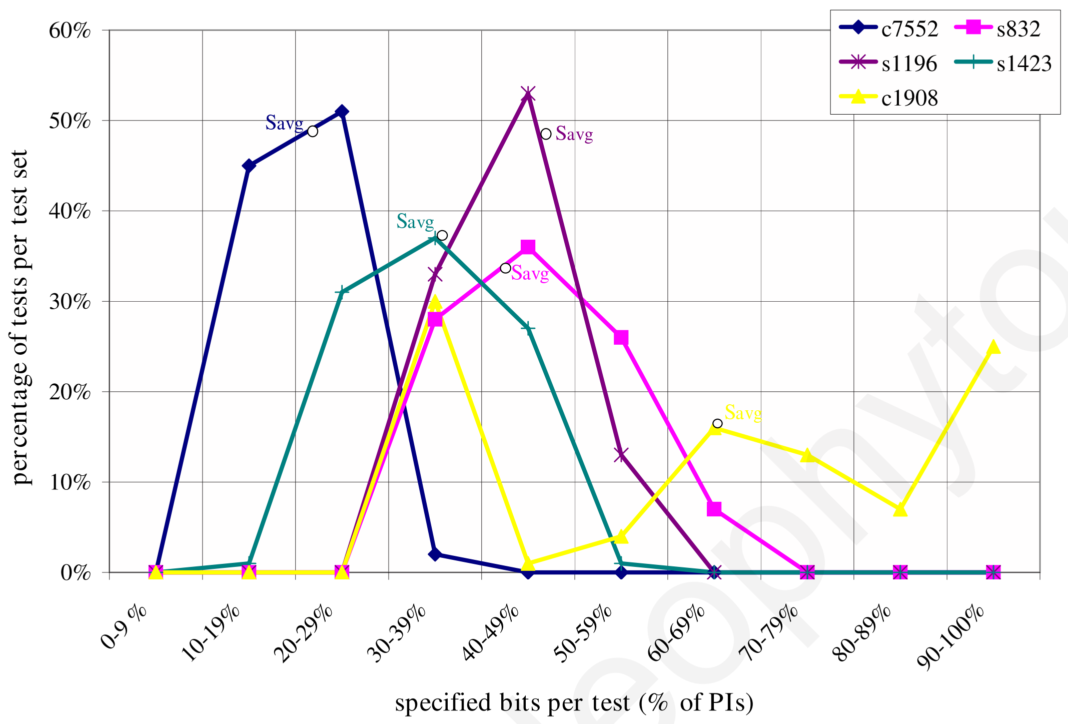


Figure 4.5: Specified bits distribution among tests after applying the hierarchical method

that of the static method (see Figure 3.3. Although the two methods may give different values for the S_{avg}/PI, they both give a large percentage of tests that have number of specified bits closed to S_{avg}. Observe, for instance, s1423. In the static method it gives 55% for the S_{avg}/PI. Approximately, 57% of the obtained tests have specified around 50-65% of the test pattern bits. Similarly, in the dynamic method, where the S_{avg}/PI is 35%, 71% of the test pattern have specified around 25-40% of each test pattern bits. The distribution of the specified bits, among the test patterns, follows this trend due to the constraint imposed to the test selection process that inherently exists in both methods. Furthermore, the dynamic method amplifies this trend, mainly because it is more flexible in selecting the tests to combine without considering given tests. For circuit c1908, the resulting test set obtained by this dynamic method does not follow the observed trend. This has to do with the structure of c1908. Since c1908 is an error detector/corrector circuit all of its primary inputs are used to generate a syndrome in order to identify the erroneous bit(s). This gives high corelation between the circuit's primary inputs, which eventually enforces large groups of the test patterns bits to form clusters of bits that influence each other. Within these clusters test bits cannot be left unspecified if one or more bits in the same cluster have been specified, and, thus the patterns with similar number of specified bits are clustered too giving two or more values of specified bits that correspond to a large number of tests.

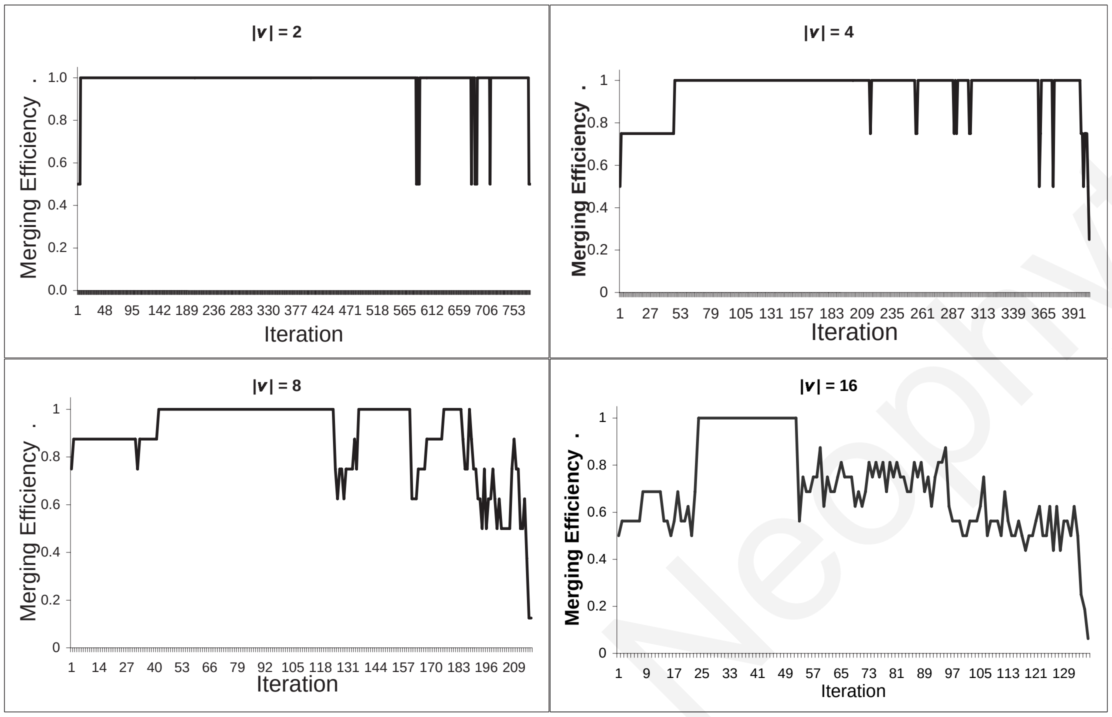


Figure 4.6: The effect of the size of ν in Merging Efficiency, for s1494.

The plots in Figure 4.6 show the merging efficiency (ME), as defined in Definition 4.2, per iteration for circuit s1494, for four different values of $|\nu|$, i.e. $|\nu| = 2, 4, 8, 16$. Observe that the bounds defined by Lemma 4.3 are clearly verified. The merging efficiency is almost always optimal for $|\nu| = 2$ and $|\nu| = 4$, and it drops for $|\nu| = 16$. Further experimentations show that it drops further as $|\nu|$ increases. This demonstrates the advantage of the hierarchical nature of the second dynamic method. It is preferable to perform more iterations, considering the merging of a small number of vertices in the constrained compatibility graph per iteration and, thus, solving an easier problem at a time. This approach gives better overall results (merging of vertices) than the case where a large portion or the entire compatibility graph (such as in the match-and-merge method) is considered at a time, since this is theoretically a harder problem. For $|\nu| = 1$, $ME = 1$ (optimal) since the single vertex in ν is always merged with a vertex outside ν.

Table 4.5 shows the number of tests and the number of total specified bits after applying the

Table 4.5: Number of specified bits for ν of different size, for s1494.

$\|\nu\|$	**Tests**	**Sp. Bits**	**Avg. ME**	**# Iterat.**
1	104	1134	1	1639
2	104	1116	0.99	829
4	102	1063	0.95	431
6	107	1136	0.92	298
8	108	1187	0.89	231
12	114	1162	0.84	163
16	116	1275	0.71	145
20	120	1228	0.69	120
30	121	1240	0.64	86
50	124	1262	0.59	56
100	129	1314	0.55	30
$\|V\|$	143	1445	0.54	8

algorithm of the hierarchical method using different values for $|\nu|$, for circuit s1494. The average merging efficiency and the number of iterations per run are also reported. In the last row, the case of not imposing any constraint on the size of ν, i.e., when ν equals the number of vertices on the graph, is reported. Observe how the Avg. ME drops as $|\nu|$ increases. Also, the number of iterations increases as $|\nu|$ decreases. In this case, the best results were obtained for $|\nu| = 4$, which has an Avg. ME close to the optimal (=0.95) and a considerably smaller number of iterations from the case of optimal Avg. ME (=1). The worst results were obtained for the case in the last row, where the entire compatibility graph is considered per iteration. As discussed in the previous paragraph, this occurs because the merging efficiency of the method is considerably higher when considering a small number of vertices at a time. The latter observation is justified by the results obtained for the match-and-merge method, where essentially ν is equal to the number of vertices in the graph (i.e., $\nu = |G|$). The main conclusion from this observation is that although $|\nu|$ can be user-defined and, thus, extensively explored to obtain the best results, this is not necessary in practice. The best results can be obtained by considering a small value for $|\nu|$.

Both algorithms of the dynamic methods consider an ordering on the vertices (faults) of

G, to guide the selection of the vertices in the graph, that are considered per iteration. We experimented with the five different vertex ordering methods, for the hierarchical method proposed:

Table 4.6: Effect of various vertex ordering methods in the method of Section 4.4

Circuit	(i)			(ii)			(iii)			(iv)			(v)		
	Tests	Sp. Bits	Avg. ME	Tests	Sp. Bits	Avg. ME	Tests	Sp. Bits	Avg. ME	Tests	Sp. Bits	Avg. ME	Tests	Sp. Bits	Avg. ME
c880	20	**734**	0.99	23	953	0.96	21	785	0.98	22	932	0.97	23	984	0.96
c1355	84	**2844**	0.99	88	3015	0.97	85	2901	0.97	87	2991	0.97	88	3016	0.96
c1908	107	**1666**	0.99	109	1911	0.96	106	1785	0.98	108	1896	0.96	109	1921	0.96
c2670	56	2875	0.98	58	2975	0.97	55	**2798**	0.98	57	2891	0.97	59	2981	0.96
c3540	100	**2005**	0.99	102	2101	0.98	100	2075	0.98	103	2198	0.98	103	2214	0.97
c5315	52	3087	0.97	55	3125	0.95	51	**3005**	0.97	56	3208	0.95	56	3212	0.95
c7552	78	6075	0.99	80	6232	0.96	78	**6044**	0.99	81	6209	0.96	81	6286	0.96
s953	78	**972**	0.97	91	1165	0.97	85	1007	0.98	92	1202	0.96	93	1311	0.95
s1196	118	**1618**	0.98	144	1978	0.97	115	1599	0.99	139	1895	0.97	145	2015	0.95
s1238	123	**1709**	0.98	153	2115	0.94	125	1761	0.97	153	1965	0.96	155	2231	0.94
s1423	24	1197	0.99	27	1301	0.97	24	**1186**	0.99	26	1253	0.97	29	1330	0.96
s1494	102	**1063**	0.99	117	1212	0.97	103	1088	0.98	118	1227	0.97	118	1241	0.94
s9234	145	**8367**	0.97	154	8985	0.96	146	8412	0.97	152	8912	0.96	159	8981	0.93
s13207	266	11512	0.96	271	12012	0.95	267	**11480**	0.96	270	11971	0.96	275	12154	0.92

i. Descending on $w\left(\upsilon_i\right)$ per vertex υ_i in the initial graph.

ii. Ascending on $w\left(\upsilon_i\right)$ per vertex υ_i in the initial graph.

iii. Ascending on the number of tests per vertex in the initial graph.

iv. Descending on the number of tests per vertex in the initial graph.

v. Random.

For example, considering the ordering of (i) the vertices with the largest weight will be included first in ν, during the initial iterations. Once a vertex cannot be further merged (to cover additional faults), the vertex with the next largest weight is included in ν. The ordering of the vertices is a preprocessing step which is performed once and is considered throughout the algorithm's execution. A vertex weight indicates how many specified bits will be needed to cover a vertex (detect one or more faults), whereas the number of tests at a vertex indicates the merging difficulty of the corresponding faults. Clearly, methods (iii) and (iv) only apply to implementations that can efficiently give all the tests for a fault or a group of faults, like function-based implementations (see Section 4.5).

Table 4.6 lists the number of tests, the total number of specified bits, and the merging efficiency (ME), for each one of the five orderings for the hierarchical method. The best results are obtained when the ordering is descending on the node weights (i) or ascending on the number of tests per fault (iii). In both of these cases, the algorithm starts by merging the nodes that correspond to more "difficult" faults. The difficulty here represents the probability of a node to be merged at a future iteration of the algorithm and is attributed either to a violation of the $\mathcal{K}$-compatibility property or due to the small number of possible tests. Merging the difficult nodes first, ensures no isolation of those nodes during future iterations which results in a smaller test set. The other two options (ii and iv) are very close to the random ordering.

4.7 Conclusions

We have proposed two new dynamic methods for obtaining test sets with a large number of unspecified bits. The primary goal here is to generate such test sets without relying on an initial test, that can bias the obtained results. We first transform the considered problem into a graph representation, motivated from traditional test set compaction techniques. We call this graph a constraint fault compatibility graph. The constrained compatibility is identified here by function-based ATPG; however structural methods can also apply. Both methods identify compatible faults, constrained by the number of specified bits in the common tests, and try to test them together in order to increase the number of the don't care bits in the test set.

The first method is essentially a simple two phase algorithm which identifies pairwise matching for the graph's vertices. The matched vertices are then merged together, in order to give a single test detecting all the faults corresponding two these vertices. We refer to this method as the *match-and-merge method.* The second method is *hierarchical* and considers a small part of the compatibility graph per iteration, in order to identify the compatible faults. Essentially, it tries to detect as many faults as possible by a small number of tests before proceeding to the generation of other tests.

The experimental results demonstrate that the match-and-merge method (first dynamic method) results in high test set sizes, despite the fact that the percentage of specified bits in the resulting test sets are higher than the hierarchical method (second dynamic method). However, the maximum size of specified bits in a single test is smaller for the first method. On the other hand, the proposed hierarchical dynamic method outperforms, in some cases considerably, existing static methods as well as the static method proposed in Chapter 3 in terms of the total number of specified bits, while keeping the size of the test set size small. These results, together with the statistics on the distribution of specified bits in the obtained test set, imply that the obtained test sets can give benefit to various applications, including test set embedding architectures. We investigate this possibility in the following chapter.

CHAPTER 5

APPLICATION OF RELAXED TEST SETS IN BIST SCHEMES WITH LFSR RESEEDING

5.1 Introduction

Built-In Self Testing (BIST) architectures have received a lot of consideration in the last fifteen years. Modern complex designs demand more reliable and effective embedded testing with no extra operational complexity and with as much reduction in hardware overhead as possible. Replacement of costly Automatic Testing Equipment (ATE) with efficient on-chip embedding architectures is desirable to fill the technology gap between ATE and integrated circuits design capabilities. In this chapter we investigate how two popular BIST architectures can benefit when considering the relaxed test sets obtained by the methods described in Chapter 3 and Chapter 4. The purpose here is to investigate the possible reduction in the overall hardware requirements for standard test set embedding schemes, since it is known that such schemes benefit from test sets with a large number of unspecified bits. Moreover, a small maximum number of specified bits per test, in the test set may reduce the overall hardware since it determines an important part of the embedding hardware.

Traditional BIST schemes use pattern generation circuitry, like Linear Feedback Shift Registers (LFSRs), to generate pseudorandom patterns [75, 76]. However, for more complex designs, it is necessary to apply deterministic test patterns (generated by an appropriate ATPG tool) to cover hard-to-detect faults not targeted by the pseudorandom patterns . Thus, a mixed-mode scheme is usually followed where a pseudorandom sequence of limited length, followed by a sequence of deterministic tests, is applied [77, 20, 46]. Popular deterministic BIST schemes involve ATPG and test set compaction, encoding and regeneration of the embedded tests on the chip, and application of the patterns via scan-chain loading.

Most of the previous work for deterministic test set embedding for stuck-at faults focuses on proposing new BIST architectures along with efficient test encoding techniques to provide for high fault coverage. See [78, 46, 44, 79, 80, 81, 82, 83, 77, 45, 84, 73, 74, 85], among others. Most of these techniques employ test-per-scan application to reduce the hardware overhead as well as the on-chip storage. In contrast some other methods, such that of [81], use test-per-clock schemes to gain from all intermediate patterns produced by the test regenerating device and, thus, reduce the test application time. Many of the existing encoding techniques try to take advantage of the unspecified bits in the test set to be embedded, in order to reduce the on-chip storage for the encoded deterministic test patterns.

Even though existing test pattern generators (TPGs) can generate tests with unspecified bits, they do not specifically consider the number and distribution of the specified bits in the final compacted test set. [79] demonstrated that the scheme of [44] can benefit greatly, in terms of reducing the storage requirement, if the embedded test patterns are generated by an ATPG tool that (i) maximizes the number of unspecified bits in each generated test and (ii) concatenates tests into sequences that have a number of specified bits equal or slightly greater than the maximum number of specified bits in any test pattern.

In this chapter we investigate how the methods presented in Chapter 3 and Chapter 4 can be used in test set embedding schemes. In Section 5.2 we review the previous work in BIST architectures following the test set embedding and/or encoding rationale. Next, we present an experimentation framework we have used in order to thoroughly explore the various test set generation and embedding parameters (Section 5.3). Section 5.4 shows the obtained experimental results and discusses the findings, while Section 5.5 concludes the chapter.

5.2 Overview of Test Set Embedding Schemes

In this section we give a brief overview of the existing work on test set embedding. We assume a *test-per-scan* architecture, with a single scan chain. This implies that the size of the the scan chain equals the number of primary inputs; let this be denoted by m.

The traditional single-polynomial LFSR scheme, used for generating pseudorandom test patterns, has been tuned to regenerate deterministic test patterns, by using different initializing seeds [77] obtained on an algebraic rationale. Thus, it can be used in a mixed mode approach (oftenly referred to as hybrid BIST) for both pseudorandom test generation and deterministic pattern decompression and application. The seeds can be computed in a systematic way, by solving a system of equations based on the number of specified bits $sp(t)$ in a test pattern t. In the general case for a test pattern with some don't care values, (i.e, $t \in \{1, 0, x\}^m$), the corresponding LFSR seed can be computed in the following way. For each specified bit in the pattern, an equation based on the companion matrix of the LFSR is obtained, forming a system of linear equations. The solution of this system of equations gives the seed of the LFSR that can be used for the reproduction of test pattern t. For this test-per-seed scheme it has been shown that a test pattern can be encoded, with a very high probability (greater than $1 - 10^{-6}$) of success, into an $sp(t) + 20$ bit-wide seed. Given a test set $T = \{t_1, t_2, ..., t_k\}$, the total storage amount for the encoded tests is determined by the maximum number of specified bits in a test pattern in the test set, i.e., $S_{max} = max\{sp(t)|t \in T\}$ as well as the distribution of the numbers $sp(t_1), sp(t_2), ..., sp(t_k)$. The hardware overhead is primarily determined by the storage requirement for the LFSR seeds corresponding to all the deterministic test patterns. The size of the LFSR, which determines the size of the LFSR seeds is equal to $S_{max} + 20$.

In the Multiple-Polynomial LFSR (MPLFSR) reseeding scheme of [44], the LFSR hardware is modified to become reconfigurable, i.e. include logic for reprogramming the LFSR feedback. Essentially, reprogramming refers to the process of changing the characteristic polynomial of the LFSR. This can be easily done by adding an AND and an XOR gate for every LFSR bit as shown in Figure 5.1. The *Linear Feedback Controller* (LFC) is a memory structure holding programming sequences, consisting of l bits, where l is the size of the LFSR. Based on the chosen polynomial, the LFC determines the polynomial implemented by the LFSR. The basic reseeding scheme implies storing only the seeds needed to initialize the LFSR. This method manages to improve the encoding efficiency of LFSR reseeding using an LFSR size equal to S_{max} by only considering 16 different polynomials. For the MPLFSR scheme, extra storage is necessary to keep the polynomial selection sequences as well as extra bits for each seed to identify the corresponding polynomials. In [79] the authors managed to further improve the encoding efficiency of the scheme of [44] by allowing a sequence of test patterns to be encoded by a single seed with size close to S_{max} (referred to as test concatenation). [74] also achieves high encoding efficiency when considering variable-length seeds. Once the seed is loaded to the configured LFSR, it requires m autonomous transitions of the LFSR to regenerate the encoded pattern and transfer it to the scan chain for application. The storage requirement for MPLFSR reseeding is given by:

$$Storage = S_b \cdot (P + G) + G$$

where G is the number of seeds (concatenated patterns) to encode the entire test set and S_b is the size of each (concatenated) seed. P refers to the number of characteristic polynomials required for the encoding. By restricting the number of specified bits in the concatenated pattern to be at least equal to S_{max} (the maximum number of specified bits over all patterns in the embedded test set), a high encoding efficiency can be guaranteed. The system formulation for MPLFSR is done on the concatenated patterns and a seed for each pattern is obtained. Thus, the storage requirements for the MPLFSR reseeding scheme is the sum of the bits needed to store the feedback polynomials, the bits to store the seeds and one extra bit for each seed to determine the corresponding polynomial. One bit is used here instead of 4 $\big(=log_2(16)\big)$, since we assume that all seeds corresponding to the same polynomial are stored in successive memory positions and so a 'next bit' is adequate for indicating reconfiguration of the LFSR. Figure 5.1 shows the underlying architecture of the MPLFSR reseeding scheme.

