THE UNIVERSITY OF CALGARY

A GLOBAL TEST GENERATION SYSTEM FOR SEQUENTIAL CIRCUITS

by

Bin Du

A DISSERTATION
SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE
DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

CALGARY, ALBERTA

DECEMBER, 1997

0-612-34670-6

ABSTRACT

With the rapid progress of VLSI technology. integrated circuit complexity increases greatly. The reliability of a chip is very important to VLSI engineering and even our daily life. Due to the increase in size and complexity of circuits placed on a chip. it is very difficult to test a chip at an affordable cost.

Automatic Test Pattern Generation (ATPG) plays an important role in VLSI technology. Reduction in test application time and test set size is highly desirable for the reduction of the overall cost in integrated circuit fabrication and testing. Testing of digital circuits involves the generation of a set of test vectors and their application to detect faults in the circuits. An important part of testing is the creation of effective test vectors. The test generation problem for sequential circuits is known to be a difficult task. It is difficult to achieve a significant breakthrough in realizing efficient ATPG algorithms to test large sequential circuits.

A global test generation approach for sequential circuits. called GLOBALTEST. is presented in this dissertation. Global test generation is formulated as the problem of tracing different sensitive paths from the primary inputs and present state lines to the primary outputs and next state lines in a circuit. It consists of two parts: a fault-independent test generation algorithm and a fault-oriented test generation algorithm. At first. a new backward assignment method is presented to extract the ON/OFF sets of the primary outputs and next state lines by partitioning circuits. The combinational test vectors can be extracted at the same time. State justification and state differentiation are efficiently performed using the ON/OFF sets of the primary outputs and next state lines. To enhance the efficiency of state differentiation in the existing three-phase ATPG. a backward deterministic method for state differentiation is proposed and the order of next state lines in state differentiation is presented. The fault-independent test generation algorithm can detect most of the testable faults in the sequential circuits. A disadvantage of the method is that it can not determine the redundant faults.

The fault-oriented test generation algorithm is developed to detect the remaining faults and determine the redundant faults. The recent advances in combinational test generation based on Boolean satisfiability and transitive closure provide a powerful method for test generation. We extend Boolean satisfiability and transitive closure to sequential circuit test generation.

Test compaction is an important part in test generation. We formulate the test compaction problem as the set covering problem. An efficient set covering algorithm (HICOMPACT) is proposed to compact test vectors. It attempts to select the necessary test vectors for the faults detected and eliminate other redundant test vectors so the original fault coverage is not compromised. This method generates a compact testing sequence for a given fault.

The global test generation algorithm for sequential circuits is tested using the ISCAS'89 sequential benchmark circuits. The proposed algorithm has yielded a high fault coverage and provided time efficient procedures to generate tests for large sequential circuits. The experimental results are compared with other existing test generation systems.

Acknowledgement

First of all. I want to thank my supervisor, Professor Jun Gu. for his advice and support during my graduate study. It is my greatest fortune to have Jun as my advisor. I have benefited tremendously from his guidance and insight over the years. I have learnt a lot of engineering optimization methods, and these methods are quite effective. I have been convinced that Jun is a dedicated educator and first class researcher. I am very grateful for his constant encouragement and his perspectives on life at large.

I would like to thank Professor Dave Halliday, Professor Jim Haslett. Professor Guojun Liu. and Professor Xiaoling Sun for their careful reading of the dissertation and many helpful suggestions. I want to specially thank all the faculty members at the department of electrical and computer engineering for providing an excellent education and a rewarding experience. In particular. I thank Professor R. A. Stein for his help over the years. I am also grateful to Professor Danny H.K. Tsang for many valuable discussions.

I have made a number of good friends during my stay in Hong Kong. I thank all my friends for their friendship, in particular, Wei Xiong, Yong Sun, Lixin Wang. We witnessed the transition of Hong Kong to China together.

I am greatly indebted to my wife, Xiaoying Liu for her encouragement and support over these years. I also thank the Liu family for their constant support.

Finally, I specially thank my family for their love and support. Mere words can not express my gratitude to my parents. My mother and father have given me everything it takes to be here. My sisters have been a source of inspiration and given me a lot of support.