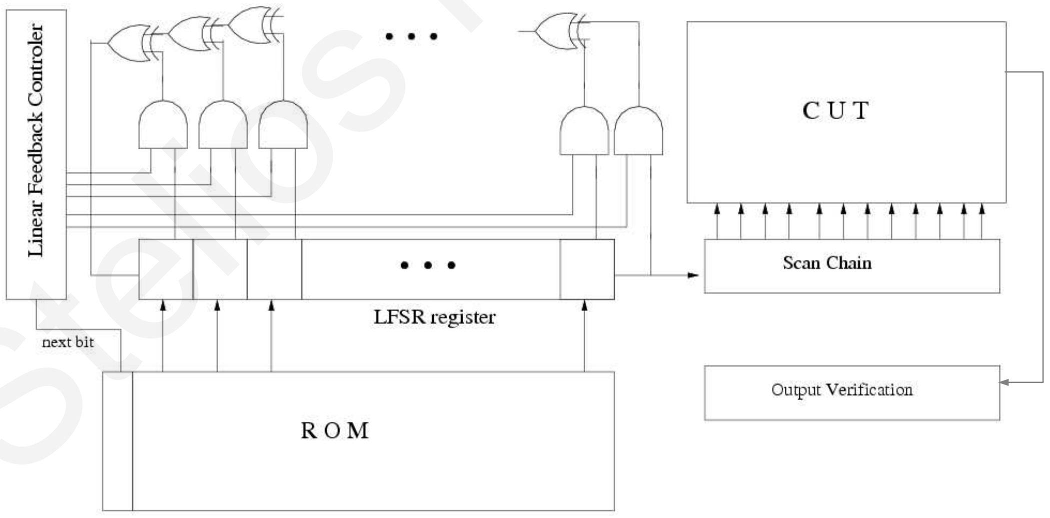


Figure 5.1: The basic Multiple Polynomial LFSR architecture

Several other LFSR reseeding schemes have been proposed that can further improve the overall seed storage at the expense of some extra BIST hardware to the one shown in Figure 5.1. [80] proposed a new architecture based on folding counters to encode the static test set. The number of seeds to be stored benefits from standard encoding techniques as well as input reduction methods, which however implies reorganization of the scan chain. Hardware overhead for this architecture consists of extra counters and comparators and when mixed-mode BIST is desired, an extra LFSR is needed to deploy the pseudo-random patterns generation. In [73] the scheme of [80] is extended to become more flexible by accommodating the LFSR and the folding counter in a more general scheme in order to reduce the overall hardware. Deterministic test cubes are encoded as LFSR seeds and those seeds are then compressed as seeds of the folding counter.

A new architecture is also presented in [85] that uses Twisted-Ring Counters (TRC) to embed a static test set. No extra hardware is added to the CUT as the test generation circuitry can be implemented using the standard scan chain, a multiplexer and an inverter in the serial input of the scan register. Moreover, a seed extraction technique is proposed to get the required seeds that will allow the TRC circuitry to reproduce a given static test set. Here, the number of specified bits in the test set needs to be small as it provides a better seed selection.

In [83] and [82] the authors present a multiphase architecture for regenerating a deterministic test set. The scan chain is fed by multiple LFSR cells (not only the last one) to provide more useful test patterns. During each phase the same set of seeds is loaded to the LFSR but a different LFSR cell drives the scan chain per phase. In this manner a smaller set of seeds needs to be stored at the expense of extra time due to the multiple phase application.

The work in [84] introduces a method for a three-phase mixed-mode BIST. First, a number of pseudo-random test patterns is applied and easy-to-detect faults are dropped from further consideration. Then ATPG is performed to get deterministic patterns for all the remaining faults. A number of repeating sequences is then obtained from those patterns and grouped using a clustering algorithm. The groups of repeating sequences are then used to construct a test set of semi-random tests which are applied at the second phase of this scheme. The repeating sequences, as well as some controlling overhead, are stored on-chip together with deterministic patterns targeting faults which remain uncovered, even after applying the semi-random patterns.

The work in [21] proposed a technique for partially reseeding the LFSR instead of fully replacing the LFSR seed, reducing in this way the seed storage. While the architecture for the partial reseeding scheme is the same as that of Figure 5.1 there is no Linear Feedback Controller which selects from a number of available characteristic polynomials for the LFSR. Instead, there is a structure between the ROM and the LFSR that appropriately combines the current LFSR seed with the partial seed stored in ROM, in order to generate the new LFSR seed. However, the size of the LFSR has to be $S_{max} + 20$ as stated in the original reseeding scheme [77]. For this scheme the storage requirements is given by:

$$Storage_{pr} = (B - 1) \cdot r + S_{max} + 20$$

where B is the number of partial seeds, r is the size of the partial seed and $S_{max} + 20$ is the size of the LFSR and, thus, the size of the initial seed.

All the encoding techniques presented here can be improved if the targeted test set has a large number of unspecified bits.

Without any loss of generality, the remainder of this chapter will focus on two simple LFSR reseeding schemes for mixed-mode BIST, that of multi-polynomial reseeding presented in [44] and that of partial reseeding presented in [21]. We will investigate the usage of the methods of Chapter 3 and Chapter 4, for generating test sets appropriate for the considered schemes. Moreover, we will evaluate the impact of the proposed guided ATPG tool in reducing the on-chip seed storage in such mixed-mode BIST schemes.

5.3 A Generic Framework for LFSR-based Reseeding Parameter Exploration

Instead of using the generated relaxed test sets as they are in the considered embedding schemes, we present a generic framework which allows to efficiently explore certain parameters relevant to the embedding mechanisms, in order to further minimize the seed space requirements. Both the methods of Chapter 3 and Chapter 4 can be incorporated in this framework.

The framework essentially integrates the two relaxation methods and the test embedding techniques discussed in the previous section (i.e., [44, 21]. This exploration provides a systematic way of exploring two parameters of the methods presented in the two previous chapters, when used in the LFSR-based BIST schemes reviewed in the previous section. Before describing the framework we show theoretical and empirical bounds for these parameters.

The framework explores two parameters namely S_h and S_b which are given as input to the relaxation methods and the MPLFSR encoding technique, respectively. S_h is a constraint imposed on the number of specified bits allowed for each test pattern. The proposed framework applies S_h on the relaxation methods (Chapter 3 and Chapter 4) as an upper bound to the maximum number of specified bits per test pattern. Specifically, for the static method (Chapter 3) S_h equals to the S_{max} parameter and for the dynamic method (Chapter 4) equals to the constraint $\mathcal{K}$. S_b is a constraint imposed on the number of specified bits per concatenated test pattern, in the MPLFSR reseeding and determines the maximum number specified bits (and, thus, test patterns) that can be encoded in a single concatenated pattern. S_b is also important because it determines the size of the LFSR and, thus, the hardware overhead of this embedding scheme. Next, we give the range of the values that these parameters take theoretically, and we show how these ranges can be safely made tighter to reduce the number of iterations applied by the framework.

Parameter S_h determines the maximum number of specified bits in any test pattern in the test set obtained by either of the proposed methods (static and dynamic). In order to extensively explore this parameter, the proposed methods should examine all its possible values, which is constrained by $1 \leq S_h \leq \# of PIs$. This is because S_h cannot be greater than the number of primary inputs since this will give a test pattern with more specified bits than its total number of bits. On the other hand, S_h cannot be smaller than the maximum number of specified bits in any test among all tests in the initial test set (i.e., S_{max} of the initial test. Specifically, the lower bound of S_h can be accurately computed when a test set containing all tests for each fault considered is generated and the test with fewer specified bits for each fault is isolated. The test with the maximum number of specified bits determines the exact lower bound of S_h. Since this information is available only when symbolic techniques (like that of [86]) are used, we have experimented by giving S_h as small values as possible, in this case 1, only avoiding fully unspecified bits. Of course this lower bound results in unacceptable fault coverage. Extensive experimentation, on the benchmark circuits considered, shows that when starting the S_h exploration form value $\frac{\#\ of\ PIs}{2}$ usually 100% fault coverage is achieved. Thus, the following empirical lower bound may be used: $\frac{\#\ of\ PIs}{2} \leq S_h \leq \#\ of\ PIs$

Another important parameter which is, however, relevant only to the embedding scheme of MPLFSR [44], is S_b. As discussed in the previous subsection, S_b is the size of the concatenated test pattern which equals the size of the LFSR. The bounds for the S_b parameter are within the S_h obtained and the total number of specified bits in the test set.

Parameter S_b which, however, is relevant only to the embedding scheme of MPLFSR [44] determines the size of the concatenated test pattern which equals the size of the LFSR. The bounds for the S_b parameter are within obtained S_h and the total # specified bits in the test set (i.e., *obtained* $S_h \leq S_b \leq$ *total # specified bits in the test set*). The lower bound is because the embedding procedure assumes the test sets obtained by one of the proposed methods as its input. Thus, the concatenated pattern cannot have fewer specified bits that the test with the maximum number of specified bits in the test set which is constrained by S_h. Moreover, S_b can be equal to the total number of specified bits of the entire test set. This is the case when all test patterns are concatenated in a single test pattern, which is clearly not desired and in most cases not feasible. In practice, we examine values of S_b that are close to and larger than S_h, as this keeps the size of the LFSR small ant, thus, efficient to implement. Nevertheless, the experimental results show that the lower storage requirements are when S_b is much smaller than S_h+200 and in most cases very close to S_h. Experimental results show that for values of S_b that are not higher than $1.2 \times S_h$ gives the best results in terms of storage requirements. Specifically, the exploration of all possible values of S_b (defined by the theoretical bounds) showed that for values between 100% and 120% of S_h, the best results have been obtained. The larger values were obtained for circuits with a small number of primary inputs, and, thus it may not be the case with large industrial circuits. For instance, for a circuit with only 32 primary inputs, the best results, in terms of storage, are obtained for $S_h = 20$ and $S_b = 24$ which implies 20% increase ($h = 120\%$). The same increase for a circuit with 200 primary inputs and $S_h = 120$ gives $h = 103\%$. However, experimentation shows that setting a constant value for S_b (instead of a percentage on S_h) makes the experimntation framework not general, and, thus not applicable for a large range of circuits. Of course, if the designer has the luxury to allow a larger sized LFSR, then the practical bound of S_b can be increased appropriately. Based on this, in the proposed framework we set the upper bound of S_b to be slightly larger than S_h, i.e., $h \times S_h$, where h is a user defined parameter greater than 1. Our expiriance showed that this number can be safely put to $h = 1.2$. Thus, the empirical bounds of parameter S_b, used in this framework, are given by $S_h \leq S_b \leq h \times S_h$.

Figure 5.2 shows the flowchart of the proposed parameter exploration framework. When the desired method is selected, it is iteratively run for the various values of S_h. For the static method (Chapter 3), S_h is used to constrain S_{max}. Any test that has more specified bits than S_h is not accepted to be in the final test set. For the dynamic method, S_h is used as the value $\mathcal{K}$ to determine $\mathcal{K}$-compatibility. Since, in the fault compatibility graph, no vertex or edge weight is larger than $\mathcal{K}$, the resulting test set does not contain tests with larger than S_h specified bits. If for any value of S_h 100% fault coverage is not achieved, the results are discarded and the next value of S_h is applied as a constraint.

The obtained test set is then encoded using one of the two techniques of MPLFSR [44] and partial reseeding [21] and the best results among those with various values of S_h are saved. Here, better results imply smaller storage requirements and at the same time small size of the LFSR. While priority is given on the test set storage, the size of the LFSR is also important and test sets with (slightly) larger storage and significantly smaller LFSR size are preferred. Next, S_h increments by one and the same process is repeated, until S_h reaches its upper bound (number of PIs). In the case of the MPLFSR technique, where test pattern concatenation is preformed, two extra steps are inserted (shown in gray background) in order to explore the S_b parameter. The first extra (upper) step is made in order to initialize S_b and apply the test concatenation algorithm. The second (lower) step increments the S_b parameter and checks if the upper bound is met. If not the concatenation and encoding procedures are repeated. Note, that this nested iterative process does not apply to partial reseeding. When both parameters of interest are fully explored, the process terminates and the (saved) best results are returned.

Since the two parameters S_h and S_b are highly related, in the MPLFSR scheme, we have experimented with taking a combined decision of the values of the two, instead of first choosing S_h and then S_b. This is illustrated in Figure 5.3 where we show the exploration of the S_b based on the proposed framework for a number of values of S_h, close to the best one as determined for the test generation process, for circuit s15850. Observe that for S_h = 36 the storage is similar and falls below 4,500 bits in two cases, for S_b = 60 and S_b = 99. In this

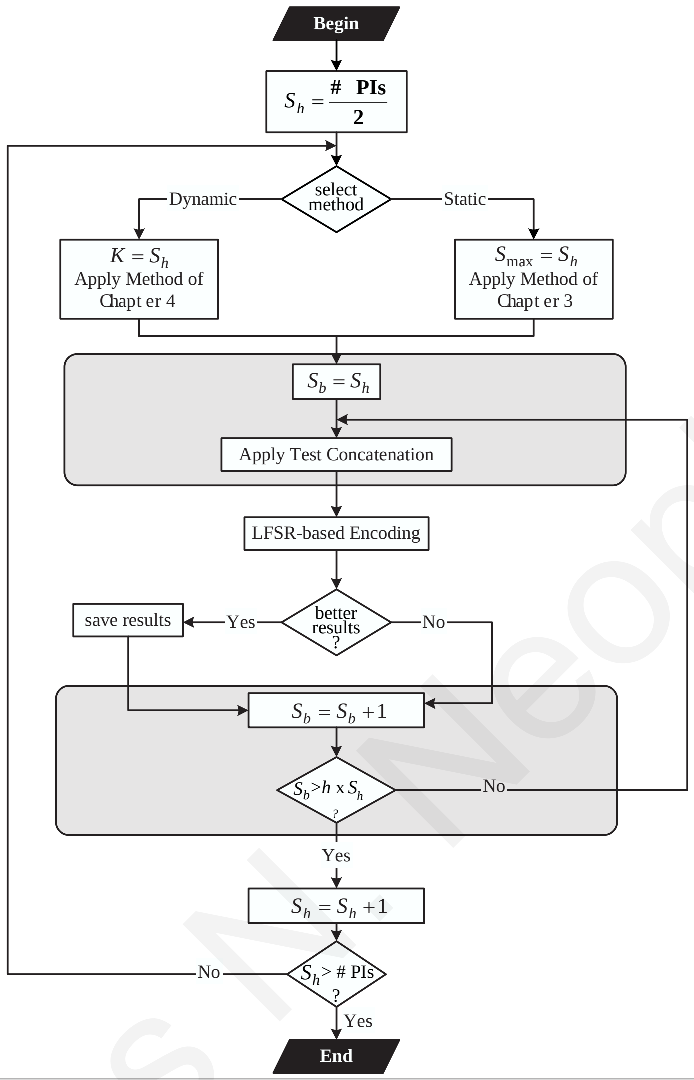


Figure 5.2: Flowchart for parameter exploration framework

case, S_b = 60 is selected since it gives a smaller size LFSR. While, this can be done with all values of the S_h, it is sufficient to be explored only for the four closest values to the best S_h, as this has shown (from experimentation) to give the lower storage requirements. Higher values of S_b can give lower total storage, yet the LFSR size will become too large and, thus, these results are not preferred. These observations confirm that the best value for S_b is very close to S_h.

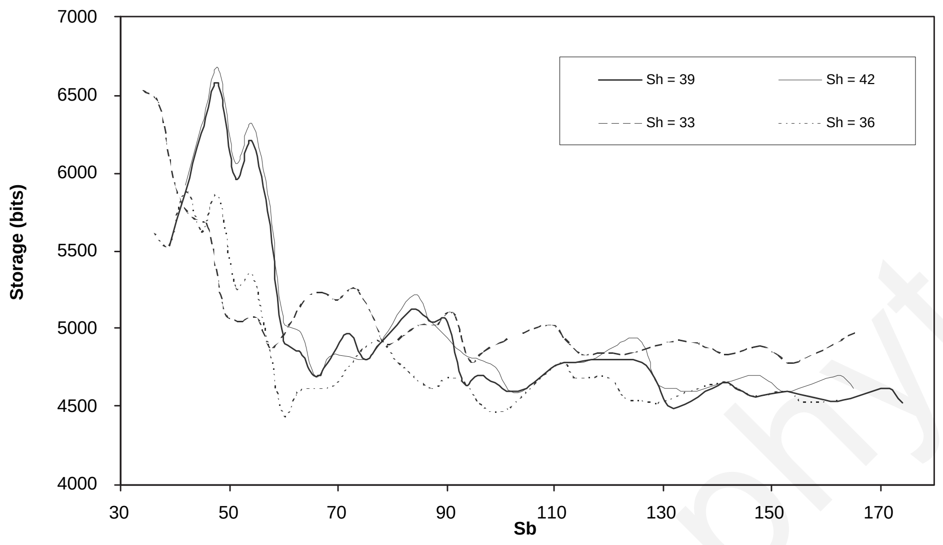


Figure 5.3: Selecting the S_h and S_b parameters.

5.4 Experimental Results

The proposed framework was implemented in C language and run on a 3GHz Pentium 4, running Linux with 1GB of RAM. We experimented with the ISCAS'85 and the full-scan versions of the ISCAS'89 benchmark circuits. The tool of [21] was used for the partial re-seeding, while the method of [44] was implemented using a first fit decreasing Bin-Packing algorithm for the test set concatenation step. We used the best test sets obtained by either the static or the dynamic methods, after exploring parameters S_h and S_b (where applicable). Since LFSR-based encoding has been shown to be beneficial for hybrid BIST schemes, we first applied 10000 pseudo-random tests in order to catch all the easy-to-detect faults, and then encoded only tests that target the remaining, hard-to-detect, faults using the two encoding schemes. All the generated test sets were verified for 100% coverage of the targeted faults, using the FSIM simulator of [87].

Table 5.1 shows these results for the larger ISCAS '85 and ISCAS '89 circuits. After the circuit name the number of faults considered by the proposed method is reported, i.e. the number of faults not targeted after application of 10,000 pseudo-random test patterns. For each of the two techniques we report the number of seeds to be stored (Columns 3,6,10 and 13), the LFSR size (Columns 4,7,11 and 14) as well as the total storage size, in bits

Table 5.1: Using the resulting test sets with two popular BIST encodings.

Circuit	No. faults	Full Reseeding (MPLFSR) [79]							Partial Reseeding [21]						
		Proposed			Reported in [79]			Red.	Proposed			Reported in [21]			Red.
		seeds	LFSR	Storage	seeds	LFSR	Storage	Ratio	seeds	LFSR	Storage	seeds	LFSR	Storage	Ratio
c2670	424	31	58	2061	52	60	3412	0.60	99	65	3165	N/A	N/A	N/A	N/A
c7552	445	19	80	2179	41	100	5241	0.42	61	110	3478	N/A	N/A	N/A	N/A
s838.1	343	50	51	2875	39	36	1623	1.79	99	69	3104	N/A	N/A	N/A	N/A
s1196	11	11	16	235	12	17	267	0.88	11	36	256	N/A	N/A	N/A	N/A
s1238	85	9	17	213	11	17	249	0.86	7	37	133	N/A	N/A	N/A	N/A
s1423	17	3	18	111	N/A	N/A	N/A	N/A	3	38	64	N/A	N/A	N/A	N/A
s9234.1	1146	46	83	4412	103	61	6923	0.66	86	85	3485	138	81	5013	0.70
s13207.1	819	43	54	2635	138	24	3570	0.74	82	58	2299	157	44	3008	0.77
s15850.1	990	57	76	4767	134	46	6528	0.73	115	70	3801	167	58	5204	0.73
s38584.1	2007	44	65	2845	46	70	3406	0.86	54	91	2328	62	75	2942	0.79

(Columns 5,8,12 and 15). The results are compared with those reported in [79][1] and [21], for mixed-mode BIST. Columns 9 and 16 show the reduction ratio in total storage for the two techniques. Here, N/A means not available values for the corresponding circuits.

For the technique of [79] in all but one circuit the test sets obtained by the proposed methods give lower seed storage, whereas the LFSR sizes, in most cases, are kept closed to those reported in [79]. Note, that the test sets used in [79] were also optimized to have a large number of unspecified bits. Nevertheless, using the framework of Section 5.3, the results with smaller LFSR size can be obtained, at the expense of more seed storage, by appropriately adjusting the decision taken regarding the results to be saved.

In the case of partial reseeding of [21], the total seed storage is also reduced, as it can be seen from Column 16. In all cases the obtained results are between 70 % and 80 % of the reported in [21] which is considerable for BIST schemes. The LFSR size is also kept close to that of [21], while the number of seeds is significantly reduced.

[1]We compare with the results reported in [79] instead of comparing with the results of [44], since the former provides more detailed results and for more circuits. The work of [79] generalizes the work of [44], yet the results in [79] are comparable to the results presented here.

5.5 Conclusions

The work in this chapter examines a specific application that can benefit when used in conjunction with test sets that have a large number of unspecified (don't care), namely built-in self test. We have reviewed the most important work for BIST related with test set embedding at the gate level. Two such methods that were made available to us have been used in order to evaluate how the test sets obtained by the methods presented later in this thesis (i.e, in Chapter 3 and Chapter 4) impact the final on-chip storage. A new systematic experimentation framework is proposed in this chapter. The framework allows a thorough exploration of the test generation and/or relaxation parameters, as well as, the test set embedding parameters, in order to find the solution with minimal area overhead. This exploration plays an important role in obtaining optimal results, since it is done systematically and based on theoretical and empirical bounds. The experimental results show improvement when used in a mixed-mode BIST scheme. The embedding storage overhead reduction is, in most of the cases, significant and denotes the importance of the static test set relaxation method and the relaxed test set generation method presented in the two previous chapters.

CHAPTER 6

RELAXATION OF n-DETECT TEST SETS

6.1 Introduction

The on-going increase in the complexity of the modern VLSI microchips demands more sophisticated post-manufacturing testing methodologies and/or procedures. Current nanometer manufacturing processes suffer from larger defective parts ratio, partly due to numerous emerging defect types. While traditional fault models, such as the stuck-at and transition delay fault models are still widely used, they have been shown to be inadequate to handle these new defects. One obvious approach is to develop complex fault models to imitate defect behavior at either the logic or layout level of abstraction. The combination of the large number of possible defect types together with the huge number of fault sites in a modern circuit implies that modeling these defects will give prohibitively large input for a systematic test generation methodology [88, 89]. Moreover, detailed layout information is typically not available until the fabrication phase, giving limited information to test engineers.

Instead, previous work proposed the use of test sets targeting each modeled fault multiple times in order to increase the probability of detecting additional fault types as well as the defect coverage. The rationale behind detecting a fault more than once is to achieve higher quality in terms of defect coverage, by generating a number of different tests for each modeled fault. Such test sets are known as n-detect test sets, since they detect each fault with n different tests. As a result, a variety of n-detect test set generation methods and their impact in the quality of the testing process have been proposed [90, 91, 92, 93, 94, 95, 96, 97, 98, 99].

Existing methodologies for n-detect test generation produce tests that are fully specified (i.e., all the test set bits have a fixed value of 0 or 1). This occurs since many of these techniques try to fix unspecified (don't care) bits to logic values such that the number of detected faults is increased. Actually, even if bit fixing does not improve on the n-detect fault coverage, it can improve on the coverage of non-targeted faults and defects even by randomly fixing the unspecified bits. As a result, existing test generation tools return fully specified test sets. This, however, limits the applicability of n-detect test sets in several currently important problems. Methods for low power test generation [11, 42, 100], for instance, can benefit when the input test set includes a large number of unspecified bits by appropriately fixing those bits. Such flexible test sets are also extremely crucial in various compression schemes for on-chip or off-chip test set embedding, given in [79, 80, 83, 84, 85] among many others.

The work in this chapter considers the problem of relaxing an n-detect test set. The given test set can be fully or partially specified. The total number of specified bits in the resulting test set is minimized, while maintaining its original n-detect fault coverage. Furthermore, the test set size is guaranteed not to increase. The motivation behind this problem is that a test bit needs to be *initially* fixed only if this helps the n-detect fault coverage, otherwise it can be left unspecified. The generated relaxed test set can then be used in a variety of applications that fix the unspecified bits appropriately. The applied fully specified test set is expected to have similar defect and non-targeted fault coverages to that of the original test set. This is justified by the simple observation that in existing test generation techniques considering the traditional n-detect fault definition, any improvement in the coverage of non-targeted faults and defects, beyond the n-detect per targeted fault improvement, is caused by the random bit fixing. The latter is supported by experimentally obtained data.

Static relaxation of 1-detect test sets was studied in [48, 49] and in Chapter 3 which have proposed methods relying on various ATPG concepts in order to identify specified bits in the test set that can be replaced by don't care values. [48] proposed a method for identifying don't care bits in a test pattern using ATPG concepts such as implication and justification. [49] used a similar rationale, taking into consideration testability measures in the justification process. The methods of Chapter 3 identify all fault detections for each fault under a given test set and explicitly remove extra detections by using different selection criteria. Extending these methods to n-detect test sets is not straightforward. Actually, these methods benefit from identifying essential tests[1] which do not exist in n-detect test sets. The two methods

[1]A test is essential if it is the only one detecting a fault

presented in Chapter 4 also consider test set relaxation, but in a dynamic manner. They do not consider an initial test set to be relaxed; rather, they both restrict the ATPG process to consider the number of specified bits in the generated compact test set. As a result, these dynamic approaches cannot be easily extended to n-detect test sets, especially for large values n. While the extension of both the static and dynamic methods to n-detect test sets could be investigates, in this chapter we propose a new methodology that is optimized based on the characteristics and parameters of the n-detect test sets.

Here, test set relaxation does not imply that the specified bits of the relaxed test set are a subset of the specified bits of the initial test set, as it is the case with the existing static relaxation methods in [49, 48]. Rather, relaxation refers to the process of increasing the total number of unspecified bits in order to make the test set more “flexible” for other applications. A novel systematic test replacement algorithm is proposed, in which each test is replaced by a new one that detects a subset of the faults detected by the first one, with fewer specified bits. In order to maintain the fault coverage, each fault is guaranteed to be detected at least n times, where this is possible[2]. The algorithm explicitly removes additional (more than n) detections for each fault. The latter is possible since experimentation shows that in n-detect test sets the average detections for each fault is much greater than n, mainly, due to the presence of many easy-to-detect (randomly detected) faults. Specifically, the methodology targets an optimization problem; it determines the n most appropriate tests to detect a fault (among all of the tests that detect the fault) that give the maximum benefit in terms of specified bits savings in the entire test set. Thus, it selects the “best” n tests to detect the fault and drops the fault from the remaining tests in order to reduce the total number of specified bits in these tests.

The rest of this chapter is organized as follows. Section 6.2 elaborates on our motivation and presents supportive data. Section 6.3 gives necessary notation and the problem formulation, and Section 6.4 describes the proposed technique. A comprehensive example illustrating the proposed method’s execution is presented in Section 6.5. Section 6.6 gives the obtained experimental results for the specified bits reduction together with necessary discussion. This experimentation was done using the popular stuck-at model; however other (linear) fault model can be used such as the transition fault model discussed in Chapter 2. In Section 6.6 we also present experimentally obtained information on the method’s parameters and usage.

[2]some faults inherently have less than n different tests

Table 6.1: Average fault detections for single-detect and multiple-detect test sets

	Fully specified single-detect test set T_A				**Partially specified single-detect test set T_B**					**Fully specified 10-detect test set T_N**	
Circuit	$\lvert T_A \rvert$	$AD(T_A)$	$AD(T'_A)$	$AD(T''_A)$	$\lvert T_B \rvert$	**Xs (%)**	$AD(T_B)$	$AD(T'_B)$	$AD(T''_B)$	$\lvert T_N \rvert$	$AD(T_N)$
s510	64	10.83	5.24	10.91	65	0.30	11.86	5.96	12.03	543	57.17
s526	60	6.58	4.70	6.82	68	4.10	5.29	4.56	7.11	492	60.07
s641	55	9.26	4.13	9.11	62	12.80	9.25	4.22	9.42	227	45.01
s820	108	3.58	2.21	3.63	116	13.50	3.67	2.35	3.85	949	31.92
s953	91	26.71	13.21	26.18	95	3.80	18.65	11.26	27.23	766	112.21
s1196	144	17.66	9.98	17.99	151	23.10	14.36	8.65	18.37	1233	93.30
s1423	24	8.65	3.23	4.48	70	8.20	8.14	3.45	17.65	269	48.82
s1488	118	4.12	2.89	4.21	123	0.40	4.15	2.95	4.42	209	8.93
s9234	411	45.62	32.12	45.63	495	26.80	35.67	28.76	47.62	1132	142.14
s13207	472	77.46	56.39	77.51	692	22.10	61.48	54.59	83.15	2341	354.22
s15850	441	64.27	43.34	64.03	519	38.50	38.54	29.39	67.31	983	156.52

6.2 Motivation

Previously proposed methods for deriving single-detect relaxed test sets can be categorized into static ([49, 48] as well that of Chapter 3) or dynamic (Chapter 4). Static methods consider an initial test set whereas dynamic methods incorporate the problem in the ATPG process. Extending these methods to n-detect test sets is not straightforward. Actually, the static methods benefit from identifying essential tests (a test is essential if it is the only one detecting a fault) which do not exist in n-detect test sets. Moreover, the common underlying idea used in both types of methods is the identification of coincidental multiple times fault detections, i.e. a fault is detected by several different tests even though it was only targeted once in the test generation process. The latter occurs very often, especially in the traditional stuck-at fault model (as well as in transition delay faults which are often modeled as stuck-at faults during test generation), because the majority of the faults are easy-to-detect (also referred to as randomly detected faults). The static methods drop multiple times detection implicitly, through fault simulation and fault dropping, in order to determine bits that can be relaxed. The dynamic method of Chapter 4 proceeds in a different rationale. It identifies sets of faults that can be detected by a single test with a small number of specified bits and explicitly avoids multiple-times detections. In any case, the derived relaxed test sets still include some multiple-times detections due to coincidental fault detection. At this point we have to define how we use the average detections concept in this context.

Definition 6.1. The *Average Detections* of a test set T (referred as $AD(T)$) denotes the average times a fault is detected by a test set T. That is, the total detections of the faults in a considered fault list F by the tests in the test set T divided by the number of faults in F. If we call $\mathcal{D}_{f_i}^T$ the number of tests in T that detect fault f_i then the average detection parameter is given by:

$$AD(T) = \frac{\sum_{f_i \in F} \mathcal{D}_{f_i}^T}{|F|}$$

When T is relaxed to a partially-specified test set T' (with the same fault coverage) then, based on the above definition and if $\mathcal{D}_{f_i}^T$ counts only detections by the specified bits of T', $AD(T') < AD(T)$. Table 6.1 lists data that supports this observation. Columns 2-10 list data for two different 1-detect stuck-at fault test sets derived from ATALANTA [101]. Test set T_A is fully-specified whereas T_B contains a small number of don't care bits. Columns 2 and 6 show the number of tests in T_A and T_B, respectively. The percentage of the don't care bits, with respect to the total number of bits, of T_B is given in Column 7. The average detections per fault in T_A and T_B are listed in Columns 3 and 8, respectively. Observe that $AD(T_A)$ and $AD(T_B)$ are much higher than 1 in these 1-detect test sets, due to the coincidental fault detections. Both test sets were relaxed using a technique similar to that of [48]. The average specified bits reduction in T_A is 82% and in T_B 75% (fully-specified test sets allow for higher specified bit reduction than partially-specified test sets of similar size). The fault coverage and the test set size of the initial test set T_A (T_B) are maintained in T'_A (T'_B). Columns 4 and 9 give the $AD(T'_A)$ and $AD(T'_B)$ of T'_A and T'_B, respectively. As expected, based on the discussion of the previous paragraph, the average detections per fault drops in the relaxed test sets, increasing the number of unspecified bits.

The motivation behind relaxing 1-detect test sets is to make these test sets amenable to addressing additional issues beyond detection of the targeted faults. For example, the unspecified bits can be specified appropriately to detect additional faults such as delay or bridging faults. This process is referred to as test enrichment in [43] and has been used in Chapter 2 in a transition fault framework. Alternatively, the unspecified bits can be specified in such a manner that power dissipation during test set application is minimized [42]. Also, relaxed test sets give storage space reduction in on-chip or off-chip compression techniques, which fully specify the test set before test application in some deterministic manner based on the de-compressing hardware. The recent work in [91] proposed a new method that takes advantage of the unspecified bits produced by a standard 1-detect ATPG tool, in order to embed multiple detection in a 1-detect or an n-detect test set. This method can be combined with the work proposed here, in order to maximize the times a fault is detected.

In any case, fully specified test sets are finally applied. Since fully-specified test sets take advantage of coincidental fault detections to increase the average detections per fault, relaxed test sets are expected to have the same advantage when they are finally applied. Columns 5 and 10 give the $AD(T''_A)$ and $AD(T''_B)$ of the test set derived after the relaxed test sets T'_A and T'_B were randomly fully specified. Observe that $AD(T_A) \approx AD(T''_A)$ and $AD(T_B) < AD(T''_B)$. The latter holds since the average fault detections per fault is lower in the original partially specified test set (T_B), than in the final fully specified test set (T''_B).

Maintaining average fault detections may not be of much importance in 1-detect test sets, since their goal is to detect the targeted faults. However, it becomes of great importance in n-detect test sets which are intended for increased non-targeted fault and defect coverage. Coincidental fault detection (by randomly fixing some test bits) occurs similarly in n-detect test sets as in 1-detect test sets. Column 12 of Table 6.1 gives $AD(T_N)$ of the fully specified 10-detect test set of [97], which is much higher than the targeted 10-detection coverage. This shows a lot of room for relaxation in the n-detect test sets. At the same time, since the $AD(T_N)-n$ detections were achieved coincidentally, a relaxed test T'_N (which maintains the n-detection fault coverage) will recover the reduction in $AD(T'_N)$ when it will be finally fully specified before test application. Thus, relaxed n-detect test sets are expected to maintain their non-targeted fault and defect coverages.

6.3 Problem Formulation and Notation

Here, we expand the problem definition of test set relaxation presented in Chapter 3 for n-detect test sets.

Consider a given n-detect test set $\mathcal{T}$=$\{t_1, t_2, ..., t_m\}$ for a combinational or a fully-scanned sequential circuit-under-test $\mathcal{C}$. Each of the m test patterns consists of strings of 3-valued bits $\in$ {0,1,x}. Consider also a fault model $\mathcal{M}$, based on which the list of faults detected by $\mathcal{T}$, denoted by F, is derived. For the considered fault list, the test set $\mathcal{T}$ has $N(\mathcal{T})$ *n-detect fault coverage* and $K(\mathcal{T})$ *specified bits ratio*. In the following we define these two parameters of a test set.

Definition 6.2. The n-detect fault coverage of a test set $\mathcal{T}$, denoted by $N(\mathcal{T})$ is the percentage of the faults considered, under a given fault model $\mathcal{M}$ (i.e. F), that are detected by $\mathcal{T}$ with n different tests. Note, that for some faults $F_p \subseteq F$, only $p < n$ different tests exist (i.e. can be generated by any test generation process), where $p \geq 0$. In the case where no tests exist for a fault f_r (i.e., $p = 0$), then fault f_r is considered to be redundant. For the cause of this definition and throughout this text, $N(\mathcal{T})$ is calculated considering all faults in F_p including redundant. For those faults that $p > 0$ then they are considered to have full n-detect fault coverage and, thus, do not reduce fault coverage.

For completeness, we repeat here the definition of $K(\mathcal{T})$, first introduced in Chapter 3, which denotes the specified bits ratio in a test set.

Definition 6.3. For a test set $\mathcal{T}$, we denote the ratio of the bits having a specified value $\{$0,1$\}$ over the total number of test set bits by $K(\mathcal{T})$. This ratio gives a test set property that indicates how flexible a test set is. Clearly, $0 \leq K(\mathcal{T}) \leq 1$, for any test set. The closer $K(\mathcal{T})$ is to 0, the more flexible $\mathcal{T}$ is. For fully specified test sets, $K(\mathcal{T}) = 1$.

The test set relaxation process refers to replacing test set $\mathcal{T}$=$\{t_1, t_2, ..., t_m\}$ with a test set $\mathcal{T}'$=$\{t'_1, t'_2, ..., t'_m\}$ such that each of the following constraints is satisfied:

(i) $N(\mathcal{T}') \geq N(\mathcal{T})$
(ii) $K(\mathcal{T}') < K(\mathcal{T})$

The above constraints give the specifications of the test set relaxation problem considered. Constraint (i) preserves the n-detect fault coverage in the same way as the fault coverage is preserved in single detect test sets. If a fault f_i is detected $p < n$ times in test set $\mathcal{T}$ (i.e., $\mathcal{D}^T_{f_i} = p < n$), then fault f_i is detected p times in the relaxed test set $\mathcal{T}'$. All other faults, i.e. those with $\mathcal{D}^T_{f_i} = p \geq n$, are detected at least n times in the relaxed test set, and not necessarily p times. Increase of the fault coverage may occur due to coincidental detections of those faults that in T they have $\mathcal{D}^T_{f_i} = p < n$ and in T' they have $\mathcal{D}^{T'}_{f_i} > p$. Constraint (ii) comes directly from the definition of the relaxation problem since the overall goal is to decrease the portion of specified bits ($K(\mathcal{T})$) in the test set.

6.4 Proposed Methodology

In the proposed method, every test in $\mathcal{T}$ is systematically replaced by a new test with more unspecified bits. The algorithm concentrates on one fault at a time to determine n different tests $\{t_j, j = 1, 2, ..., n\} \in \mathcal{T}$ that detect the fault such that the number of bits that can be relaxed in the entire test set is maximized. Put differently, the algorithm determines n tests to explicitly target the detection of the fault and relaxes the bits required to detect the fault in the remaining tests.

Consider a fault f_i detected by $\mathcal{T}$. Let $\mathcal{T}_i \subseteq \mathcal{T}$ denote the set of tests in $\mathcal{T}$ that detect fault f_i. The algorithm finds the n tests in $\mathcal{T}_i$, given in $\mathcal{T}_i^n \subset \mathcal{T}_i$ that should detect fault f_i. Consider a test $t_k \in \mathcal{T}_i$. Let the number of specified bits in t_k that can be unspecified if t_k no longer detects f_i be denoted by c_{ik}. In other words, c_{ik} is the contribution, in specified bits, of fault f_i in test t_k. Then, the total number of specified bits in $\mathcal{T}$ that can become unspecified if fault f_i is only detected by test $t_j \in \mathcal{T}_i$ (and not by any other in $\{\mathcal{T}_i - t_j\}$) is given by:

$$G_{ij} = \sum c_{ik}, \quad t_k \in \{\mathcal{T}_i - t_j\} \tag{6.1}$$

Thus, G_{ij} denotes the gain in unspecified bits if fault f_i is only explicitly targeted during the test generation by test t_j. Of course, coincidental detection of f_i by other tests may occur but this is done by no extra cost in terms of specified bits.

In order to determine which n tests of $\mathcal{T} = \{t_1, t_2, ..., t_m\}$ must explicitly target fault f_i , we calculate:

$$\hat{G}_i = \max_j\{G_{ij}\}, \quad t_j \in \mathcal{T}_i \tag{6.2}$$

n different times, removing t_j from $\mathcal{T}_i$ each time this calculation is made. All the selected tests form the set of tests $\mathcal{T}_i^n$ that explicitly target fault f_i.

Equation 6.1 and Equation 6.2 are the same used in 1-detect test set relaxation. At this point we expand the two equations is such a way that can ensure n-detect fault coverage. It can be argued that, since we want to find the number of test set bits that can become unspecified after keeping n detections only, Equation 6.1 and Equation 6.2 should take into consideration all combinations of n detections. In other words, it is a question whether keeping the n detections that give the larger specified bits relaxation is quite as effective as keeping the combination of n detections that give the larger relaxation. Next, we prove that the two decision criteria are identical. First we slightly modify Equation 6.1 and Equation 6.2 in order to evaluate the gain and maximum gain in specified bits considering all combinations of n detections for the same fault.

$$G_{ij}^n = \sum c_{ik}, \quad t_k \in \{\mathcal{T}_i - \tau_j\} \tag{6.3}$$

With τ_j we denote a subset of $\mathcal{T}_i$ that have size n. Thus, G_{ij}^n is calculated for all combination of n tests out of all tests that detect fault f_i.

$$\hat{G}_i^n = \max_j \{G_{ij}^n\}, \quad \tau_j \subset \mathcal{T}_i \tag{6.4}$$

Theorem 6.1. *When an n-detect test set T is fault simulated against a fault list F, there exist a set of faults $F_m \subseteq F$ that are detected more than n times. For every fault $f_i \in F_m$ we keep those detections (n tests) that when using Equation 6.2 give the n higher values, in set $S(i)$. Moreover, for the same fault $f_i \in F$ we keep those detections (tests in τ_j) that when using Equation 6.4 give the higher value, in set $C(i)$. We next show that $S(i)$ and $C(i)$ coincide for all faults in F_m.*

Proof. Set $S(i)$ contains the n tests that give the maximum values when calculating Equation 3.2, for fault f_i. By substituting Equation 3.1 in Equation 3.2 we have:

$$\hat{G}_i = \max_j \{ \sum_{t_k \in \{\mathcal{T}_i - t_j\}} c_{ik} \}, \quad t_j \in \mathcal{T}_i \quad \Leftrightarrow$$

$$\hat{G}_i = \max_j \{ \sum_{t_k \in \mathcal{T}_i} c_{ik} - c_{ij} \}, \quad t_j \in \mathcal{T}_i$$

and since all c_{ij} are non-negative integers we have :

$$\hat{G}_i = \min_j \{c_{ij}\}, \quad t_j \in \mathcal{T}_i \tag{6.5}$$

In the same manner set $C(i)$ contains the n tests which are elements of τ_j that gives the maximum value in Equation 6.4, for fault f_i. By substituting Equation 6.3 in Equation 6.4 we have:

$$\hat{G}_i^n = \max_j \{ \sum_{t_k \in \{\mathcal{T}_i - \tau_j\}} c_{ik} \}, \quad \tau_j \subset \mathcal{T}_i \quad \Leftrightarrow$$

$$\hat{G}_i^n = \max_j \{ \sum_{t_k \in \mathcal{T}_i} c_{ik} - \sum_{t_h \in \tau_j} c_{ih} \}, \quad \tau_j \subset \mathcal{T}_i$$

and since all c_{ih} are non-negative integers we get:

$$\hat{G}_i^n = \min_j \{ \sum_{t_h \in \tau_j} c_{ih} \}, \quad \tau_j \subset \mathcal{T}_i \tag{6.6}$$

Equation 6.5 implies that the set $S(i)$ contains the n tests that have the minimum c_{ij} i.e., the tests that are elements of the subset of $\mathcal{T}_i$ of size n that has the minimum sum of c_{ij}. The subset of $\mathcal{T}_i$ of size n that has the minimum sum of c_{ij} can be identified by Equation 6.6 which, however, gives the set $C(i)$. Thus, sets $S(i)$ and $C(i)$ are identical. $\square$

Equation 6.1 suggests that using Equation 6.1 and Equation 6.2 for selecting the best tests detect each fault gives exactly the same decision as using Equation 6.3 and Equation 6.4. Thus, there is no need of finding the contribution in specified bits for all combinations of n tests in the set $\mathcal{T}_i$ for fault f_i, in order to keep those tests that give the higher reduction in terms of specified bits.

Figure 6.1 shows the proposed algorithm. The input parameters are the circuit-under-test $\mathcal{C}$, the test set to be relaxed $\mathcal{T}$, the n-detect parameter n, and the considered fault model $\mathcal{M}$ based on which the targeted fault list F is derived (lines 1-2 of Figure 3.2). First, fault simulation is performed to derive the complete fault list F as well as the fault lists F_j for each test $t_j \in \mathcal{T}$. Then, the algorithm iterates over each fault $f_i \in F$, following a predefined ordering (see discussion in Subsection 6.6.1), in order to determine the "best" n tests to detect f_i. This is done by examining only tests in $\mathcal{T}$ that detect f_i, that is $\mathcal{T}_i$ (lines 6-19). For every test $t_j \in \mathcal{T}_i$ the contribution of f_i in t_j (c_{ij}) is first calculated (line 8). This is a crucial step which invokes a test generation routine. Specifically, to find c_{ij} for a fault f_i and a test t_j detecting the faults in F_j, we generate a test cube t' targeting faults in $F_j - f_i$. If the number of specified bits in a test t_j is denoted by $s(t_j)$, then, $c_{ij} = s(t_j) - s(t')$. This is the number of specified bits savings if test t_j no longer detects fault f_i. Once c_{ij} is calculated for every test $t_j \in \mathcal{T}_i$, the total gain G_{ij} in unspecified bits (meaning f_i is detected by t_j but not by $\{\mathcal{T}_i - t_j\}$) for every test t_j is easily computed (line 10). Consequently, the n tests giving the maximum gain are determined (lines 11-14). This is achieved by calculating n times the maximum gain, each time removing all the previously found tests with maximum gain. Tests $t_{m_d} \in \mathcal{T}_i, d = 1, 2...n$ form the test $\mathcal{T}_i^n$ containing all tests that detect f_i.