To

my Mom. Dad. Xiaoying, Kaixiang. Wen. Heng, and Min

TABLE OF CONTENTS

CHAPTERS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Very Large Scale Integration (VLSI) is the process of integrating hundreds of thousands of semiconductor components and interconnections into a monolithic integrated circuit. As a result of the continuous progress in integrated circuit fabrication techniques. the complexity of digital systems which can be implemented on a single micro-electronic chip has increased. A major problem. one which is growing in importance. is testing. Since VLSI has been widely used in many application areas. ranging from consumer products to critical controllers. the reliability of VLSI circuits is of paramount importance. The problems associated with testing of VLSI circuits have been exacerbated with the rapid advances in VLSI technology. With little or no increase in the number of input/output (I/O) pins. more logic must be accessed with almost the same number of I/O pins. i.e.. reduction in the pin-to-gate ratio. making it much more difficult to test a VLSI chip.

As a consequence of growing circuit complexity. testing is taking an increasingly larger proportion of total product cost. Ironically. the very software design tools that make it possible to put more circuits on a chip at a reduced cost are effectively increasing the cost of circuit testing. The advantages of VLSI are reduced system cost. good performance. and great reliability. These advantages would be lost unless VLSI devices can be tested economically.

1.1 Background

The manufacturing of a VLSI chip consists of fabrication and testing. Testing is required in order to discover defects in the VLSI chip. Design and test development precede manufacture. Test activities are interwoven with the VLSI design process. Architectural design consists of the partitioning of a VLSI chip into realizable blocks. With the increase in complexity of the VLSI system, architectural design becomes more and more important. Either the logic should be synthesized in a testable form or the synthesized logic should be analyzed and improved for testability.

The objectives in testing a VLSI design are twofold. The first is to verify logic correctness and timing behavior of the circuit before fabrication. The second is to determine, after fabrication, whether components and interconnections on the chip are fabricated correctly. The testing after fabrication is, by far, the most pressing problem confronting both designers and test engineers. It is considered as the major obstacle to the full exploitation of the benefits obtained from realizing extremely complex VLSI systems [55]. These tests should thoroughly check every node in the circuit and ideally, cover all faults that can possibly occur during fabrication. In this dissertation, we concentrated on the second kind of tests.

In VLSI circuit design, the testing process is referred to as test generation and fault simulation. The goal of test generation is to obtain test vectors of high quality at an affordable cost. The quality of a set of test vectors is measured by *fault coverage* (a fraction of the modeled faults detected by the test vectors) and by *test length* (the number of vectors in the test set). Given a set of faults and a set of test vectors, the goal of fault simulation is to determine which faults can be detected by the test vectors. Both test generation and fault simulation rank equally in importance and complement one another. Test vectors capable of distinguishing between good circuits and faulted circuits do not become effective until these vectors are simulated so that their effects can be determined. Conversely, extremely accurate simulation with very precise models, and poor test vectors. will not effectively detect many defects.

Various factors contribute to testing and its cost. Testing cost is determined mainly by the cost of real time test pattern generation and test application. The cost of test pattern generation depends on the computer time required to run the test pattern generation program. The cost of test application is determined by the cost of equipment plus the testing time required to apply the test. The testing time may be assumed to be directly proportional to the number of tests. For combinational circuits. a test is a test vector. For sequential circuits. a test is a sequence of test vectors.

A straightforward method for determining the testability of a circuit is to use an Automatic Test Pattern Generation (ATPG) program. It generates test vectors and determines the fault coverage. The running time of the program. the number of test patterns generated, and the fault coverage provide a measure of the testability of the circuit.

1.2 Fault Models

Fault models are a means of describing the effects of defects in a circuit. The accuracy of the fault models in emulating the operation of the circuit under fault conditions alters the effectiveness of test patterns in detecting faulty devices. The test fault models are identified and formulated in this section.