The next steps (lines 16-19) convey the dynamic nature of the algorithm. Once set $\mathcal{T}_i^n$ is determined, it is no longer necessary for tests $\{\mathcal{T}_i - \mathcal{T}_i^n\}$ to detect f_i. Therefore, the fault list F_j for each of the remaining tests $t_j \in \mathcal{T}_i$ is updated. In this manner, fault f_i will never be targeted in any subsequent test generation step (line 8). Observe that if a test's fault list becomes empty at any point, the test can be fully relaxed which means it can be dropped

```
n_detect_relax
Inputs:   circuit $\mathcal{C}$, test set $\mathcal{T}$, $n$, fault model $\mathcal{M}$
Outputs:  relaxed test set $\mathcal{T}'$
01: fault simulate $\mathcal{T}$ based on fault model $\mathcal{M}$
02: $F$ = list of faults detected by $\mathcal{T}$
03: for each test $t_j \in \mathcal{T}$
04:     $F_j$ = list of faults detected by $t_j$
05: for each fault $f_i \in F$
06:     $\mathcal{T}_i$ = list of tests detecting $f_i$
07:     for each test $t_j \in \mathcal{T}_i$
08:         use $F_j$ to calculate $c_{ij}$
09:     for each test $t_j \in \mathcal{T}_i$
10:         calculate $G_{ij} = \sum c_{ik}, \;\; k \in \{\mathcal{T}_i - t_j\}$
11:     $\mathcal{T}_i^n = \emptyset$
12:     for $d = 1$ to $n$
13:         $\hat{G}_{im_d} = \max\{G_{ij}\}, \;\; t_j \in \{\mathcal{T}_i - \mathcal{T}_i^n\}$
14:         $\mathcal{T}_i^n = \mathcal{T}_i^n + t_{m_d}$
15:     % tests in $\mathcal{T}_i^n$ keep $f_i$, tests in $\{\mathcal{T}_i - \mathcal{T}_i^n\}$ drop $f_i$
16:     for each $t_j \in \{\mathcal{T}_i - \mathcal{T}_i^n\}$
17:         $F_j = F_j - f_i$
18:     if $F_j = \emptyset$
19:         $\mathcal{T} = \mathcal{T} - t_j$ % drop test $t_j$
20: $\mathcal{T}' = \emptyset$
21: for each test $t_j \in \mathcal{T}$
22:     generate test $t'_j$ that detects all faults in $F_j$
23:     add $t'_j$ to $\mathcal{T}'$
24: return $\mathcal{T}'$
```

Figure 6.1: Proposed n-detect relaxation algorithm

since all of the faults it used to detect are now detected by some other test(s). The fault coverage of $\mathcal{T}$ is maintained since every fault f_i is guaranteed to be detected by n different tests $t_{m_d} \in \mathcal{T}_i^n, d = 1, 2, ..., n$ with $F_{m_d} \neq \emptyset$.

Once all faults are examined, the relaxed test set $\mathcal{T}'$ is generated based on the updated fault list F_j for each test t_j that has remained in $\mathcal{T}$ (lines 20-23). Each new test $t'_j \in \mathcal{T}'$ is guaranteed to detect a subset of the faults detected by the corresponding test $t_j \in \mathcal{T}$, since the size of the updated fault list per test is reduced or, in the worst case, remains the same.

The effectiveness of the proposed method depends greatly on the ability of the test generation process (line 8 and line 22 of Figure 6.1) to derive tests with a large number of unspecified bits. Several existing methods can be used to solve this problem effectively. Both of the structural methods of [49, 48] propose specific ATPG-like routines (using implications, justifications, and testability measure concepts) to find a large test cube (test with a large number of unspecified bits) that detects a number of faults. Alternatively, the function-based framework presented in Section 4.5 can derive a large cube by extracting the shortest path in a BDD-based implementation. Any of these techniques can be used by the proposed method whose main contribution is not on this specific single test generation problem but on finding a systematic method to replace an entire test set such that the total number of specified bits is minimized.

The proposed algorithm takes $|\mathcal{T}|$ fault simulations plus, in the worst case, $|\mathcal{T}| \cdot |F| + |\mathcal{T}|$ test generations. In practice, however, the factor $|\mathcal{T}| \cdot |F|$ is much smaller since each fault $f_i \in F$ is examined only against the small number of tests in $\mathcal{T}_i \subseteq \mathcal{T}$ that detect the fault, and not for the entire test set $\mathcal{T}$.

6.5 Test Replacement Example

This section illustrates the proposed algorithm with an example. For simplicity, a 2-detect test set is considered.

Consider an initial test set $\mathcal{T} = \{t_1, t_2, t_3, t_4, t_5, t_6\}$ for a circuit C with 12 Primary Inputs that detects the 12 faults in $F = \{f_1, f_2, ..., f_{12}\}$. For this test set the n-detect fault coverage is $N(\mathcal{T}) = \frac{22}{24}$, since all faults, except f_9 and f_{12}, are detected $n = 2$ times. Table 6.2 (c) through (h) summarize the execution of the proposed algorithm and Table 6.2 (a) and (b) show the initial test set and the final test set after the relaxation process has been applied. Each pair of tables corresponds to an iteration of the algorithm, noted at the header of each table. Recall that each iteration examines one fault and so iteration 1 is for f_1, iteration 2 for f_2 and so on. At each pair, the leftmost table $($i.e. Table 6.2 (c),(e) and (g)$)$ corresponds to the gain computation step of the algorithm (lines 6-14 of Figure 6.1), while the rightmost table $($i.e. Table 6.2 (d),(f) and (h)$)$ corresponds to the fault lists updating step of the algorithm (lines 16-20 of Figure 6.1). The rows of each table correspond to the test patterns in the test set $\mathcal{T}$ showing the actual bit orientation in the test pattern. The last but one column at each table shows the faults detected by each test (i.e. the list F_j for test t_j), while the last column

Table 6.2: Test Replacement Method Example

Initial Test Set $\mathcal{T}$ after Fault Simulation													
Test Pattern													**Detects**
$\mathbf{t_1}$	1	0	1	0	1	0	1	1	0	1	0	1	$f_1, f_2, f_3, f_4, f_5, f_6$
$\mathbf{t_2}$	0	1	0	0	0	1	0	1	1	0	0	1	$f_2, f_4, f_6, f_7, f_8, f_{10}$
$\mathbf{t_3}$	1	0	0	1	0	1	1	1	1	0	1	1	$f_1, f_3, f_4, f_5, f_6, f_8$
$\mathbf{t_4}$	0	1	0	0	0	1	0	0	0	1	0	1	$f_2, f_4, f_5, f_7, f_8, f_{10}$
$\mathbf{t_5}$	1	1	1	1	0	1	1	0	0	1	0	1	$f_1, f_4, f_9, f_{11}, f_{12}$
$\mathbf{t_6}$	0	0	0	1	0	0	1	0	1	1	1	0	$f_1, f_3, f_4, f_6, f_8, f_{11}$

(a)

Relaxed Test Set $\mathcal{T}'$													
Test Pattern													**Detects**
$\mathbf{t_1}$	1	0	x	x	1	x	1	1	0	1	0	1	f_2, f_3, f_4, f_5, f_6
$\mathbf{t_2}$	0	1	x	0	x	x	x	1	1	0	0	1	$f_2, f_6, f_7, f_8, f_{10}$
$\mathbf{t_3}$	x	x	x	x	x	x	x	x	x	x	x	x	*none (can be removed)*
$\mathbf{t_4}$	0	x	x	0	0	1	0	0	x	x	x	x	f_5, f_7, f_8, f_{10}
$\mathbf{t_5}$	1	1	1	1	x	1	1	x	0	1	0	x	f_1, f_9, f_{11}, f_{12}
$\mathbf{t_6}$	0	0	x	x	0	0	x	x	x	x	1	0	f_1, f_3, f_4, f_{11}

(b)

Iteration for f_1 - Identify extra detections														
Test Pattern													**Detects**	$\mathrm{G_{1j}}$
$\mathbf{t_1}$	1	0	1	0	1	0	1	1	0	1	0	1	$\mathbf{f_1}, f_2, f_3, f_4, f_5, f_6$	7
$\mathbf{t_2}$	0	1	0	0	0	1	0	1	1	0	0	1	$f_2, f_4, f_6, f_7, f_8, f_{10}$	
$\mathbf{t_3}$	1	0	0	1	0	1	1	1	1	0	1	1	$\mathbf{f_1}, f_3, f_4, f_5, f_6, f_8$	6
$\mathbf{t_4}$	0	1	0	0	0	1	0	0	0	1	0	1	$f_2, f_4, f_5, f_7, f_8, f_{10}$	
$\mathbf{t_5}$	1	1	1	1	0	1	1	0	0	1	0	1	$\mathbf{f_1}, f_4, f_9, f_{11}, f_{12}$	$\langle 9 \rangle$
$\mathbf{t_6}$	0	0	0	1	0	0	1	0	1	1	1	0	$\mathbf{f_1}, f_3, f_4, f_6, f_8, f_{11}$	$\langle 8 \rangle$

(c)

Iteration for f_1 - Remove extra detections														
Test Pattern													**Detects**	$\mathrm{G_{1j}}$
$\mathbf{t_1}$	1	0	**x**	**x**	1	**x**	1	1	0	1	0	1	~~f_1~~, f_2, f_3, f_4, f_5, f_6	
$\mathbf{t_2}$	0	1	0	0	0	1	0	1	1	0	0	1	$f_2, f_4, f_6, f_7, f_8, f_{10}$	
$\mathbf{t_3}$	1	0	0	**x**	0	1	**x**	1	**x**	**x**	1	1	~~f_1~~, f_3, f_4, f_5, f_6, f_8	
$\mathbf{t_4}$	0	1	0	0	0	1	0	0	0	1	0	1	$f_2, f_4, f_5, f_7, f_8, f_{10}$	
$\mathbf{t_5}$	1	1	1	1	0	1	1	0	0	1	0	1	$\mathbf{f_1}, f_4, f_9, f_{11}, f_{12}$	9
$\mathbf{t_6}$	0	0	0	1	0	0	1	0	1	1	1	0	$\mathbf{f_1}, f_3, f_4, f_6, f_8, f_{11}$	8

(d)

Iteration for f_2 - Identify extra detections														
Test Pattern													**Detects**	$\mathrm{G_{2j}}$
$\mathbf{t_1}$	1	0	x	x	1	x	1	1	0	1	0	1	~~f_1~~, $\mathbf{f_2}, f_3, f_4, f_5, f_6$	$\langle 6 \rangle$
$\mathbf{t_2}$	0	1	0	0	0	1	0	1	1	0	0	1	$\mathbf{f_2}, f_4, f_6, f_7, f_8, f_{10}$	$\langle 6 \rangle$
$\mathbf{t_3}$	1	0	0	x	0	1	x	1	x	x	1	1	~~f_1~~, f_3, f_4, f_5, f_6, f_8	
$\mathbf{t_4}$	0	1	0	0	0	1	0	0	0	1	0	1	$\mathbf{f_2}, f_4, f_5, f_7, f_8, f_{10}$	4
$\mathbf{t_5}$	1	1	1	1	0	1	1	0	0	1	0	1	$f_1, f_4, f_9, f_{11}, f_{12}$	
$\mathbf{t_6}$	0	0	0	1	0	0	1	0	1	1	1	0	$f_1, f_3, f_4, f_6, f_8, f_{11}$	

(e)

Iteration for f_2 - Remove extra detections														
Test Pattern													**Detects**	$\mathrm{G_{2j}}$
$\mathbf{t_1}$	1	0	x	x	1	x	1	1	0	1	0	1	~~f_1~~, $\mathbf{f_2}, f_3, f_4, f_5, f_6$	6
$\mathbf{t_2}$	0	1	0	0	0	1	0	1	1	0	0	1	$\mathbf{f_2}, f_4, f_6, f_7, f_8, f_{10}$	6
$\mathbf{t_3}$	1	0	0	x	0	1	x	1	x	x	1	1	~~f_1~~, f_3, f_4, f_5, f_6, f_8	
$\mathbf{t_4}$	0	**x**	**x**	0	0	1	0	0	0	1	**x**	**x**	~~f_2~~, $f_4, f_5, f_7, f_8, f_{10}$	
$\mathbf{t_5}$	1	1	1	1	0	1	1	0	0	1	0	1	$f_1, f_4, f_9, f_{11}, f_{12}$	
$\mathbf{t_6}$	0	0	0	1	0	0	1	0	1	1	1	0	$f_1, f_3, f_4, f_6, f_8, f_{11}$	

(f)

•

•

•

Iteration for f_8 - Identify extra detections														
Test Pattern													**Detects**	$\mathrm{G_{8j}}$
$\mathbf{t_1}$	1	0	x	x	1	x	1	1	0	1	0	1	~~f_1~~, f_2, f_3, f_4, f_5, f_6	
$\mathbf{t_2}$	0	1	x	0	x	x	x	1	1	0	0	1	f_2, ~~f_4~~, $f_6, f_7, \mathbf{f_8}, f_{10}$	$\langle 6 \rangle$
$\mathbf{t_3}$	1	0	0	x	x	x	x	x	x	x	x	x	~~f_1, f_3, f_4, f_5, f_6~~, $\mathbf{f_8}$	6
$\mathbf{t_4}$	0	x	x	0	0	1	0	0	x	x	x	x	~~f_2, f_4~~, $f_5, f_7, \mathbf{f_8}, f_{10}$	$\langle 9 \rangle$
$\mathbf{t_5}$	1	1	1	1	x	1	1	x	0	1	0	x	f_1, ~~f_4~~, f_9, f_{11}, f_{12}	
$\mathbf{t_6}$	0	0	x	x	0	0	1	0	1	x	1	0	f_1, f_3, f_4, ~~f_6~~, $\mathbf{f_8}, f_{11}$	6

(g)

Iteration for f_8 - Remove extra detections														
Test Pattern													**Detects**	$\mathrm{G_{8j}}$
$\mathbf{t_1}$	1	0	x	x	1	x	1	1	0	1	0	1	~~f_1~~, f_2, f_3, f_4, f_5, f_6	
$\mathbf{t_2}$	0	1	x	0	x	x	x	1	1	0	0	1	f_2, ~~f_4~~, $f_6, f_7, \mathbf{f_8}, f_{10}$	6
$\mathbf{t_3}$	**x**	**x**	**x**	x	x	x	x	x	x	x	x	x	~~$f_1, f_3, f_4, f_5, f_6, f_8$~~	
$\mathbf{t_4}$	0	x	x	0	0	1	0	0	x	x	x	x	~~f_2, f_4~~, $f_5, f_7, \mathbf{f_8}, f_{10}$	9
$\mathbf{t_5}$	1	1	1	1	x	1	1	x	0	1	0	x	f_1, ~~f_4~~, f_9, f_{11}, f_{12}	
$\mathbf{t_6}$	0	0	x	x	0	0	**x**	**x**	**x**	x	1	0	f_1, f_3, f_4, ~~f_6, f_8~~, f_{11}	

(h)

reports the values of G_{ij} where applicable. A valid detection for the fault considered at each iteration is denoted by bold font, whereas a sketched fault denotes that the fault is no more detected by the corresponding test. For instance, in Table 6.2(d) fault f_1 is no more detected by test t_1, while it is detected by t_5.

The given initial test set $\mathcal{T}$ has no unspecified bits $($Table 3.2(a)$)$ and the fault simulation identifies the lists of faults F_j that are detected by each test t_j. We follow the execution of the proposed algorithm by considering the fault ordering: $f_1 < f_2 < f_3 < f_4 < f_5 < f_6 < f_7 < f_8 < f_9 < f_{10} < f_{11} < f_{12}$.

The first iteration considers the fault f_1. Using Equation 6.3 we calculate the gain in specified bits for each one of the tests t_1, t_3, t_5 and t_6, when fault f_1 is only considered in one of the corresponding lists (F_1, F_3, F_5 and F_6, respectively). Recall that G_{1j} denotes how many specified bits can be changed into unspecified if f_1 is explicitly targeted only by test t_j. According to Table 6.2(c), if f_1 is enforced to be detected by test t_1 and not by any other test that detects it (i.e. t_3, t_5, t_6), 7 specified bits can be converted into don't cares. This gain is 6 bits for t_3, 8 bits for t_5 and 9 bits for t_6. Since, $n = 2$ the algorithm selects the 2 "best" values, i.e. those that give the higher gain in specified bits, which in this case is tests t_5 and t_6. The 2 higher gains are shown in angular brackets. In Table 6.2(d), fault f_1 is removed from F_1 and F_3 so that 7 bits in the corresponding tests become unspecified, that is 3 in t_1 and 4 in t_3 shown in bold.

In the second iteration (Table 6.2 (e) and (f)), the algorithm considers fault f_2 which is detected by tests t_1, t_2 and t_4. Computing the values for the G_{2j} results in 6, 6, and 4 bits for tests t_1, t_2 and t_4, respectively. Thus, f_2 is removed from F_4 giving 4 bits that can be converted into unspecified, and which are shown in bold in Table 6.2(f). We follow the same process for faults $f_3, f_4, f_5,$ and f_6, while fault f_7 is kept in the only two lists that exists (i.e. F_2 and F_4), without any computation.

The iteration for fault f_8 identifies four tests that detect the fault, i.e., t_2, t_3, t_4 and t_6. Test t_4 is certainly one of the two "best", yet no clear decision can be made for the second best test. Any secondary decision criterion can be used, yet in our implementation we decide in favor of the first test in the test set order. Thus, fault f_8 is removed from F_3 and F_6 which gives 3 more unspecified bits in tests t_3 and t_6, respectively. The latter relaxation gives a test (i.e. t_3) what detects no faults. t_3 can be removed or not in the final test set depending on the intended application. For instance, if the application requires small application time t_3 should be removed, whereas if the targeted application demands high defect coverage t_3 should be left in the test set and all bits should be fixed appropriately.

All remaining faults, i.e., f_9, f_{10}, f_{11}, and f_{12} are detected by only 2 or 1 tests, and, thus, no further action is necessary. Recall, that, keeping all detections for a fault that is detected n or fewer times is essential, since, from the problem formulation (Section 6.3), the n-detect fault coverage should be preserved. Thus, processing these faults give no more unspecified bits and leaves the test set unchanged. In Table 6.2(b) the final relaxed test set $\mathcal{T}'$ is shown, together with the list of faults detected by each test. Observe that the $N(\mathcal{T}') = N(\mathcal{T}) = \frac{22}{24}$ since all faults are detected n times, except faults f_9 and f_{12} which are detected only once, like they do in the given test set $\mathcal{T}$. Moreover, $K(\mathcal{T}) > K(\mathcal{T}')$ since $\mathcal{T}'$ has only 38 specified bits ($K(\mathcal{T}') = 0.53$), while $\mathcal{T}$ is a fully specified test set ($K(\mathcal{T}) = 1$). Finally, the number of test patterns is the same or can be smaller (by removing t_3), depending on the targeted application. Thus, all three constraints of the problem considered have been satisfied.

6.6 Experimental Results

The proposed algorithm was implemented using ANSI C++ language, in a UNIX environment. All experiments were run on a 1GHz SunBlade 1500 with 4GB of RAM, using the full-scan versions of the ISCAS'89 benchmark circuits. The initial n-detect test sets were obtained from [97].

From the method's algorithm (Figure 6.1) we note that the test generation and the fault simulation processes are important for the proposed technique. Although, any previously proposed test generation process that produce tests with a lot of don't cares bits can be used, we use the in-house function-based tool for single stuck-at faults, based on Binary Decision Diagrams and using the package of [41] described in Section 4.5.

First we present the test set characteristics before and after applying the proposed method on the compact 10-detect test sets of [97]. Table 6.3 shows the number of Primary Inputs in Column 2, next to the circuit name. Column 3 reports the number of faults considered for each circuit. The number of faults considered for each faults was obtained after applying function-based fault equivalence rules similar to those used in [102] on top of the Checkpoint Theorem [59]. Column 4 reports the size of the initial test set $\mathcal{T}$ and Column 5 the number of specified bits in $\mathcal{T}$. Column 6 reports the n-detect fault coverage calculated using Definition 6.2. Columns 7-9 list the same information for the derived relaxed test set $\mathcal{T}'$. Moreover, the specified to total bits ratio $K(\mathcal{T}')$ (Definition 6.3), after the test relaxation is reported, in Column 10. The initial test sets are fully-specified, thus, $K(\mathcal{T}) = 1$, in all cases. Finally, Column 11 shows the time required for the proposed method, in seconds. For all circuits reported the n-detect fault coverage has been preserved. Circuits s386, s420, s1196, and s1488 the n-detect fault coverage is increased due to coincidental detections of faults that have less than n detections in the initial test set. The latter occurs since, as we mentioned before, our method performs test generation for the list of faults remained for each test and, thus, extra fault detections may arise. Clearly, the proposed method helps significantly in identifying bits that can get don't care values. The reduction in specified bits is, in most cases, around 50% and on the average is 59%, since the average $K(\mathcal{T}')$ is 1-0.41. This reduction is significant, despite the fact that the 10-detect test sets used are very compact (close to the optimal size for 10-detect). Typically, less compact test sets allow for higher specified bits reduction.

Since, to our knowledge, there is no prior work on test set relaxation for n-detect test sets we have implemented a simple test relaxation technique in order to demonstrate the effectiveness of the proposed method. Specifically, this method is a brute-force method in which each test is fault simulated and only the detected faults that have not been covered n times are used to generate a new test, with fewer specified bits, to replace the one from the original test set. Consequently, fault dropping is performed after each test replacement. In this manner, each considered test to be relaxed will no longer have to target a fault if it has already been detected n times. Table 6.4 lists the obtained results. The initial test sets $\mathcal{T}$ are the same as those used for the experiment in Table 6.3. Columns 3 and 4 list the specified to total bits ratio $K(\mathcal{T}')$ for the brute-force and the proposed approach, respectively. In all cases the proposed methodology is more effective in decreasing the number of specified bits. This demonstrates that the optimization goal targeted in the proposed approach helps in finding better sets of n (10 for this experiment) tests to target a fault such that the number of specified bits is reduced, than a straightforward approach that selects these n test sets in a brute-force manner (first n tests in the test set that detect the fault).

Table 6.3: Test Set Relaxation for 10-detect Test Sets

			Initial Test Set			After Proposed Method				
Circuit	**PIs**	**# faults**	$\lvert\mathcal{T}\rvert$	**sp. bits**	$N(\mathcal{T})$**(%)**	$\lvert\mathcal{T}'\rvert$	**sp. bits**	$N(\mathcal{T}')$**(%)**	$K(\mathcal{T}')$	**CPU(s)**
s208	18	210	271	4878	73.190	156	1050	63.524	**0.22**	1.36
s298	17	332	234	3978	100.000	234	2320	100.000	**0.58**	1.99
s344	24	334	138	3312	100.000	136	1907	100.000	**0.58**	2.32
s382	24	418	253	6072	100.000	251	2988	100.000	**0.49**	3.36
s386	13	430	201	2613	60.442	201	1903	60.535	**0.73**	3.49
s420	34	446	433	14722	58.991	224	2852	59.170	**0.19**	9.35
s510	25	572	543	13575	100.000	543	4278	100.000	**0.32**	4.01
s526	24	625	492	11808	100.000	491	6621	100.000	**0.56**	4.12
s641	54	518	227	12258	100.000	227	5997	100.000	**0.49**	8.25
s820	23	1018	949	21827	100.000	942	10270	100.000	**0.47**	9.85
s953	45	1078	766	34470	100.000	764	9442	100.000	**0.27**	13.12
s1196	32	1294	1233	39456	97.295	1131	15262	97.303	**0.39**	14.35
s1423	91	1408	269	24479	100.000	265	11974	100.000	**0.49**	13.75
s1488	14	1642	209	2926	55.164	209	2482	55.201	**0.85**	11.94
s9234	247	6960	1132	279604	100.000	1132	84656	100.000	**0.30**	232.32
s13207	700	9788	2341	1638700	100.000	2341	113449	100.000	**0.07**	325.36
s15850	611	11182	983	600613	100.000	983	114596	100.000	**0.19**	486.32
s38417	1664	31183	784	1304576	100.000	784	756652	100.000	**0.58**	912.33
				Average:	**91.394**		**Average:**	**90.874**	**0.41**	

6.6.1 Fault Ordering Effect on Relaxation

The algorithm of Figure 3.2 implies that the proposed method highly relies on the order in which the faults are examined. The reason is that the main decision on which tests must explicitly detect each fault is taken based on the gain function (Equation 6.3 and Equation 6.4) which is computed using the contribution, in specified bits, of each fault at each test (i.e. c_{ij}). This contribution changes during the relaxation process, since every fault detection that is removed from each test's list (F_j for test t_j) disturbs the bit orientation in the considered test.

For example, consider a test $t_e = \{1, 0, 1, 0, 1, 0, 1, 1\}$ that detects 3 faults f_x, f_y, and f_z (i.e., $F_e = \{f_x, f_y, f_z\}$), shown in Table 6.5. The contributions c_{xe}, c_{ye}, and c_{ze} is 2, 1, and 2 specified bits, respectively. Specifically, assume that by removing f_x the first two bits become don't care, whereas removing f_y the fifth bit of the test becomes of unspecified value. Removing f_z results in two don't cares bits, i.e., the last two bits of the test (bits 7

Table 6.4: Comparing with a Brute-Force Technique

Circuit	$\|\mathcal{T}\|$	**Brute-force** $K(\mathcal{T}')$	**Proposed** $K(\mathcal{T}')$
s208	271	0.48	0.22
s298	234	0.82	0.58
s344	138	0.72	0.58
s382	253	0.89	0.49
s386	201	0.91	0.73
s420	433	0.35	0.19
s510	543	0.67	0.32
s526	492	0.74	0.56
s641	227	0.78	0.49
s820	949	0.66	0.47
s953	766	0.45	0.27
s1196	1233	0.54	0.39
s1423	269	0.69	0.49
s1488	209	0.91	0.85
s9234	1132	0.57	0.30
s13207	2341	0.69	0.58
s15850	983	0.38	0.19
s38417	784	0.72	0.58

and 8). Bits 3 and 4 become don't cares if both faults f_x and f_y are removed from F_e and bit 6 becomes don't care if both faults f_y and f_z are removed from F_e. Let us concentrate on the contribution of fault f_x and assume that this fault is examined second. If the algorithm examines f_z first and the decision is to remove f_z from F_e then the contribution of fault f_x remains 2, as it is shown in the first double-row of Table 6.5. If the algorithm examines fault f_y first and removes it form F_e, then the contribution of the fault becomes $c_{xe} = 4$ (second double-row of Table 6.5). When the decision on fault f_x is to be made, the outcome may change depending on which of f_y and f_z has been examined first. Table 6.5 summarizes the changes in the faults contribution for different orderings. For real circuits, the faults contribution change range can become very large, especially with n-detect test sets where the average number of detections are larger and, thus, each test's list of detected faults is larger. The latter implies that the ordering of the examination of the considered faults has a great effect on the final ratio of specified bits in the test set.

Table 6.5: Fault Contribution for Different Fault Orderings

Ordering	**Fault Removed**	**Test Pattern**								c_{xe}	c_{ye}	c_{ze}	**Sp. bits in t_e:**
none	none	1	0	1	0	1	0	1	1	**2**	**1**	**2**	**8**
$f_z < f_x < f_y$	f_z	1	0	1	0	1	0	x	x	**2**	**2**	**-**	**6**
	f_x	x	x	1	0	1	0	x	x	**-**	**4**	**-**	**4**
$f_y < f_x < f_z$	f_y	1	0	1	0	x	0	1	1	**4**	**-**	**3**	**7**
	f_x	x	x	x	x	x	0	1	1	**-**	**-**	**3**	**3**
$f_x < f_y < f_z$	f_x	x	x	1	0	1	0	1	1	**-**	**3**	**2**	**6**
	f_y	x	x	x	x	x	0	1	1	**-**	**-**	**3**	**3**
$f_z < f_y < f_x$	f_z	1	0	1	0	1	0	x	x	**2**	**2**	**-**	**6**
	f_y	1	0	1	0	x	x	x	x	**4**	**-**	**-**	**4**
$f_y < f_z < f_x$	f_y	1	0	1	0	x	0	1	1	**4**	**-**	**3**	**7**
	f_z	1	0	1	0	x	x	x	x	**4**	**-**	**-**	**4**
$f_x < f_z < f_y$	f_x	x	x	1	0	1	0	1	1	**-**	**3**	**2**	**6**
	f_z	x	x	1	0	1	0	x	x	**-**	**4**	**-**	**4**

Next, we give experimental results for the proposed test relaxation method under a number of different fault orderings. Sorting the faults under the different criteria can either be done once before the application of the algorithm of the proposed method, or can be updated dynamically during the execution of the algorithm. We have experimented using the first method which keeps the execution time small. Specifically, our experimentation investigates five different fault orderings:

i. Follow topological order of faults, i.e., examine faults closer to the primary inputs first.

ii. Consider faults with more tests detecting them first, i.e., consider faults with more minterms in the corresponding test functions, first.

iii. Consider faults with fewer tests detecting them first, i.e., consider faults with less minterms in the corresponding test functions, first.

iv. Consider faults with more tests in the initial test detecting them, first.

v. Consider faults with fewer tests in the initial test detecting them, first.

Table 6.6: Test Relaxation using Different Fault Orderings

Circuit	**PIs**	**Initial Test Set**			**Topological Order**			**More Minterms First**		
		Tests	Sp. Bits	BCE+ (%)	Tests	Sp. Bits	BCE+ (%)	Tests	Sp. Bits	BCE+ (%)
s208	18	271	4878	92.688	177	1100	90.012	156	1052	89.516
s298	17	234	3978	99.595	234	2340	99.486	234	2329	99.582
s344	24	138	3312	98.643	138	1985	98.212	137	1923	98.133
s382	24	253	6072	99.722	253	3169	99.594	251	**2988**	99.515
s386	13	201	2613	94.471	201	**1903**	94.479	201	1904	94.479
s420	34	433	14722	85.641	243	3008	85.214	225	2862	84.972
s510	25	543	13575	98.383	542	4304	98.210	542	4291	98.141
s526	24	492	11808	99.382	492	6907	99.251	491	6668	99.124
s641	54	227	12258	99.125	226	6208	98.427	227	**5997**	98.350
s820	23	949	21827	99.959	948	10466	99.950	942	10285	99.948
s953	45	766	34470	90.938	766	9620	89.746	766	9469	89.745
s1196	32	1233	39456	96.564	1155	15715	95.867	1133	15299	95.890
s1423	91	269	24479	97.374	269	12755	96.198	269	12135	96.186
s1488	14	209	2926	89.162	209	2492	89.112	209	2483	89.161
s9234	247	1132	279604	99.058	1131	85273	88.430	1132	**84656**	88.302
s13207	700	2341	1638700	90.784	2341	114173	88.994	2341	**113449**	88.664
s15850	611	983	600613	92.798	983	114712	90.321	983	114618	90.331
s38417	1664	784	1304576	98.789	784	757011	87.413	784	756852	87.143

(a)

Circuit	**PIs**	**Fewer Minterms First**			**More Test Detecting it first**			**Fewer Tests Detecting it First**		
		Tests	Sp. Bits	BCE+ (%)	Tests	Sp. Bits	BCE+ (%)	Tests	Sp. Bits	BCE+ (%)
s208	18	195	1146	90.214	156	**1050**	89.527	206	1195	90.311
s298	17	234	2373	99.581	234	**2320**	99.579	234	2358	99.580
s344	24	137	1975	98.251	136	**1907**	98.175	138	1985	98.312
s382	24	253	3237	99.612	251	2993	99.521	253	3228	99.595
s386	13	201	1905	94.479	201	1904	94.479	201	1905	94.479
s420	34	275	3271	85.121	224	**2852**	84.965	286	3306	85.384
s510	25	543	4320	98.201	542	4281	98.134	543	4327	98.208
s526	24	492	7006	99.312	491	**6621**	58.346	492	6976	98.152
s641	54	226	6217	98.544	226	6001	98.483	226	6187	98.519
s820	23	949	10573	99.949	942	**10270**	99.949	949	10558	99.950
s953	45	765	9980	89.787	764	**9442**	89.745	766	9861	89.753
s1196	32	1180	16153	95.911	1131	**15262**	95.899	1178	16055	95.921
s1423	91	269	12856	96.198	265	**11974**	96.194	269	12918	96.201
s1488	14	209	2505	89.183	209	**2482**	89.178	209	2510	89.184
s9234	247	1132	85436	88.491	1132	84933	88.386	1132	85764	88.615
s13207	700	2340	113721	88.830	2341	113488	88.818	2341	114001	88.936
s15850	611	983	114811	90.242	983	**114596**	90.234	982	114686	90.239
s38417	1664	783	756768	87.186	784	**756652**	87.087	783	756822	87.217

(b)

Table 6.7: Test Relaxation using Different Fault Orderings (essential faults first)

Circuit	PIs	**Initial Test Set**			**Topological Order**			**More Minterms First**		
		Tests	Sp. Bits	BCE+ (%)	Tests	Sp. Bits	BCE+ (%)	Tests	Sp. Bits	BCE+ (%)
s208	18	271	4878	92.688	178	1125	90.012	157	1057	89.516
s298	17	234	3978	99.595	234	2348	99.486	234	2333	99.582
s344	24	138	3312	98.643	138	1991	98.212	138	1935	98.133
s382	24	253	6072	99.722	253	3195	99.594	252	3012	99.515
s386	13	201	2613	94.471	201	1905	94.479	201	1905	94.479
s420	34	433	14722	85.641	243	3015	85.214	226	2931	84.972
s510	25	543	13575	98.383	542	4358	98.210	542	4302	98.141
s526	24	492	11808	99.382	492	7001	99.251	491	6702	99.124
s641	54	227	12258	99.125	226	6211	98.427	227	6004	98.534
s820	23	949	21827	99.949	948	10502	99.948	942	10320	99.948
s953	45	766	34470	90.938	766	9702	89.846	766	9511	89.773
s1196	32	1233	39456	96.564	1154	15680	95.877	1133	15320	95.890
s1423	91	269	24479	97.374	269	12854	96.203	269	12245	96.194
s1488	14	209	2926	89.162	209	2511	89.188	209	2503	89.193
s9234	247	1132	279604	90.058	1132	84927	88.322	1132	85002	88.421
s13207	700	2341	1638700	90.784	2340	114012	89.041	2341	113781	88.731
s15850	611	983	600613	92.798	983	114671	90.251	983	114681	90.257
s38417	1664	784	1304576	98.789	784	756799	87.255	784	756724	87.192

(c)

Circuit	PIs	**Fewer Minterms First**			**More Test Detecting it first**			**Fewer Tests Detecting it First**		
		Tests	Sp. Bits	BCE+ (%)	Tests	Sp. Bits	BCE+ (%)	Tests	Sp. Bits	BCE+ (%)
s208	18	195	1151	90.223	175	1095	89.518	168	1073	90.320
s298	17	234	2421	99.591	234	2357	99.569	234	2325	99.570
s344	24	137	1977	98.261	138	1930	98.165	136	1931	98.322
s382	24	253	3241	99.622	253	3051	99.511	251	3119	99.585
s386	13	201	1905	94.479	201	1904	94.470	201	1904	94.488
s420	34	275	3299	85.130	224	2891	84.957	236	2931	85.393
s510	25	543	4403	98.211	543	**4278**	98.120	542	4335	98.218
s526	24	492	7012	99.322	492	6708	58.340	491	6812	98.142
s641	54	226	6258	98.649	227	6075	98.414	226	6040	98.521
s820	23	949	10580	99.948	949	10445	99.949	943	10446	99.950
s953	45	765	9997	99.836	766	9507	99.803	764	9563	99.811
s1196	32	1181	16192	95.976	1152	15581	95.889	1131	15408	95.915
s1423	91	269	12906	96.269	269	12232	96.056	265	12465	96.116
s1488	14	209	2509	89.169	209	2502	89.169	209	2485	89.157
s9234	247	1132	85554	88.429	1132	84711	88.342	1132	85281	88.496
s13207	700	2341	113851	88.838	2341	113563	88.719	2341	113982	88.739
s15850	611	983	114760	90.313	982	114842	90.355	983	114691	90.292
s38417	1664	784	756801	67.265	783	756742	67.118	784	756912	67.309

(d)

Orderings (iv) and (v) actually sort faults based on the number of tests that detect them on the initial test set. This information can be easily obtained by the fault simulation procedure preceding the main algorithm application. Ordering (i) examines faults based on the order the corresponding circuit lines are visited during on a topological traversal of the circuit's graph. A topological order can be obtained in linear, to the size of the circuit's graph, time [103]. Orderings (ii) and (iii) are different from (iv) and (v), since they consider all tests that detect the corresponding fault obtained by a test generation process. Since, in our experimentation the test generation is carried out by the function-based framework presented in Chapter 3, we are able to obtained accurate information on the total number of tests that exist for each fault, efficiently. The latter can be done by counting the number of minterms in the test function corresponding to each fault. Recall that we use a BDD-based function implementation and, thus, counting the function's minterms is a linear operation on the diagram's size.

Alternatively, if the test generation process used does not allow efficient retrieval of such information, a limited number of test generation queries for each fault can be made, in order to classify faults according to criteria (ii) and (iii). The motivation for this classification is an attempt to sort faults depending on how "difficult" is to be detected. Thus, orderings (ii) and (iv) give priority to easy-to-detect faults, while, ordering (iii) and (v) favor hard-to-detect faults. Intuitively, we expect that orderings (ii) and (iii) will give more accurate classification of faults for this criterion.

Table 6.6 (a) and (b) report experimental results for all five different orderings. For each ordering the relaxed test set size, the number of specified bits in the relaxed test set and the bridging fault coverage estimation (BCE+) are reported. Table Table 6.6(a) reports results for orderings (i) and (ii) in Columns 6-8 and 9-11, respectively, while Columns 3-5 report this information for the initial test set. Table 6.6(b) provides results for orderings (iii), (iv) and (v) in Columns 3-6 , 6-8, and 9-11, respectively. The same information is reported in Table 6.7 (a) and (b) when the orderings considered the essential faults first, i.e., those faults that have fewer than n or exactly n detections. The best results for all orderings are shown in boldfaced font. Observe that there is a clear advantage for ordering (iv) i.e., when considering faults with more tests in test set $\mathcal{T}$, first. In all but six circuits the most relaxed test sets are those where the faults are examined based on this rationale. Nevertheless, our method allows applying two or three different ordering with small increase on the CPU time and keep the best results in terms of specified bits. Considering essential faults first does not give better results, thus, one can only focus on orderings (ii) and (iv) in order to get the best relaxation results. Both these orderings try to “accommodate” easy-to-detect faults first, in the tests considered. While ordering (ii) is most accurate than ordering (iv) the best results are obtained by the latter ordering, in most of the cases, while in the cases while (ii) is best the difference in specified bits is not large. This observation eliminates the need for considering ordering (ii) when no accurate information on the number of tests for each fault is available, if for instance, a structure-based test generation framework is used.

6.6.2 Random Fixing of Unspecified Bits

Table 6.8 shows experimental results justifying that random bit fixing restores the average detection parameter as well as the defect fault coverage. We have consider *Bridging Faults* to represent defects, using the standard non-feedback bridging fault model [104]. After the circuit name and the number of primary inputs for each one of the circuits, we report experimentally obtained information about three different n-detect test sets. The first test set (Columns 3-6) is the initial test set, the second one (Columns 7-10) is the relaxed test set obtained by the proposed method and the third test set (Columns 11-14) is after random bit fixing. For the bit fixing we have used a 0-1 random generator with uniform distribution and have fixed all the unspecified bits, even in those tests that after the relaxation process detect no faults and could be removed from the test set.

For each of these test sets we report the size of the test set (Columns 3,7, and 11), its estimated bridging fault coverage (Columns 4 and 5, 8 and 9, 12 and 13), and its average detection (Columns 6, 10, and 14). The bridging fault coverage was estimated using two different estimators. The bridging fault coverage under the Columns named BCE+ (Columns 5, 9 and 13) were calculated using the estimator proposed in [105] which takes into consideration the number of detections of each fault and the probabilities for each line to have the logic one or the logic zero value. [105] shows that this estimator gives very good approximation of the actual bridging fault coverage. Yet, in Columns 4, 8 and 11 we also provide a more commonly used estimator (i.e., BCE), proposed in [106], for completeness.

The average detections in the obtained test set ($AD(\mathcal{T}')$) has dropped, as expected, after the relaxation, in all but three circuits (s298, s386 and s1488). These circuits have the lower reduction (42 %, 27 % and 15 %, respectively). The latter two have $AD(\mathcal{T}') < 10$ since there exist a lot of faults with lower than 10 different tests and only few with more than 10.

We observe that the drop in $AD(\mathcal{T}')$ is, most of the times, analogous to $K(\mathcal{T}')$. In other words, the higher the reduction in specified bits (lower $K(\mathcal{T}')$), the higher is the $AD(\mathcal{T}')$ drop. This is inherent to the test relaxation process, as explained in Section 6.2.

The major observation, here, is that the average detection is tightly correlated with the bridging fault coverage. When average detection decreases, due to the relaxation process, the bridging fault coverage also decreases and in all but two cases (s208, s420) the decrease is proportional for these two measures. Since an advantage of n-detect test sets is the detection of defects and faults not explicitly targeted, it could be argued that reducing the average fault detections may cancel this advantage. Essentially, all the fixed value bits in the unrelaxed test set $\mathcal{T}$ contribute to the detection of non-targeted faults and defects as it is concluded from the bridging fault coverages before and after relaxation. Those of the test set bits that are converted to don't cares during the relaxation process remove their contribution to this extra detection property. By observing the bridging fault coverage for the relaxed test set after applying random bit-fixing, this is not problem that remains during test application where all bits are fixed. The latter confirms that although our method removes additional modeled fault coverages (i.e., beyond n-detect), non-targeted fault and defect coverages are maintained or can be even increased, if bit fixing is applied. Despite the fact that the relaxation process affects the values for both the average detection and the bridging fault coverage, after random bit fixing the values for average detection, as well as for the bridging fault coverage, are restored to the same level as in the initial test set $\mathcal{T}$. Thus, experimentation shows that the statement made in Section 6.2 assuming that the test relaxation process has no catastrophic effect on the defect detection ability of n-detect test sets, is very realistic. Since all applications that benefit from having relaxed test sets as input, will fix the don't care bits prior to the test application, the average fault detections for test set $\mathcal{T}'$ will eventually increase to that of $\mathcal{T}$.