1.2.1 Faults in VLSI Systems

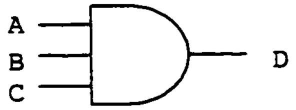
The testing of a digital logic circuit involves the application of stimuli to the circuit and the measurement of the response to determine whether the circuit is functionally correct. An important part of testing is the creation of effective stimuli. A fault is a physical or intellectual defect such as an open circuit, a short circuit, or a ground in a circuit, component, or line [38]. A defect in a circuit may cause either a permanent fault or an intermittent fault. Usually, testing is performed primarily to detect permanent faults. The most commonly occurring faults are modeled. The *fault model* is a computer model of the circuit that has been modified to conform to some premise or conjecture about real physical defects. Then, input stimuli are created which can distinguish between the fault-free and the faulted models. This approach has a number of advantages [40]:

- It is effective to create specific tests for faults most likely to occur.
- The effectiveness of the test set can be measured by determining how many faults can be covered by the set of test vectors.
- Specific defects can be associated with specific test patterns. If a circuit under test responds to a test pattern incorrectly, the information indicates that the defects exist.

This method has become a standard approach to developing tests for digital logic failures.

It is desirable to describe faults at various levels of abstraction in VLSI systems. A fault which is described at a very low level, e.g., the transistor level, may accurately describe the physical phenomena causing the fault, but one of the difficulties with this level is the tedious task of analyzing each individual component in the circuit. Further complicating the task is the fact that there are several technologies in use and each has its own way to perform digital logic operations.

Logic symbols have long been used to represent logic circuits. These symbols reduce the complexity of the logic circuit drawings and have the advantage of being technology-independent. Consider the logic diagram of an AND gate and its truth table as an example, as shown in Figure 1.1.



A	B	C	D
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Figure 1.1. Three input AND gate with its truth table.

With these symbols. the circuits can be logically represented at a higher level. i.e.. the gate level. The faults can be described at the gate level and it would be simpler to consider the faults at that level. An important advantage of this representation is the fact that a computer algorithm can be designed upon these logic operators. which are, for most part, independent of the particular technology chosen to implement the circuits.

1.2.2 Fault Models

Fault models are used to describe the effect of a defect or failure in a circuit. The development of a suitable fault model is a complex task and requires a knowledge of circuit design. logic design, and fault testing. One of the earliest and still widely used fault models at the gate level of abstraction is the *stuck-at* model. In this model, it is assumed that physical defects and faults will result in the lines at the logic gate level of the circuit being permanently stuck at logic value 0 or 1. This model is popular since many defects at the transistor level can be modeled at the gate level. It has proven to be an effective measure of test quality.

It is impractical to test the combinations of all the stuck-at faults in a circuit. This has led to the adoption of the *single-fault* assumption. When a test is attempted. it is assumed that only a single fault exists at a time.

Consider a circuit containing nodes which interconnect various components in the circuit. At one time. each node may have only one of the following results:

- Fault-free.
- Stuck-at-1. i.e.. s-a-1.
- Stuck-at-0. i.e.. s-a-0.

The stuck-at-1 fault inhibits the node from switching to a 0. while the stuck-at-0 fault inhibits switching to a 1. The single stuck-at fault model is adopted in this dissertation.

1.2.3 Fault Equivalence and Dominance

In building fault lists, it is often observed that some faults are indistinguishable from others. In Figure 1.1. faults A. B. or C stuck-at 0 would result in the output D being permanently 0 and, therefore. it is impossible to distinguish between an input stuck-at 0 from the output stuck-at 0. These faults are said to be *equivalent.* There is no test that can distinguish between them. More precisely. if T_a is the set of tests which detect fault a and T_b is the set of tests which detect fault b. and if $T_a = T_b$. then it is impossible to distinguish fault a from fault b.

When we test for inputs, e.g., A, B or C s-a-1, we simultaneously test for the output D s-a-1. A s-a-1 fault on the output, however, prevents one from testing any of the input s-a-1 faults. It is said that the output D s-a-1 fault dominates the input s-a-1 fault. In general, fault a *dominates* fault b if T_b is included in T_a. According to this definition, if fault a dominates fault b, then any test which detects fault b will detect fault a.

Since circuit testing time is affected by the size of the fault list, the reduction of the fault list, a process called *fault collapsing*, can reduce test generation and simulation time. Therefore, fault equivalence and dominance relations can be used to reduce the size of fault lists.