6.7 Conclusions

In this chapter we have investigated the impact of test set relaxation in n-detect test sets. We presented a systematic methodology for decreasing the number of specified bits in a given n-detect test set. The motivation comes from the observation that the specified bits in an

Table 6.8: Random Unspecified Bits Fixing Effect

		Initial Test Set				After Proposed Method				After Random Bit Fixing			
Circuit	**PIs**	$\lvert\mathcal{T}\rvert$	**BCE (%)**	**BCE+ (%)**	$AD(\mathcal{T}')$	$\lvert\mathcal{T}'\rvert$	**BCE$'$ (%)**	**BCE+$'$ (%)**	$AD(\mathcal{T}')$	$\lvert\mathcal{T}''\rvert$	**BCE$''$ (%)**	**BCE+$''$ (%)**	$AD(\mathcal{T}'')$
s208	18	271	97.188	92.688	**37.45**	156	95.346	89.527	**11.18**	271	96.012	91.434	**29.16**
s298	17	234	99.950	99.595	**25.57**	234	99.946	99.579	**25.21**	234	99.959	99.661	**34.32**
s344	24	138	99.973	98.643	**29.92**	136	99.951	98.175	**22.31**	138	99.964	98.475	**30.30**
s382	24	253	99.979	99.722	**38.22**	251	99.953	99.515	**25.34**	253	99.964	99.880	**38.29**
s386	13	201	94.752	94.471	**9.88**	201	94.758	94.479	**9.41**	201	94.813	94.535	**9.82**
s420	34	433	93.917	85.641	**54.14**	224	93.903	84.965	**18.23**	433	96.410	88.214	**54.92**
s510	25	543	99.976	98.383	**57.17**	543	99.972	98.134	**51.34**	543	99.976	98.413	**57.11**
s526	24	492	99.984	99.382	**60.07**	491	99.958	99.110	**39.1**	492	99.968	99.258	**60.38**
s641	54	227	99.980	99.125	**45.01**	227	99.944	98.350	**32.22**	227	99.968	98.865	**43.40**
s820	23	949	99.959	99.959	**31.92**	942	99.956	99.949	**29.67**	949	99.956	99.956	**31.71**
s953	45	766	99.979	90.938	**112.21**	764	99.978	89.745	**79.59**	766	99.978	90.903	**111.27**
s1196	32	1233	99.814	96.564	**93.3**	1131	99.805	95.899	**63.12**	1233	99.841	99.709	**94.97**
s1423	91	269	99.991	97.374	**48.82**	265	99.964	96.194	**26.12**	269	99.990	97.359	**47.70**
s1488	14	209	90.727	89.162	**8.93**	209	90.744	89.178	**8.15**	209	90.760	89.202	**8.95**
s9234	247	1132	99.985	90.058	**148.28**	1132	99.969	88.302	**60.51**	1132	99.980	89.462	**167.42**
s13207	700	2341	99.984	90.784	**354.22**	2341	99.960	88.664	**62.89**	2341	99.974	89.705	**413.97**
s15850	611	983	99.991	92.978	**156.52**	983	99.969	90.234	**48.41**	983	99.982	91.722	**172.15**
s38417	1664	784	99.981	98.789	**67.12**	784	98.212	87.087	**38.34**	784	99.982	98.804	**67.84**

n-detect test set adds extra detections for faults that are easy to detect. The actual number of detections for these faults in these test sets is much larger than n and these extra detections can be removed, by un-fixing the values for some test bits. This process makes the test sets flexible and, thus, suitable for a number of applications that appropriately fix the values for the don't care test bits. The experimental results reported demonstrate the effectiveness of the proposed method in achieving high specified bit reduction rates in n-detect test sets, while maintaining the n-detect fault coverage. Provided discussion also explains how the defect coverage and non-targeted fault coverage of the relaxed test sets will be similar to that of the initial test sets, when the relaxed test sets are fully-specified before test application.

CHAPTER 7

GENERATING INCREASED QUALITY n-DETECT TEST SETS VIA FAULT CONE PARTITIONING

7.1 Introduction

In the previous chapter we have argued that test sets targeting each modeled fault more that once are beneficial for targeting non-modeled faults and emerging defects. The rationale behind detecting a fault more than one time is to achieve higher quality in terms of defect coverage, by generating a number of *different* tests for each modeled fault. Test sets that detect each fault multiple times or with at least n different tests have been shown to give high non-modeled fault coverage [90, 91, 92, 93, 94, 95, 97].

Most of the ATPG methods for multiple-detect or n-detect test sets concentrate, mainly, on reducing the test generation effort while keeping the overall test set size small. Another important issue addressed in the literature is the diversity of the different generated tests per fault. While test generation is generally done with well defined techniques, in an iterative manner, the main target is to quantify the difference between the tests that target the same fault and, thus, generate as diverse tests as possible for each modeled fault. A number of previous works address the issue of how different the n tests for each fault should be, either by changing the test set characteristics [96, 97], or by using structure-based metrics [89, 94, 107, 108]. This is because a test set with diverse tests per targeted modeled fault has been shown to increase the defect and non-modeled fault coverage of the test set [96, 107, 108, 109]. Diversity, i.e., how different are the tests that detect a fault, has been defined in various ways. [96] proposed a definition for *sufficiently different* tests, in terms of how different certain primary input signal values are with respect to the already generated tests for a fault. The works in [96, 108, 109] introduce measures that quantify the difference between tests detecting the same fault, based either on the internal signal values that excite the fault [96, 108] or on propagating the fault to a primary output [109]. In essence, the motivation behind all these methods is to find tests that activate the fault in different ways (internal signal values) and propagate the fault effect to some primary output via different propagation paths. They attempt to achieve that by incorporating randomness or some efficient brute-force heuristic in the test pattern generation process. Hence, none of the existing methods that generate diverse tests examines activation and/or propagation paths. As a result, the number (and different constituent circuit lines) of the different activation/propagation paths for a fault cannot be guaranteed.

The work in this chapter is motivated by previous work on the need for generating diverse tests for increasing the quality, in terms of non-modeled fault coverage, of an n-detect test set. Since, from the work presented in Chapter 2, we have observed that differentiating the propagation paths increase the quality of the test sets, we follow the same rationale here, for introducing diversity for n-detect test sets. Thus, we propose a methodology for generating n-detect test sets such that *each of the n tests detecting a fault is guaranteed to propagate the fault via a different propagation path than the remaining n-1 tests*. In the case where only $k < n$ such tests exist for a fault (the proposed method can efficiently determine this) n tests are still generated in order to maintain 100% n-detect fault coverage. However, two or more tests that propagate a fault through the same propagation path can still differ since the proposed method can work in a complementary manner to the methods of [96, 108] to generate tests with different excitation conditions or sufficiently different test pattern values.

Essentially, the proposed method gives a systematic way of partitioning the circuit under test into non-overlapping *propagation subcircuits*. For each fault a number of such subcircuits is obtained, each of which contains a number of paths that allow the propagation of the fault to at least one primary output. Each propagation subcircuit is guaranteed to contain at least one different propagation path from any other propagation subcircuit for the same fault. The proposed algorithm identifies the circuit cone starting from each fault site towards the primary outputs. Then, it performs a breadth-first traversal on the cone and groups the propagation paths (paths segments) in propagation subcircuits, based the cone's fanout branches. The process accommodates paths in different propagation subcircuits as soon as they are identified to be distinct cone's stem. This decision criterion not only ensures non-overlapping propagation subcircuits, but includes the longest paths from the fault site to some primary output in different subcircuits for the fault under consideration. For each fault, n different subcircuits are obtained. For faults with only $k < n$ propagation paths, k subcircuits are obtained with each subcircuit being a physical path segment.

Next, the algorithm generates test functions for each one of the faults, for all the subcircuits obtained. Thus, the test space for each fault is partitioned into n (or k when less than n propagation paths exist) groups which generally contain different tests. An undirected graph is then constructed where each graph vertex corresponds to a test group. The proposed algorithm iterates on the graph and merge compatible vertices together in a bottom-up rationale. For this work we use a modified definition of test compatibility where two test groups corresponding to the same fault are not considered as compatible, in order to avoid merging the n different tests for the same fault and, consequently, destroy the n-detect property.

The main contributions of the method are:

- In contrast to existing methods, the propagation of a fault via different paths (if these exist) is *guaranteed*.
- The different propagation paths have as little overlap (common circuit lines) as possible.
- Path and path segment enumeration is *explicitly avoided*, allowing the method to be efficient and scalable.
- The methods of [96, 108] can be used on-top of the proposed method. Hence, each of the n generated tests is guaranteed to propagate the fault via a different propagation path *and* with different internal signal values than any of the other n-1 tests.
- The method can be easily generalized to apply to any other fault model (other than the stuck-at fault model) that is linear to the size of the circuit. In particular, for dynamic models such as the transition delay fault model, the method can guarantee different activation paths on top of the n different propagation paths.

While the proposed methodology is presented in this context for the stuck-at fault model, it can be generalized to any fault model that is linear to the size of the circuit. For instance, the transition fault model can be used in the same manner. Specifically, for the transition fault model, the subcircuit partitioning can be done not only for propagation, but also for activation in order to obtain more diversity in the n different tests for the same fault. Appropriate sensitization criteria can be used in both activation and propagation in the same way as they have been used in Chapter 2.

The rest of the chapter is organized as follows. Section 7.2 describes the process for partitioning a circuit under test into n *propagation subcircuits* per fault. Section 7.3 presents the test generation methodology. Section 7.4 presents and discusses the obtained experimental results which demonstrate the effectiveness of the proposed methodology in terms of increasing the number of different fault propagation paths. The impact on defect coverage (using the bridging fault model as surrogate) is also reported. Section 7.5 concludes the work in this chapter.

7.2 Partitioning the Fault Site Cone into Propagation Subcircuits

The significant difference in this work in terms of quality is that the proposed methodology ensures as much diversity as possible between the different tests targeting the same fault. In order to achieve this we perform test generation for different parts of the circuit for every fault. This way we enforce implicit partitioning of the test space for each fault based on a fault propagation criterion. An important role in the proposed methodology is accorder to the partitioning process, described in this section.

Each fault defines a circuit cone which starts at the fault site and terminates at the primary outputs driven by the fault site. In order to find n different tests, each detecting the fault via different propagation paths, the proposed method partitions the propagation paths into n groups and generates a test that detects the fault via at least one propagation path per group (the latter is discussed in detail in Section 7.3). In this section we present how these n groups of different propagation paths can be derived in linear to the circuit size time, without performing any path enumeration.

We first give some necessary definitions used later in this section.

Definition 7.1. The **fault cone** of line l_i is the part of the circuit originating from line l_i which includes all physical path segments from line l_i (all circuit lines driven by line l_i up to the primary outputs).

Definition 7.2. A **propagation path** is a physical path segment from a circuit line l_i to a primary output which, under an appropriate value assignment at the primary inputs, allows the propagation of the effect of a fault at line l_i to a primary output.

Definition 7.3. A **propagation subcircuit** for a fault at line l_i is a subpart of the fault cone for l_i that contains at least one propagation path. Any non-redundant fault has at least one such propagation subcircuit.

For any propagation subcircuit of a stuck-at fault at line l_i, at least one primary input assignment exists that allows the propagation of the fault effect through at least one propagation path of the propagation subcircuit. Any such assignment is a valid test for the stuck-at fault at line l_i. Moreover, it enforces propagation of the fault at least through the propagation paths that form the propagation subcircuit. The idea of the proposed algorithm is to generate the n different tests for each fault for n different propagation subcircuits corresponding to that fault. If only $k < n$ propagation subcircuits exist then the proposed method uses multiple copies of these subcircuits and tries to differentiate the tests using unspecified bits value fixing.

The algorithm of Figure 7.1 describes the proposed decomposition of the fault cone into n propagation subcircuits for a given fault site[1]. The algorithm performs a breadth-first-search-like traversal of the circuit to identify the n fanout branches that are closer (minimum distance) to the fault site but to not contain same path segments. Each of these branches is then used along with the fault site, to define a propagation subcircuit. The traversal starts at the fault site f, using a First-In-First-Out queue. For each line that is removed from the queue all the successor lines are inserted in the queue. If the line under examination is a fanout branch, then it is kept in a separate list B (lines 07-08). Moreover, if this branch is driven by some other branch q' in the fault cone (which is already in B), q' is removed from

[1]In this section, the paths in an identified subcircuit correspond to physical path segments, instead of propagation paths (as in Definition 7.2). This requires ATPG to be determined, and is examined in the next section. For simplicity, we still refer to propagation subcircuits here.

```
find_propagation_subcircuits($C, n, f$)
Inputs:   circuit $C$, $n$, fault site $f$
Outputs:  set of propagation subcircuits $PS$
01: queue $Q = \emptyset$
02: lists $L = \emptyset$,$B = \emptyset$
03: insert $f$ at the end of $Q$
04: do
05:     $q$ = first element of $Q$
06:     remove $q$ from $Q$
07:     if $q$ is a fanout branch
08:         insert $q$ in $B$
09:         if $B$ contains branch $q'$ that drives branch $q$
10:             remove $q'$ from $B$
11:     $L$ = successor lines of $q$ in $C$
12:     for each line $l \in L$
13:         insert $l$ at the end of $Q$
14: until $Q$ is empty OR size of $B$ = $n$
15: for each branch line $q$ in $B$
16:     $ps$ = subcircuit defined by lines $f$ and $q$
17:     save $ps$ in $PS$
18: end if
19: return $PS$
```

Figure 7.1: Proposed Fault Cone Decomposition

B. This ensures that no two propagation subcircuits will contain identical path segments (starting a the fault site). The traversal terminates when n such fanout branches are identified or when the primary outputs are reached.

The second step of this algorithm (lines 15-20) derives the n propagation subcircuits using the branch lines included in B. For some line q in B, the propagation subcircuit is defined as the part of the circuit that includes all path segments starting at the fault site f and passing through the branch line q. Since no two branch lines that can be on the same path are included in B, every generated subcircuit contains different path segments. Moreover, the path segments included in a subcircuit are guaranteed to have minimum overlap (in terms of circuit lines) with the segments of other subcircuits, since the algorithm considers the n appropriate branch lines that are closer to the fault site f. Thus, the number of lines between the fault site f and the fanout stem of two branches q and q' in B (which is the common lines between path segments in the subcircuits derived by q and q') is minimized.

In the case where the number of path segments from the fault site f is $k < n$, the algorithm returns k propagation subcircuits, each containing a single path segment. For $k > n$, the generated subcircuit may contain more than one path segments. For the examined problem, there is no need to have balanced subcircuits (in terms of the number of the path segments) since propagating a fault through at least one path segment in each propagation subcircuit suffices to derive the desired n-detect test set. This method can be easily extended to the transition fault model for which activation path diversity is desired, in addition to propagation path diversity. The activation subcircuits can be obtained using the algorithm of Figure 7.1, traversing the circuit from the fault site towards the primary inputs and keeping gate inputs in list B, instead of fanout branches.

7.3 Test Generation Methodology

After partitioning the graph into n (if possible) subcircuits, we perform test generation for each one of those subcircuits. The target here is to have n tests that propagate the fault via n different paths in order to achieve as much diversity as possible. The algorithm here is again graph-theoretic like the one presented in Chapter 4 and takes special consideration for preserving both the n-detect fault coverage and the difference on the propagation paths.

The proposed test generation procedure is function-based and starts by generating one test function per subcircuit identified in Section 7.2. Hence, it will generate n different test functions per fault. A test function in this case is derived by only considering the circuit cone from the primary inputs to the fault site (fault activation subcircuit) and one propagation subcircuit, instead of the entire circuit that is considered in traditional function-based ATPG like that of Section 4.5. Figure 7.2 shows this first step of the proposed test generation method.

For each fault the fault cone partitioning procedure of Figure 7.1 is invoked in order to get n different subcircuits per fault (line 02). Then, for each subcircuit, a test function is generated that implicitly holds all the tests that allow the propagation of the fault effect via any path segment in the subcircuit (line 04). It is possible that a test function is the constant zero (0) function, implying that no tests exist that can propagate the fault effect via one of the path segments in a subcircuit, i.e., the fault is redundant when considering the particular subcircuit. In this case the propagation subcircuit decomposition procedure of Section 7.2

```
generate_n_test_functions($C, n, F$)
Inputs:   circuit $C$, $n$, fault list $F$
Outputs:  $n$ test functions for each fault in $F$
01: for each fault $f_i$ in $F$
02:    $PS$ = find_propagation_subcircuits($C, n, f_i$)
03:    for each subcircuit $ps_j \in PS$
04:       $tf_{ij}$ = generate test function for $f_i$ propagating via $ps_j$
05:       insert $tf_{ij}$ in $TF$
06:    # replace subcircuits in $PS$ containing no propagation paths
07:    $k = |PS|$
08:    if $k == n$
09:       while $\exists\ tf_{ij} = 0$
10:          $k = k + 1$
11:          $PS_t$=find_propagation_subcircuits($C, k, f_i$)
12:          for each $ps_j \in PS_t - PS$
13:             $tf_{ij}$=generate test function for $f_i$ under $ps_j$
14:             insert $tf_{ij}$ in $TF$
15:       end while
16:    end if
17:    else
18:       for $z = k$ to $n$
19:          $tf_{iz}$ = $tf \in TF$ with $z$th larger number of minterms
20:    end else
21: return $TF$
```

Figure 7.2: Generating n test functions for each fault

is called again, until n different subcircuits with non-zero test functions are derived (lines 07-16). In this case the algorithm of Section 7.2 is specifically guided to ignore the part of the circuit corresponding to subcircuits already determined as undesired and concentrate on the remaining part of the circuit (this is not shown in Figure 7.2 due to space limitations). Finally, if fewer than n propagation paths exist (and thus, fewer than n test functions), the already obtained test functions are used in order to cover the extra detections (lines 18-20). Test functions with a higher number of minterms (tests) are preferred for this purpose.

The procedure of Figure 7.3 describes a graph based approach that allows for the generation of propagation path diverse tests that detect a large number of faults, for reducing the test size, without destructing the n-detect test coverage. After generating n different test functions per fault using the algorithm of Figure 7.2, an undirected graph G is constructed, where each vertex in the graph corresponds to one test function (set V) and an edge on the graph denotes compatibility (or not) between two test functions (set E). There are two kind of edges in our graph: weighted edges and barriers. The weighted edges between two vertices denote compatibility, i.e., that the two set of tests implicitly represented by the corresponding test functions can be reduced into a single set of tests since they include common tests that can detect all faults detected by both test functions (lines 08-15). The weight of an edge denotes the number of specified bits in the largest cube (partially specified test pattern) in the intersection of the two set of tests corresponding to the edges' endpoints. The process of combining minimum edge vertices in a compatibility graph has been shown to produce compact test sets, in the case where single detection is desired for stuck-at faults (see Chapter 4. The barriers are essentially non-edges (or edges with infinite weight) and denote that the vertices at their endpoints can never be combined, because this will reduce the number

```
*barrier_compatibility_merging*
**Inputs**:   circuit $C$, $n$, fault list $F$
**Outputs**: $n$-detect test set $T$
01: $TF$=generate_$n$_test_functions($C, n, F$)
02: undirected graph $G(V, E) = \emptyset$
03: **for each** fault $f_i$ in $F$
04:     **for** $j = i$ **to** $n$
05:         insert vertex corresponding to $tf_{ij}$ in $V$
06:         insert barrier between $tf_{ij}$ and all $tf_{ik}$ for $k \neq j$ in $E$
07: **do**
08:     **for each** vertex $u_i \in V$
09:         **for each** vertex $u_j \in V,\ u_j \neq u_i$
10:             $tf_i$ = test function corresponding to $u_i$
11:             $tf_j$ = test function corresponding to $u_j$
12:             **if** ($tf_i \bullet tf_j \neq 0$) AND ($\nexists$ barrier between $u_i$ and $u_j$)
13:                 insert edge between $u_i$ and $u_j$ in $E$
14:                 weight($u_i, u_j$)= size of maximum cube in $tf_i \bullet tf_j$
15:             **end if**
16:     **for each** vertex $u_i \in V$
17:         **while** adj($u_i$) $\neq 0$
18:             $u_{min}$ = vertex of minimum weight in adj($u_i$)
19:             $u_i$ = merge $u_i$ and $u_{min}$
20:             adj($u_i$) = adj($u_i$) $\cap$ adj($u_{min}$)
21:             barr($u_i$) = barr($u_i$) $\cup$ barr($u_{min}$)
22:             $tf_i$ = $tf_i \bullet tf_{min}$
23:         **end while**
24: **until** $E$ contains only barriers
25: **for each** vertex $u_i \in V$
26:     $tf_i$ = test function corresponding to $u_i$
27:     $T$ = $T$ + cube from $tf_i$
28: fix unspecified bits in $T$
29: **return** $T$
```

Figure 7.3: Proposed ATPG method

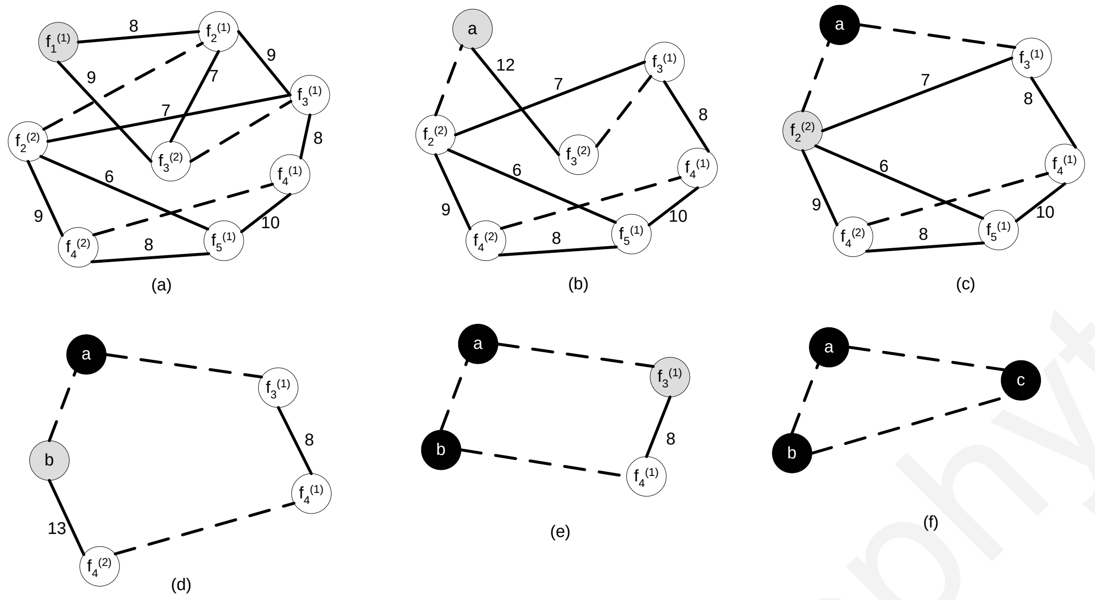


Figure 7.4: Example of Barrier Test Compatibility Reduction Algorithm

of detections for some fault (line 06), and, thus, disturb the n-detect fault coverage. The algorithm then focuses on a single vertex (line 16) and identifies the minimum weighted edge among the list of its adjacent edges (line 17-18). This edge is then absorbed by u_i (lines 19-22). Only the common edges of u_i and u_{min} remain in the new adjacent list of u_i (line 20). The barriers of u_{min} are included in the list of barriers of u_i (line 21) to avoid the combination of u_i with a vertex that u_{min} can not be combined. Of course, the new test function of u_i is the result of the logic AND operation between the corresponding functions (line 22). The same process is repeated until only barrier edges are left in graph G. The desired n-detect test set is generated by obtaining one test cube from every test function corresponding to the remaining vertices. The don't care bits in each test cube can be fixed either in a random manner, or as in [108, 96] to increase the possibility of different fault excitation conditions for even further diverse tests.

We demonstrate the execution of the algorithm of Figure 7.3 with an illustrative example for $n = 2$, shown in Figure 7.4. Let the number of faults be 5, $f_1, ...f_5$. Initially graph G contains 8 vertices. For each one of the faults f_2, f_3 and f_4 there are two vertices in the graph indicating the two different detections and differentiated by the superscript index. For faults f_1 and f_5, only one test exists and so a single vertex is present in G, for each one of these faults. A solid line between two vertices represents a weighted edge in G, while a dashed line represents a barrier (Figure 7.4(a)).

First, the algorithm examines vertex $f_1^{(1)}$ (shown in light grey) and selects the minimum edge adjacent to it (Figure 7.4(a)). The vertex at the other endpoint of the edge (i.e., $f_2^{(1)}$) in absorbed by $f_1^{(1)}$ and the new node a (Figure 7.4(b)) contains all tests that detect both $f_1^{(1)}$ and $f_2^{(1)}$. Only one common vertex exists in the lists of adjacent vertices of $f_1^{(1)}$ and $f_2^{(1)}$, i.e., the vertex corresponding to fault $f_3^{(2)}$. This edge is included in the list of adjacent vertices of a as it can be observed from Figure 7.4(b). The barrier of $f_2^{(1)}$ is included in the barrier list of $f_1^{(1)}$. Next, the only weighted edge of vertex a is selected and vertex $f_3^{(2)}$ is absorbed by a adding a new barrier between a and $f_3^{(1)}$ (Figure 7.4(c)). Since, vertex a has no more weighted edges the algorithm selects another vertex to examine, i.e., $f_2^{(2)}$ and vertex a is retired (shown in black). In the same rationale, the edge with fault $f_5^{(1)}$ is selected and $f_5^{(1)}$ is absorbed (Figure 7.4(d)). The algorithm terminates when only barriers are left in G (Figure 7.4(f)). Each vertex left corresponds to a test function that contains valid tests for detecting all absorbed faults. Specifically, each test in the test function corresponding to vertex a detects faults $f_1^{(1)}, f_2^{(1)}$ and $f_3^{(2)}$ which means that faults f_1 and f_2 are propagated via some paths in their subcircuit named 1 and fault f_3 via some path in its subcircuit named 2. Vertex b covers faults $f_2^{(2)}, f_4^{(2)}$ and $f_5^{(1)}$ and vertex c $f_3^{(1)}$ and $f_4^{(1)}$. Thus, the n-detect coverage required in the initial G is maintained, and each of the 2 tests for faults f_2, f_3 and f_4 propagate their effect via different paths.

The vertices left correspond to a test function that contains valid tests for the same number of detections of the faults initially present in G. Specifically, a contains all valid tests for all faults f_1, f_2 and f_3, vertex b for faults f_2, f_4 and f_5 and vertex c for f_3 and f_4. Obtaining a cube from each one of the test functions corresponding to these vertices, gives a valid 2-detect test set for the circuit under examination.

7.4 Experimental Results

The proposed methodology was implemented using ANSI C++, in a UNIX environment. All experiments were run on a 1GHz SunBlade 1500 with 4GB of RAM, using the ISCAS'85 and the full-scan versions of the ISCAS'89 benchmarks. The function-based ATPG tool was implemented using BDDs (on top of CUDD) (see Section 4.5).

Table 7.1: Average Number of Propagation Paths Per Fault

			1-detect test set	10-detect test sets					
	avg.	avg. non-empty	avg.	method of Chapter 4		[97]		Proposed	
Circuit	paths segments	prop. subcircuits	prop. paths	avg. prop. paths	% increase	avg. prop. paths	% increase	avg. prop. paths	% increase
c880	6.634	6.234	3.691	4.121	1.000	n/a	n/a	6.317	6.107
c1355	10.585	9.874	10.122	10.213	1.000	n/a	n/a	10.527	4.451
c1908	9.666	8.123	8.198	8.213	1.000	n/a	n/a	8.431	15.533
c2670	9.901	9.123	8.771	8.980	1.000	n/a	n/a	9.686	4.378
c3540	12.857	9.345	8.931	9.045	1.000	n/a	n/a	9.362	3.781
c5315	8.681	7.213	6.853	6.987	1.000	n/a	n/a	7.583	5.448
c7552	10.401	9.111	5.139	6.436	1.000	n/a	n/a	7.951	2.168
s208	2.877	2.789	2.473	2.769	1.000	2.789	1.068	2.794	1.084
s298	2.626	2.267	2.264	2.265	1.000	2.266	2.000	2.271	7.000
s344	3.863	3.685	3.449	3.663	107.000	3.451	1.000	3.686	118.500
s382	3.469	3.300	3.259	3.269	1.000	3.275	1.600	3.310	5.100
s420	3.683	3.415	2.825	2.843	1.000	2.881	3.111	3.467	35.667
s510	2.569	2.562	2.473	2.479	1.000	2.503	5.000	2.562	14.833
s526	2.578	2.471	2.213	2.391	1.113	2.373	1.000	2.474	1.631
s641	5.425	5.203	4.421	5.174	9.779	4.498	1.000	5.209	10.234
s820	1.571	1.570	1.570	1.570	1.000	1.570	1.000	1.570	1.000
s953	3.993	3.992	3.538	3.698	1.000	3.719	1.131	3.996	2.863
s1196	5.849	4.562	3.714	3.923	1.000	4.152	2.096	4.660	4.526
s1423	6.376	6.000	4.451	5.664	1.626	5.197	1.000	6.006	2.084
s9234	5.762	5.463	4.648	4.912	7.765	4.682	1.000	5.487	24.676
s13207	6.142	5.931	5.634	5.705	1.000	5.801	2.352	5.976	4.817
s15850	6.081	5.897	5.315	5.452	1.305	5.420	1.000	5.996	6.486
s38417	6.744	6.441	5.983	6.063	1.000	6.121	1.725	6.487	6.300

We consider the stuck-at fault model and obtain the fault lists using the checkpoint theorem. We compare the generated test sets with two traditional n-detect test sets available to us, in order to demonstrate the increase in propagation paths obtained when using our method for n-detect test generation. The first one was derived using the 1-detect test generation method of Chapter 4, run 10 different times with special consideration in maintaining the n-detect property. The second is the compact 10-detect test sets of [97]. A comparison with the test sets of [96, 108] is not possible at this point, since we do not have them by the time of this thesis completion. Nevertheless, as explained in the introduction of this chapter, these two methods do not target explicitly the problem of guaranteeing different propagation paths. More importantly, they are complementary to the proposed method (our technique can apply on top of these methods). Finally, a 1-detect test set obtained by the academic tool ATALANTA was used in order to show that the increase in the number of propagation paths in the tests sets of the proposed method is not an inherent property of traditional n-detect test sets.

Table 7.1 shows comparisons on the average number of propagation paths per fault.After the circuit name we show the average number of path segments per fault, obtained by a forward traversal from the fault site to all primary outputs. This is essentially a static information which gives an upper bound on how many different propagation paths exist per fault. Of course this number is not expected to be reached for two reasons: (i) some of these paths cannot propagate the fault effect to a primary output and, (ii) the algorithm does not generate more than n tests, per fault, even if they exist. In Column 3 we report the average number of possible non-empty propagation subcircuits (in each subcircuit, there exists at least one path segment that can propagate the fault effect) as obtained by the algorithms of Figure 7.1 and Figure 7.2. Observe that this gives the exact number of possible propagation paths per fault, on the average, for n=10. For example, for c880 there exists only 6.234 propagation paths per fault, on the average, for n=10. Hence, the proposed method should generate test sets that detect each fault at least as many times. This is demonstrated in Column 9 (the slightly higher numbers are attributed to additional coincidental fault propagations in a subcircuit). We have used an in-house tool based on Zero-suppressed BDDs in order to count propagation paths. This tool allows counting a propagation path only once (no double counting), without an explicit enumeration of the paths. Moreover, it does not consider the subpaths of a path as different propagation paths. Thus, in cases where propagation is achieved through two paths segments one of which is a subpath of the other, the tool counts only one path, specifically the larger one. Columns 4, 5, and 7 list the average number of fault propagations in the 1-detect, the 10-detect using the method of Chapter 4, and the 10-detect of [97] test sets, respectively. A **n/a** here indicates that no test set is available for the corresponding method. In all cases the average number of propagation paths per fault is higher in the proposed method. The small difference in the averages implies that the extra effort by the proposed method is necessary when we need to target additional propagation paths. As an example consider circuit s382. The average number of propagation paths per fault for the 1-detect test set is 3.259 and for the 10-detect test set of the proposed method is 3.31, give an 0.051 improvement. However, the traditional 10-detect test sets give an even smaller improvement with average numbers of 3.269 and 3.275, i.e., 0.01 and 0.016 improvement respectively. Hence, the proposed method achieved 5 times more improvement from the first test set and 3.2 improvement from the second one (the 1-detect test set is used as a reference point here, in order to show the real improvement on the number of propagation paths in the various n-detect test sets). This analysis explains the difficulty in obtaining additional propagation paths, from those already obtained in a 1-detect test set, and, thus, the importance of the proposed method in increasing the quality of the n-detect test sets. In Columns 6, 8 and 10 we show the % increase (improvement) of the average number of propagation paths for the three different 10-detect test sets, similar to the example given before. This percentage is computed as the difference in propagation paths from the 1-detect test set, normalized to the smaller such difference among the 10-detect test sets presented here. For most of the cases this increase is considerable and in some cases extremely high.

Table 7.2: Total Number of Propagation Paths

		10-detect test sets		
Circuit	**1-detect**	**method of Chapter 4**	**[97]**	**Proposed**
c880	1422	1459	n/a	6121
c1355	702908	712138	n/a	748038
c1908	8342	8621	n/a	9863
c2670	20959	21293	n/a	331307
c3540	10103212	10121231	n/a	12912812
c5315	589591	592103	n/a	642440
c7552	38350	40415	n/a	41878
s1196	2157	1989	1962	2157
s1423	3939	8647	6574	13940
s9234	20064	25727	20143	47692
s13207	582132	592346	598211	612391
s15850	7010166	7017295	7012133	7285178
s38417	424693	429089	432093	465903

Table 7.2 gives the total number of propagation paths. We show numbers only for the larger circuits since the number of propagation paths for the small circuits are close to the total number of physical paths in each circuit, even for the 1-detect test sets. For each test set we count the total number of propagation paths among all fault avoiding double counting of propagation paths as well as counting propagation paths that are fully contained in larger propagation paths. Hence, the reported paths are all distinct, and not properly contained in other, path segments that propagate at least one fault in each circuit. The number of these paths is increased considerably by the proposed method, allowing us to conclude that in traditional 10-detect test sets there is a lot of propagation path overlap between tests for different faults.

Table 7.3 gives test set sizes (Columns 2, 4, and 6) and an estimation on non-modeled fault coverages, using the Bridging Fault (BF) model as a surrogate for defects (Columns 3, 5, 7). The BF coverages are estimated using the formula proposed in [105], indicated by BCE+ in Table 7.3. Observe that the test set size may be larger in the case of the proposed algorithm, yet this is necessary in order to guarantee propagation through different paths. For all the examined circuits the BF coverage is increased, indicating that the proposed n-detect test

Table 7.3: Test Set Sizes and Bridging Fault Coverage

	10-detect method of Chapter 4		10-detect of [97]		Proposed	
Circuit	**tests**	**BCE+**	**tests**	**BCE+**	**tests**	**BCE+**
c880	200	0.97044	n/a	n/a	260	0.97493
c1355	840	0.92412	n/a	n/a	891	0.92712
c1908	1070	0.95872	n/a	n/a	1193	0.95934
c2670	550	0.95863	n/a	n/a	689	0.96687
c3540	1000	0.96100	n/a	n/a	1545	0.96809
c5315	510	0.97986	n/a	n/a	872	0.99382
c7552	780	0.95213	n/a	n/a	1238	0.96871
s208	340	0.92833	271	0.92688	350	0.95650
s298	290	0.99780	234	0.99595	281	0.99619
s344	200	0.98362	138	0.98643	168	0.98989
s382	290	0.99830	253	0.99722	283	0.99764
s420	690	0.89006	433	0.85641	705	0.91460
s510	560	0.97828	543	0.98383	586	0.98541
s526	580	0.99204	492	0.99382	568	0.99427
s641	330	0.98896	227	0.99125	307	0.99405
s820	1050	0.99899	949	0.99959	1064	0.99969
s953	780	0.88213	766	0.90938	935	0.91464
s1196	1150	0.95145	1233	0.96564	1450	0.96804
s1423	240	0.95621	269	0.97374	371	0.97114
s9234	1450	0.88093	1132	0.90058	1471	0.88620
s13207	2670	0.91213	2341	0.90784	2614	0.99989
s15850	1600	0.94524	983	0.92798	1824	0.96242
s38417	880	0.98127	784	0.98789	981	0.99012

sets are of higher quality.

7.5 Conclusions

In this chapter we presented a novel method for generating high quality n-detetct test sets. The methodology guarantees to increase the number of paths in the circuit that propagate a fault to a primary output of the circuit. A new method for obtaining subcircuits per fault cone and enforcing propagation through each subcircuit is presented, without explicit path or path segment enumeration. A graph-based algorithm controls the size of the test set without removing the n-detect property of the test set. The obtained results show considerable improvement for the n-detect test sets generated using the proposed method, in terms of propagating the fault effect via additional propagation paths as well as increased non-modeled fault coverages.

CHAPTER 8

CONCLUDING REMARKS

Manufacturing testing of digital VLSI circuits is becoming an increasingly challenging problem due to the enormous circuit complexity, as well as, the nanometer technology scaling. Very complex large designs with tight timing constraints give rise to a number of diverse types of defects. Thus, traditional fault models and/or test generation techniques are no longer adequate for guaranteeing that, within reasonable time and effort, delivered devices will not exhibit malfunction.

This dissertation presents a number of techniques that provide high quality test sets, for modern digital integrated circuits. Quality is defined in a number of different ways, depending on the application under examination, defining a number of different problems.

The first part of the dissertation examines the problem of high quality test generation for delay faults. The two traditional fault models for delay are either not efficient or not effective enough. The Transition Fault model, while of linear complexity, can only detect delays lumped at a specific circuit site. On the other hand, the Path Delay Fault model can detect both lumped and distributed delays, yet its complexity is exponential to the circuit size, limiting its applicability. Thus, we considered a hybrid model that is of increased quality, yet its complexity remains linear to the size of the circuit. A function-based test generation framework was proposed following established path sensitization criteria that provide different quality levels. Moreover, we have proposed a number of enhancements on top of the generated test sets in order to provide more quality characteristics. The test set compaction method proposed, reduces the test set size with small decrease on the number of sensitized paths, and allows for enriching the test set with tests that excites the circuit's potential fault sites through different paths. Additional quality attributes can be assigned to the test sets by restricting fault propagation only through critical parts of the circuit. The enrichment techniques proposed here are some of many different enhancement methods that can be applied on top of the test sets obtained for delay faults using the proposed function-based framework. Future work could examine more such enhancement methods that take advantage of the size reduction obtained by the proposed test set compaction technique. In addition, appropriate combination of the test generation framework and compaction techniques could be used for a number of different test related problems. An interesting direction would be to take advantage of the test generation framework in order to generate a large number of tests, with different path sensitization criteria and for different critical parts of the circuit. On top of that the proposed test compaction technique can be used, tailored for low power testing, i.e., a technique which selects the tests patterns that will give lower power density during test application. The latter can be done effectively since, the compaction technique proposed can take full advantage of the information on the tests' quality used in the test generation process of the function-based framework.

Making the test generation process flexible can benefit a number of test related applications, yet for each different application the generation process has to be appropriately tuned in order to give the desired results. A similar, yet more general approach is to have flexible test sets, i.e. develop techniques that give test sets that can be appropriately utilized and give benefit to different applications. In order for the test sets to be flexible, they should have a large number of test bits that do not have a specified logic value, i.e., can either get the logic one or the logic zero value. We proposed a number of different such techniques, with this dissertation, that follow two different rationales for the stuck-at fault model, that can also be used with any linear fault model. A static one, that concentrates on relaxing a given test set, which may be fully or partially specified and a dynamic one, which essentially performs test generation, keeping the number of specified bits in the test set, small. The static methods give high unspecified to total test bits ratio and, at the same time, keep all the given test set attributes. On the other hand, the dynamic methods give slightly larger test sets, yet the unspecified to total test bits ratios are significantly larger. Moreover, they give more balanced distribution of specified bits among the test patterns.

The flexible test sets can be used in a number of test related applications. Some of them, like on-chip test set embedding and test set compression, use the unspecified bits as degrees of freedom in order to increase the encoding efficiency and keep the required overhead small. A number of other applications, like low power testing and test set enrichment, fix the values of the unspecified bits appropriately so as to reach the desired result. We have experimented with two, fairly simple, test set embedding schemes that are based on the reseeding of Linear Feedback Shift Registers (LFSRs) using the flexible test sets obtained by the proposed methods. The results demonstrate the benefit that these applications have when using the flexible test sets, instead of using general purpose test sets. While using flexible test sets in different application has an interesting perspective, another promising direction is generating flexible tests sets under non-traditional fault models in order to be used in a more demanding, in terms of quality, scheme. Future research could concentrate on the generation of flexible test sets for delay fault models like the recently proposed models for small delay faults or even for modeled defects like bridging faults. Developing test generation of test sets with a large number of unspecified bits, under the hybrid delay model proposed in this dissertation, is expected to provide more room for quality enhancements of the obtained test sets. Even examining static relaxation for the obtained test sets under the function-based framework for this hybrid model, could be considered as a continuation of this work.

Finally, a different approach is presented in order to produce test sets of increased quality by generating test sets that target each modeled fault more than one times. This has been previously shown to give test sets with high defect detection capabilities, beyond stuck-at fault coverage. These increased quality tests sets are commonly known as n-detect test sets, as they enforce detection of each fault by n different test patterns. Our experimentation on the relaxation of n-detect test sets has shown high relaxation rates, similar to those of 1-detect test sets. At the same time, they maintain their high quality after the relaxation process and the unspecified bit value fixing, even in a random fashion. Relaxation is of increased importance in n-detect test sets, since their large size can limit their usage in a large number of applications. Future work should try to evaluate the relaxation effect on the applicability of n-detect test sets. Preliminary work on higher quality n-detects test set generation was presented here. This technique imposes a diversity constraint on the n different tests that target the same fault. Diversity is ensured by systematic fault cone partitioning of the circuit-under-test, which is guaranteed to be as balanced as possible. Experimental results show increased defect coverage to that of traditional n-detect test sets, denoting that there is great potential. We intend to investigate more sophisticated methods to improve test set size, while guaranteeing the maximum possible diversity. Promising ideas on developing a complete function-based framework that controls the diversity of the tests via different propagation paths, without explicit path enumeration are going to be explored in the future.

BIBLIOGRAPHY

[1] “International Technology Roadmap for Semiconductors,” tech. rep., ITRS, 2007.

[2] G. Gielen and R. Rutenbar, “Computer-aided Design of Analog and Mixed-Signal Integrated Circuits,” *Proceedings of the IEEE*, vol. 88, no. 12, pp. 1825–1854, Dec. 2000.

[3] S. Hamilton, “Taking Moore’s law into the next century,” *IEEE Computer*, vol. 32, no. 1, pp. 43–48, 1999.

[4] N. Verghese and D. Allstot, “Computer-aided Design Considerations for Mixed-Signal Coupling in RF Integrated Circuits,” *IEEE Journal of Solid-State Circuits*, vol. 33, no. 3, pp. 314–323, 1998.

[5] Y. Joannon, V. Beroulle, C. Robach, S. Tedjini, and J.-L. Carbonero, “Decreasing Test Qualification Time in AMS and RF Systems,” *IEEE Design & Test of Computers*, vol. 25, no. 1, pp. 29–37, Jan.-Feb. 2008.

[6] S. Krishnaswamy, I. Markov, and J. Hayes, “Tracking Uncertainty with Probabilistic Logic Circuit Testing,” *IEEE Design & Test of Computers*, vol. 24, no. 4, pp. 312–321, July-Aug. 2007.

[7] S. Akbay, A. Halder, A. Chatterjee, and D. Keezer, “Low-cost test of embedded RF/analog/mixed-signal circuits in SOPs,” *IEEE Transactions on Advanced Packaging*, vol. 27, no. 2, pp. 352–363, 2004.

[8] N. Kranitis, A. Merentitis, G. Theodorou, A. Paschalis, and D. Gizopoulos, “Hybrid-SBST Methodology for Efficient Testing of Processor Cores,” *IEEE Design & Test of Computers*, vol. 25, no. 1, pp. 64–75, Jan.-Feb. 2008.

[9] Y. Zorian, S. Dey, and M. Rodgers, “Test of future system-on-chips,” *IEEE/ACM International Conference on Computer Aided Design*, pp. 392–398, 2000.

[10] Y. Zorian, E. Marinissen, and S. Dey, “Testing Embedded-Core-Based System Chips,” *IEEE Computer*, vol. 32, no. 6, pp. 52–60, 1999.

[11] P. Girard, “Survey of Low-Power Testing of VLSI Circuits,” *IEEE Design & Test*, vol. 19, no. 3, pp. 82–92, 2002.

[12] J. Monteiro and S. Devadas, *Computer-Aided Design Techniques for Low Power Sequential Logic Circuits*. Kluwer Academic Publishers Norwell, MA, USA, 1997.

[13] R. Aitken, “Nanometer Technology Effects on Fault Models for IC Testing,” *IEEE Computer*, vol. 32, no. 11, pp. 46–51, Nov 1999.

[14] M. K. Michael, S. Neophytou, and S. Tragoudas, “Functions for Quality Transition Fault Tests,” in *Proc. of IEEE International Symposium on Quality of Electronic Design*, pp. 327–332, 2005.

[15] S. Neophytou and M. K. Michael and S. Tragoudas, “Test Set Enhancement for Quality Transition Faults using Function-based Methods,” in *Proc. of the 15th IEEE/ACM Great Lakes Symposium on VSLI*, pp. 182–187, 2005.