1.3 Test Generation and Its Problems

The objective of test pattern generation is to derive input vectors to the circuit which will excite the circuit in such a way that if any faults are present in the circuit the output response of the circuit will differ from that of the fault-free circuit. With the progress of VLSI technology, the problem of fault testing for logic circuits is becoming increasingly difficult. Different approaches and much research work have been applied to deal with the test problem. Despite the maturity of test generation, the testing of VLSI circuits is still considered to be an area with a number of unsolved problems [37].

1.3.1 NP-Completeness of Combinational Test Generation

Ibarra and Sahni [28] showed that test generation for combinational circuits belongs to the class of NP-complete problems. This strongly suggests that no test generation algorithm with a polynomial time complexity is likely to exist. The problem of combinational circuit test generation can be viewed as a finite space search problem [19]. For a combinational circuit with m primary inputs. there exist 2^m combinations of input assignments. Therefore. it is impossible to exhaust all the combinations for large size circuits.

In practice. test generation algorithms for combinational circuits appear to be able to achieve lower *average* time growth by using heuristic search techniques. Up to now. some well-known test generation algorithms for combinational circuits. such as D-algorithm [53]. PODEM [19]. FAN [16]. NEMESIS [32]. and TRAN [9]. have been developed. Some of them perform well for certain circuit structures.

1.3.2 Test Generation Problems in Sequential Circuits

Test generation for sequential circuits has long been recognized as a difficult task [7. 39]. It remains to be a challenge in spite of a history of attempts dating back to the late 1960s. One new factor which complicates the task of creating tests for sequential circuits is the presence of memory elements. The outputs of the sequential circuit depend not only on the primary input vectors but also on the present states of the circuit.

For combinational circuits. it is possible. but not necessarily reasonable. to create a complete test for logic faults by applying all possible binary combinations to the inputs. This is not true for sequential circuits with memory elements. Not only may they require more than 2^m tests. they are also sensitive to the *order* in which stimuli are applied. It has been shown [7] that a fault in a general synchronous sequential circuit may require a test sequence of up to 2^{n+1} input test vectors. where n is the number of memory elements in the sequential circuits. Therefore. the search space for sequential circuit test generation is very large.

1.4 Approaches to Test Generation Problems Presented in the Dissertation

This dissertation presents a new global search approach for test generation of sequential circuits. The approach traces different sensitive paths taken by a fault at a primary output or next state line by justifying the fault to the primary inputs and present state lines. Therefore. all fault patterns at the primary inputs and present state lines are generated. During the global test generation process. many faults are considered as candidates to be tested simultaneously. Such process aims at utilize common search spaces for different faults to generate common test sequences. The approach consists of two parts: a fault-independent test generation algorithm and a fault-oriented test generation algorithm.

The fault-independent test generation algorithm is independent of individual fault. It considers all faults simultaneously. At first. by partitioning circuits. a new backward assignment method is presented to extract the ON/OFF sets of the primary outputs and next state lines. The combinational excitation vectors can also be extracted at the same time. So cover extraction and excitation vector generation are combined into one phase. State justification and state differentiation are performed using the ON/OFF sets of the primary outputs and next state lines. To enhance the efficiency of state differentiation in the existing three-phase ATPG. a backward deterministic method for state differentiation is proposed and the order of choosing next state lines in state differentiation is presented. The fault-independent test generation algorithm can detect most of the testable faults in the sequential circuits. A disadvantage of the method is that it is difficult to determine all redundant faults.

The fault-oriented test generation algorithm is developed to detect the remaining faults and determine the redundant faults. Each time. it considers one fault at a time. The Boolean satisfiability and the implication graph algorithms are extended to the sequential circuit test generation. A three-phase test generation for sequential circuits is used in this algorithm.

Test compaction is an important part in test generation. The test compaction problem is formulated as the set covering problem. An efficient set covering algorithm is proposed to compact test vectors. It attempts to select the necessary test vectors for the faults detected and eliminate other redundant test vectors. so the original fault coverage is not compromised. A local reduction and expansion algorithm is then used to further compact the test set. This method generates compact test sequences for the given faults.

1.5 Organization of the Dissertation

The dissertation is organized as follows: Previous work in test generation for combinational circuits and sequential circuits is reviewed in Chapter 2. Some related work in test compaction is also presented.