[16] S. N. Neophytou, M. K. Michael, and S. Tragoudas, “Functions for Quality Transition-Fault Tests and Their Applications in Test-Set Enhancement,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 12, pp. 3026–3035, Dec. 2006.

[17] S. Neophytou and M. K. Michael, “Two New Methods for Accurate Test Set Relaxation via Test Set Replacement,” in *Proc. of IEEE International Symposium on Quality of Electronic Design*, 2008.

[18] S. Neophytou, M. K. Michael, and S. Tragoudas, “Efficient Deterministic Test Generation for BIST Schemes with LFSR Reseeding,” *Proc. of IEEE International On-Line Testing Symposium*, pp. 43–50, 2006.

[19] S. Neophytou and M. K. Michael, “Hierarchical Fault Compatibility Identification for Test Generation with a Small Number of Specified Bits,” in *Proc. of IEEE Defect and Fault Tolerance Symposium*, pp. 439–447, 2007.

[20] S. Hellebrand, S. Tarnick, B. Courtois, and J. Rajski, “Generation of Vector Patterns Through Reseeding of Multipe-Polynominal Linear Feedback Shift Registers.,” in *Proc. of International Test Conference*, pp. 120–129, 1992.

[21] C. V. Krishna, A. Jas, and N. A. Touba, “Achieving high Encoding Efficiency with Partial Dynamic LFSR Reseeding,” *ACM Transactions on Design Automation of Electronic Systems*, vol. 9, no. 4, pp. 500–516, 2004.

[22] S. Neophytou and M. K. Michael, “On the Relaxation of n-detect Test Sets,” in *Proc. of IEEE International VLSI Test Symposium*, pp. 187–192, 2008.

[23] A. Krstic and K. Cheng, *Delay Fault Testing for VLSI Circuits*. Kluwer Academic Publishers, 1998.

[24] A. Majhi and V. Agrawal, “Delay Fault Models and Coverage,” in *Proc. of International Conference on VLSI Design*, pp. 364–369, 1998.

[25] K. Cheng and H. Chen, “Classification and Identification of Nonrobust Untestable Path-Delay Faults,” *IEEE Transactions on CAD*, vol. 15, no. 8, pp. 845–853, 1996.

[26] C. Lin and S. Reddy, “On Delay Fault Testing in Logic Circuits,” *IEEE Transactions on CAD*, vol. 6, no. 5, pp. 694–703, 1987.

[27] G. Smith, “Model for delay faults based upon paths,” in *Proc. of International Test Conference*, pp. 342–349, 1985.

[28] Y. Levendel and P. Menon, “Transition Faults in Combinational Circuits: Input Transition Test Generation and Fault Simulation,” in *Proc. of International Fault Tolerant Computing Symposium*, pp. 278–283, 1986.

[29] A. Pramanick and S. Reddy, “On the Detection of Delay Faults,” in *Proc. of International Test Conference*, pp. 845–856, 1988.

[30] M. Schulz and F. Brglez, “Accelerated Transition Fault Simulation,” in *Proc. of Conference on Design Automation*, pp. 237–243, 1987.

[31] J. Waicukauski, E. Lindbloom, B. Rosen, and V. Iyengar, “Transition Fault Simulation,” *IEEE Design & Test of Computers*, vol. 4, no. 2, pp. 32–38, 1987.

[32] Y. Shao, I. Pomeranz, and S. Reddy, “On Generating High Quality Tests for Transition Faults,” in *Proc. of Asian Test Symposium*, pp. 1–8, 2002.

[33] H. Konuk, “On Invalidation Mechanisms for Non-Robust Delay Tests,” in *Proc. of International Test Conference*, pp. 393–399, 2000.

[34] R. Bryant, “Graph-Based Algorithms for Boolean Function Manipulation.,” *IEEE Transactions on Computers*, vol. C-35, no. 8, pp. 677–691, 1986.

[35] M. Michael, T. Haniotakis, and S. Tragoudas, “A Unified Framework for Generating all Propagation Functions for Logic Errors and Events,” *IEEE Transactions on CAD*, vol. 23, no. 6, pp. 980–986, 2004.

[36] K. Yang, K. Cheng, and L. Wang, “TranGen: A SAT-Based ATPG for Path-Oriented Transition Faults,” in *Proc. of Asia and South Pacific Design Automation Conference*, pp. 92–97, 2004.

[37] I. Hamzaoglu and J. Patel, “Compact two-pattern Test Set Generation for Combinational and Full Scan Circuits,” in *Proc. of International Test Conference*, pp. 944–953, 1998.

[38] P. Agrawal, D. Bhattacharya, and V. D. Agrawal, “Test Generation for Path Delay Faults Using Binary Decision Diagrams,” *IEEE Transactions on Computer*, vol. 44, no. 3, pp. 434–447, 1995.

[39] D. Kirkpatrick and A. Sangiovanni-Vincentelli, “Digital Sensitivity: Predicting Signal Interaction Using Functional Analysis,” in *Proc. of International Conference on Computer-Aided Design*, pp. 536–541, 1996.

[40] M. Michael and S. Tragoudas, “Function-based Compact Test Pattern Generation for Path Delay Faults,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 13, no. 8, pp. 996–1001, Aug. 2005.

[41] F. Somenzi, “CUDD: CU Decision Diagram Package.” Dept. of ECE, The University of Colorando., release 2.3.0 1999.

[42] S. Wang and S. Gupta, “ATPG for Heat Dissipation Minimization During Test Application,” *IEEE Transactions on Computers*, vol. 47, no. 2, pp. 256–262, 1998.

[43] I. Pomeranz and S. M. Reddy, "Test Enrichment for Path Delay Faults Using Multiple Sets of Target Faults," *IEEE Transactions on CAD*, vol. 22, no. 1, pp. 82–90, 2003.

[44] S. Hellebrand, J. Rajski, S. Tarnick, S. Venkataraman, and B. Courtois, "Built-In Test for Circuits with Scan Based on Reseeding of Multiple-Polynomial Linear Feedback Shift Registers," *IEEE Transactions on Computers*, vol. 44, pp. 223–233, 1995.

[45] C. Krishna, A. Jas, and N. A. Touba, "Test Vector Encoding Using Partial LFSR Reseeding," in *Proc. of International Test Conference*, pp. 885–893, 2001.

[46] K. Chakrabarty, B. T. Murray, and V. Iyengar, "Deterministic Built-In Pattern Generation for High-Performance Circuits Using Twisted Ring Counters," *IEEE Transactions on VLSI Systems*, vol. 8, pp. 633 – 636, 2000.

[47] I. Hamzaoglu and J. H. Patel, "Test Set Compaction Algorithms for Combinational Circuits," *IEEE Transactions on CAD*, vol. 19, no. 8, pp. 957–963, 2000.

[48] K. Miyase and S. Kajihara, "XID: Don't care Identification of Test Patterns for Combinational Circuits," *IEEE Transactions on CAD*, vol. 23, no. 2, pp. 321–326, 2004.

[49] A. El-Maleh and A. Al-Suwaiyan, "An Efficient Test Relaxation Technique for Combinational & Full-Scan Sequential Circuits," in *Proc. of VLSI Test Symposium*, pp. 53–59, 2002.

[50] A. Narayan, J. Jain, M. Fujita, and A. Sangiovanni-Vincentelli, "Partitioned ROBDDs – a compact, canonical and efficiently Manipulable Representation for Boolean Functions," in *Proc. of International Conference on Computer-Aided Design*, pp. 547–554, 1996.

[51] J. Chang and C. Lin, "Test Set Compaction for Combinational Circuits," *IEEE Transactions on CAD*, vol. 14, no. 11, pp. 1370–1378, 1995.

[52] T. Niermann, R. Roy, J. Patel, and J. Abraham, "Test Compaction for Sequential Circuits," *IEEE Transactions on CAD*, vol. 11, no. 2, pp. 260–267, 1992.

[53] S. Kajihara, K. Ishida, and K. Miyase, "Test Vector Modification for Power Reduction during Scan Testing," in *Proc. of VLSI Test Symposium*, pp. 160–165, 2002.

[54] S. Chakradhar and A. Raghunathan, “Bottleneck Removal Algorithm for Dynamic Compaction in Sequential Circuits,” *IEEE Transactions on CAD*, vol. 16, no. 10, pp. 1157–1172, 1997.

[55] I. Pomeranz, L. Reddy, and S. Reddy, “COMPACTEST: a Method to Generate Compact Test Sets for Combinational Circuits,” *IEEE Transactions on CAD*, vol. 12, no. 7, pp. 1040–1049, 1993.

[56] M. Schulz, E. Trischler, and T. Sarfert, “SOCRATES: a Highly Efficient Automatic Test Pattern Generation System,” *IEEE Transactions on CAD*, vol. 7, no. 1, pp. 126–137, 1988.

[57] S. Kajihara, I. Pomeranz, K. Kinoshita, and S. Reddy, “Cost-Effective Generation of Minimal Test Sets for Stuck-at Faults in Combinational Logic Circuits,” *IEEE Transactions on CAD*, vol. 14, no. 12, pp. 1496–1504, 1995.

[58] R. Karp, “Reducibility Among Combination Problems,” *Complexity of Computer Computations*, pp. 85–103, 1972.

[59] M. Bushnell and V. Agrawal, *Essentials of Electronic Testing*. Kluwer Academic Publishers, 2000.

[60] D. Drake and S. Hougardy, “A simple approximation algorithm for the weighted matching problem,” *Information Processing Letters*, pp. 211–213, 2003.

[61] M. Michael and S. Tagoudas, “ATPG Tools for Delay Faults at the Functional Level,” *ACM Transactions on Design Automation of Electronic Systems*, vol. 7, no. 1, pp. 33–57, 2002.

[62] D. Sahoo, S. Iyer, J. Jain, C. Stangier, A. Narayan, D. Dill, and E. Emerson, “A Partitioning Methodology for BDD-Based Verification,” in *Proc. of International Conference on Formal Methods in Computer-Aided Design*, pp. 399–413, 2004.

[63] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, “Chaff: Engineering an Efficient SAT Solver,” in *Proc. of Design Automation Conference*, pp. 530–535, 2001.

[64] E. Goldberg and Y. Novikov, “BerkMin: A fast and robust Sat-solver,” *Discrete Applied Mathematics*, vol. 155, no. 12, pp. 1549–1561, 2007.

[65] M. Davis, G. Logemann, and D. Loveland, “A Machine Program for Theorem-Proving,” *Communications of the ACM*, vol. 5, no. 7, pp. 394–397, 1962.

[66] T. Larabee, “Test Pattern Generation using Boolean Satisfiability,” *IEEE Transactions on CAD*, vol. 1, no. 1, pp. 4 – 15, 1992.

[67] P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli, “Combinational Test Generation Using Satisfiability,” *IEEE Transactions on CAD*, vol. 15, no. 9, pp. 1167–1176, 1996.

[68] J. Marques-Silva and K. Sakallah, “Boolean satisfiability in electronic design automation,” in *Proc. of ACM/IEEE Design Automation Conference*, vol. 5, pp. 675–680, 2000.

[69] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, “Symbolic Model Checking without BDDs,” *Lecture Notes in Computer Science*, vol. 1579, pp. 193–207, 1999.

[70] A. Biere, E. Clarke, R. Raimi, and Y. Zhu, “Verifying Safety Properties of a PowerPC Microprocessor Using Symbolic Model Checking without BDDs,” *Lecture Notes in Computer Science*, vol. 1633, p. 60, 1999.

[71] G. Parthasarathy, M. Iyer, and K. Cheng, “A comparison of BDDs, BMC, and sequential SAT for model checking,” in *Proc. of the IEEE International Workshop on High-Level Design Validation and Test*, 2003.

[72] A. Chandra and K. Chakrabarty, “System-on-a-chip test-data compression and decompressionarchitectures based on Golomb codes,” *IEEE Transactions on CAD*, vol. 20, no. 3, pp. 355–368, 2001.

[73] H. Liang, S. Hellebrand, and H.-J. Wunderlich, “Two-dimensional test data compression for scan-based deterministic BIST,” in *Proc. of International Test Conference*, pp. 894–902, 2001.

[74] J. Rajski, J. Tyszer, and N. Zacharia, “Test Data Decompression for Multiple Scan Designs with Boundary Scan.,” *IEEE Transactions on Computers*, vol. 47, no. 11, pp. 1188–1200, 1998.

[75] K. D. Wagner, C. K. Chin, and E. J. McCluskey, “Pseudorandom Testing,” *IEEE Transactions on Computers*, vol. 36, no. 3, pp. 332–343, 1987.

[76] M. Lempel, S. Gupta, and M. Breuer, "Test Embedding with Discrete Logarithms," *IEEE Transactions on CAD*, vol. 15, no. 5, pp. 554–566, 1995.

[77] B. Koenemann, "LFSR-Coded Test Patterns for Scan Designs," in *Proc. of European Test Conference*, pp. 237–242, 1991.

[78] A. A. Al-Yamani and E. J. McCluskey, "BIST-guided ATPG," in *Proc. of International Symposium on Quality of Electronic Design*, (USA), pp. 244–249, IEEE Computer Society, 2005.

[79] S. Hellebrand, B. Reeb, H.-J. Wunderlich, and S. Tarnick, "Pattern Generation for a Deterministic BIST Scheme," in *Proc. of International Conference on Computer-Aided Design*, pp. 88–94, 1995.

[80] S. Hellebrand, H.-J. Wunderlich, and H. Liang, "A Mixed Mode BIST Scheme Based on Reseeding of Folding Counters," in *Proc. of International Test Conference*, pp. 778–784, 2000.

[81] E. Kalligeros, D. Kaseridis, X. Kavousianos, and D. Nikolos, "Reseeding-Based Test Set Embedding with Reduced Test Sequences.," in *Proc. of International Symposium on Quality of Electronic Design*, pp. 226–231, 2005.

[82] E. Kalligeros, D. Kaseridis, X. Kavousianos, and D. Nikolos, "Efficient Multiphase Test Set Embedding for Scan-based Testing," in *Proc. of International Symposium on Quality of Electronic Design*, pp. 433 – 438, 2006.

[83] E. Kalligeros, X. Kavousianos, and D. Nikolos, "Multiphase BIST: a new reseeding technique for high test-data compression.," *IEEE Transactions on CAD*, vol. 23, no. 10, pp. 1429– 1446, 2004.

[84] L. Li and K. Chakrabarty, "Hybrid BIST Based on Repeating Sequences and Cluster Analysis," in *Proc. of Design Automation and Test in Europe*, vol. 2, pp. 1142–1147, 2005.

[85] S. Swaminathan and K. Chakrabarty, "On Using Twisted-Ring Counters for Test Set Embedding in BIST," *Journal of Electronic Testing, Theory and Applications*, vol. 17, pp. 529 – 542, Dec 2001.

[86] P. Flores, H. Neto, and J. Marques-Silva, "An Exact Solution to the Minimum Size Test Pattern Problem," *ACM Transactions on Design Automation of Electronic Systems*, vol. 6, no. 4, pp. 629–644, 2001.

[87] H. Lee and D. Ha, "An efficient forward fault simulation algorithm based on the parallel pattern single fault propagation," in *Proc. of International Test Conference*, pp. 946–955, Oct 1991.

[88] K. Butler and M. Mercer, "The Influences of Fault Type and Topology on Fault Model Performance and the Implications to Test and Testable Design," in *Proc. of Design Automation Conference*, pp. 673–678, 1990.

[89] J. Dworak, J. D. Wicker, S. Lee, M. R. Grimaila, M. R. Mercer, K. M. Butler, B. Stewart, and L.-C. Wang, "Defect-Oriented Testing and Defective-Part-Level Prediction," *IEEE Design & Test of Computers*, vol. 18, no. 1, pp. 31–41, 2001.

[90] B. Benware, C. Schuermyer, S. Ranganathan, R. Madge, P. Krishnamurthy, N. Tamarapalli, K.-H. Tsai, and J. Rajski, "Impact of Multiple-Detect Test Patterns on Product Quality," in *Proc. of International Test Conference*, pp. 1031–1040, 2003.

[91] J. Geuzebroek, E. J. Marinissen, A. Majhi, A. Glowatz, and F. Hapke, "Embedded Multi-Detect ATPG and Its Effect on the Detection of Unmodeled Defects," in *Proc. of International Test Conference*, pp. 1–10, 2007.

[92] K. Kantipudi and V. Agrawal, "A Reduced Complexity Algorithm for Minimizing N-Detect Tests," in *Proc. of VLSI Design*, pp. 492–497, 2007.

[93] E. J. McCluskey and C.-W. Tseng, "Stuck-Fault Tests vs. Actual Defects," in *Proc. of International Test Conference*, pp. 336–343, 2000.

[94] J. Nelson, J. Brown, R. Desineni, and R. Blanton, "Multiple-detect ATPG based on physical neighborhoods," in *Proc. of Design Automation Conference*, pp. 1099–1102, 2006.

[95] I. Pomeranz and S. Reddy, "On the Use of Fault Dominance in n-Detection Test Generation," in *Proc. of of IEEE International VLSI Test Symposium*, pp. 352–357, 2001.

[96] I. Pomeranz and S. M. Reddy, "Definitions of the Numbers of Detections of Target Faults and Their Effectiveness in Guiding Test Generation for High Defect Coverage," in *Proc. of Design Automation and Test in Europe*, pp. 504–508, 2001.

[97] I. Pomeranz and S. M. Reddy, “Forming N-detection test sets without test generation,” *ACM Transactions on Design Automation of Electronic Systems*, vol. 12, no. 2, p. 18, 2007.

[98] C.-W. Tseng and E. J. McCluskey, “Multiple-output Propagation Transition Fault Test,” in *Proc. of International Test Conference*, pp. 358–366, 2001.

[99] S. Venkataraman, S. Sivaraj, E. Amyeen, S. Lee, A. Ojha, and R. Guo, “An Experimental Study of N-Detect Scan ATPG Patterns on a Processor,” in *Proc. of VLSI Test Symposium*, pp. 23–28, 2004.

[100] V. Dabholkar, S. Chakravarty, I. Pomeranz, and S. Reddy, “Techniques for Minimizing Power Dissipation in Scan and Combinational Circuits During Test Application,” *IEEE Transactions on CAD*, vol. 17, no. 12, p. 1325, 1998.

[101] H. Lee and D. Ha, “Atalanta: an Efficient ATPG for Combinational Circuits,” tech. rep., Dept of Electrical Eng., Virginia Polytechnic Institute and State University, 1993.

[102] R. Adapa, S. Tragoudas, and M. Michael, “Evaluation of Collapsing Methods for Fault Diagnosis,” in *Proc. of IEEE International VLSI Test Symposium*, pp. 439–444, 2006.

[103] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*. McGraw-Hill, 1990.

[104] K. C. Y. Mei, “Bridging and stuck-at faults,” *IEEE Transactions on Computers*, vol. 23, no. 7, pp. 720–727, 1974.

[105] H. Tang, G. Chen, S. M. Reddy, C. Wang, J. Rajski, and I. Pomeranz, “Defect aware test patterns,” in *Proc. of Design Automation and Test in Europe*, pp. 450–455, 2005.

[106] B. Benware, C. Schuermyer, S. Ranganathan, R. Madge, P. Krishnamurthy, N. Tamarapalli, K.-H. Tsai, and J. Rajski, “Impact of Multiple-Detect Test Patterns on Product Quality,” in *Proc. of International Test Conference*, pp. 1031–1040, 2003.

[107] J. Dworak, M. R. Grimaila, S. Lee, L.-C. Wang, and M. R. Mecer, “Enhanced DO-RE-ME Based Defect Level Prediction Using Defect Site Aggregation-MPG-D,” in *Proc. of International Test Conference*, pp. 930–939, 2000.

[108] J. Dworak, B. Cobb, J. Wingfield, and M. R. Mercer, “Balanced excitation and its effect on the fortuitous detection of dynamic defects,” in *Proc. of Design Automation and Test in Europe*, pp. 1066–1071, 2004.

[109] M. R. Grimaila, S. Lee, J. Dworak, K. M. Butler, B. Stewart, H. Balachandran, B. Houchins, V. Mathur, J. Park, L.-C. Wang, and M. R. Mercer, “REDO - Probabilistic Excitation and Deterministic Observation - First Commercial Experiment,” in *Proc. of VLSI Test Symposium*, pp. 268–274, 1999.

VITA

Contact Information:	Stelios Neophytou E-mail: sneophytou@ucy.ac.cy, sneophytou@hotmail.com

Education

2003 - 2009	PhD candidate at Electrical and Computer Engineering Department, University of Cyprus.
1998 - 2003	Engineering Diploma from Computer Engineering and Informatics Department, University of Patras, Greece. Grade: 8.80 (distinction).

Publications

1. M. K. Michael, S. Neophytou, and S. Tragoudas, "Functions for Quality Transition Fault Tests", *in Proc. of IEEE International Symposium on Quality of Electronic Design*, pp. 327-332, 2005.
2. S. Neophytou and M. K. Michael and S. Tragoudas, "Test Set Enhancement for Quality Transition Faults using Function-based Methods", *in Proc. of the 15th IEEE/ACM Great Lakes Symposium on VSLI*, pp. 182-187, 2005.
3. S. Neophytou, M. K. Michael, and S. Tragoudas, "Efficient Deterministic Test Generation for BIST Schemes with LFSR Reseeding", *in Proc. of IEEE International On-Line Testing Symposium*, pp. 43-50, 2006.
4. S. N. Neophytou, M. K. Michael, and S. Tragoudas, "Functions for Quality Transition-Fault Tests and Their Applications in Test-Set Enhancement", *IEEE Transactions on*

Computer-Aided Design of Integrated Circuits and Systems, vol. 25, no. 12, pp. 3026-3035, Dec. 2006.

5. S. Neophytou and M. K. Michael, “Hierarchical Fault Compatibility Identification for Test Generation with a Small Number of Specified Bits”, *in Proc. of IEEE Defect and Fault Tolerance Symposium*, pp. 439-447, Sep. 2007.

6. S. Neophytou and M. K. Michael, “Two New Methods for Accurate Test Set Relaxation via Test Set Replacement”, *in Proc. of IEEE International Symposium on Quality of Electronic Design*, pp. 827-831, Mar. 2008.

7. S. Neophytou and M. K. Michael, “On the Relaxation of N-detect Test Sets”, *in Proc. of IEEE VLSI Test Symposium*, pp. 187-192, Apr. 2008.