Chapter 3 introduces the related test generation terminologies used in this dissertation.

In Chapter 4. observations that initiated this research work in test generation for sequential circuits are given. A global search test generation system for sequential circuits is presented. Then the steps of cover extraction. combinational circuit test generation, state justification, and state differentiation used in the system are briefly introduced. The algorithms used in these steps are described in Chapters 5 - 6 in detail.

A backward assignment algorithm for cover extraction is described in detail in Chapter 5. It can efficiently extract the ON/OFF sets of the primary outputs and next state lines and the combinational excitation vectors. A method of partitioning circuits is proposed to enhance the efficiency of extraction.

In Chapter 6, state justification and state differentiation are described. To enhance the efficiency of state differentiation in the existing ATPG system. a backward deterministic algorithm for state differentiation is developed and the order of choosing next state lines in state differentiation is studied.

Test compaction for sequential circuit test generation is discussed in Chapter 7. First. the test compaction problem is formulated as a set covering problem. Then an efficient set covering algorithm is proposed to compact test vectors. A local reduction and expansion algorithm is used to further compact the test vectors.

Experimental results with ISCAS'89 benchmarks are presented in Chapter 8. These results are compared to existing test generation systems. Our algorithm has obtained close to the maximum fault coverage on the most benchmarks.

Chapter 9 concludes this dissertation and discusses possible future work.

CHAPTER 2

OVERVIEW OF EXISTING METHODS

In this chapter, an overview of the work previously done toward test generation is presented. Previous work on the test compaction problem is then reviewed.

Since test generation methods for combinational circuits and sequential circuits are related to each other. previous work on combinational circuit test generation is first briefly discussed.

2.1 Overview of Test Generation for Combinational Circuits

A theoretical study suggests that no test generation algorithm for combinational circuits with polynomial time complexity is likely to exist [28]. Though test generation for combinational circuits is NP-complete. some efficient test generation systems have been developed.

Up to now, some well-known test generation algorithms for combinational circuits have been developed and perform well for certain circuit structures. The existing test generation systems for combinational circuits are divided into two classes: structural methods, such as PODEM [19], and algebraic methods.

2.1.1 Structural Methods

The structural methods derive test vectors from a topological gate description of the circuit. The path sensitization method is one of the frequently used techniques.

Among structural search methods in test generation for combinational circuits. the D-algorithm. developed by Roth [53]. is one of the oldest and the best known test generation algorithms. This algorithm adopts a five-valued 0. 1. $X.D.\overline{D}$ calculus to carry out the sensitization and the line justification procedures in a very formal manner. The faulty line is assigned a D or $\overline{D}$ depending on the fault on the line. The calculus and the circuit structure information are used to determine values on the other lines so that D or $\overline{D}$ can be sensitized to the primary outputs. A line justification step is then carried out to justify the values assigned in the preceding step. Both the sensitization and the line justification steps may have to be applied many times before a test vector is obtained.

A class of circuits for which the D-algorithm performs particularly poorly are those containing exclusive-or trees. Degradation in performance arises due to an excessive amount of backtracking. This observation motivated Goel [19] to devise a new test generation algorithm called *path oriented decision making* (PODEM). A branch and bound technique was used in PODEM. The algorithm starts by assigning a value of 0 or 1 to a selected primary input (PI) line. and then determines its implication on the propagation of D or $\overline{D}$ to a primary output. If no inconsistency is found. it again selects another PI line and assigns a 0 or 1 to it. and then repeats the process. which is referred to as *branching*. If an inconsistency is determined in the branching. the branching stops and bounding starts. The PI line which was most recently assigned a binary value is assigned the complimentary value. and branching starts again. The complete process stops when either a test vector is found or when the fault is determined to be undetectable. PODEM implementations are known to run an order of magnitude faster than the D-algorithm on most circuits.

Fujiwara and Shimono [16] described techniques to further accelerate a path-sensitization algorithm like PODEM. Their algorithm. called FAN. does extensive analysis of the circuit connectivity in a preprocessing step to minimize backtracking. FAN has employed a better heuristic in the bounding-and-branching steps to speed up the test generation process. Schulz. Trischler. and Sarfert [57] presented a unique sensitization method and an improved multiple backtrace method to further improve the performance of FAN. Their system. called SOCRATES. improved the implication procedure.

In these structural methods. backtracking. which is a branch procedure terminated by a bound step. is the most computationally expensive step in the process of searching for a test vector. The branching step goes as deep in the binary search tree as possible. while the bound step backs up in the binary search tree to the most recent node with an unused alternative assignment.

2.1.2 Algebraic Methods

Instead of performing a search on a data structure representing a circuit. algebraic methods produce an equation describing all possible tests for a particular fault and then simplify the resulting equation. A typical algebraic method is the Boolean difference method. proposed by Sellers *et al.* [58]. Once the Boolean difference formula for the testing problem is obtained. it is simplified by using the basic laws of Boolean algebra or using identities specific to the Boolean difference. The tedious nature of the algebraic manipulations involved in solving formulae using the Boolean difference led to its disfavor as a practical tool for test pattern generation [40].

Recently. Larrabee [32] proposed a Boolean satisfiability (SAT) method for generating test vectors for single stuck-at faults in combinational circuits. This new method generates test vectors in two steps. First. it constructs a formula expressing the Boolean difference between the fault-free and faulty circuits. Secondly, instead of performing symbol manipulation, it applies a SAT algorithm to satisfy the formula. This method has, in practice, produced good results for the problem of combinational circuit test generation. Many existing SAT algorithms can be used to solve large size SAT formulas [20, 21, 24, 26].

Later, Chakradhar, Agrawal, and Rothweiler [9] developed a transitive closure algorithm for combinational circuit test generation. A Boolean difference equation is derived from the model of the circuit incorporating necessary conditions for fault activation and path sensitization and then a test vector is obtained by determining signal values that satisfy the equation. The method consists of two main steps that are repeatedly executed: transitive closure computation and decision-making. The transitive closure contains global pairwise (or binary) logical relationships among all signals. Higher-order signal relationships are represented as additional ternary and M-ary ($M > 3$) relations. A key feature of the algorithm is that signal dependencies derived from the transitive closure are used to reduce ternary relations to binary relations that in turn dynamically update the transitive closure. The signals are either determined from the transitive closure or are enumerated until the Boolean equation is satisfied.

An efficient ATPG system for combinational circuits has been implemented in SIS [59]. It consists of two parts: random test generation and deterministic test generation, which is based on the algorithm reported in [32]. Both fault equivalence and fault dominance are used to reduce the fault list.

Cox and Rajski [15] developed a mathematical basis for the identification of necessary and nonconflicting assignment. The algorithmic assignment identification can be used to reduce or eliminate backtracking.

2.2 Overview of Test Generation for Sequential Circuits

Though adequate test generation systems exist for combinational circuits. the same cannot be said for sequential circuits. The major difficulty in test generation for sequential circuits is that the outputs of a sequential circuit depend not only on the primary input vectors but also on the present states of the circuit.

The earlier algorithms modeled sequential circuits as iterative combinational circuits. Some test generation algorithms for combinational circuits were extended to test sequential circuits [31, 41]. An algorithm based on this method has been programmed into a commercial package called LASAR [61]. Several approaches [36. 60] based on extensions of the classical D-algorithm were presented to solve the test generation problem for sequential circuits. Shteingart. Nagle. and Grason [60] gave an efficient technique for modeling sequential components. Although some progress was made in these attempts, an effective solution for circuits with more than a few hundred gates and large sequential depths was not available at that time.

Due to the relative ineffectiveness of these ATPG systems. many large digital systems are being designed in compliance with design-for-testability rules which attempt to reduce the complexity of the test problem. The objective of design-for-testability is to provide guidelines which ensure the creation of testable designs. A popular approach is to make the memory elements controllable and observable. i.e.. a scan design [1]. The flip-flops and/or latches are designed to be able to operate in either parallel load or serial shift mode. In the normal mode of operation. flip-flops and latches are configured for parallel load. The flip-flops are switched to a serial shift mode for testing purposes. In serial mode, any needed test values can be loaded by serially shifting in the desired values. In a similar fashion. any values in the flip-flops can be observed by shifting out their contents while in the serial shift mode.

Scan design approaches have been successfully used to reduce the complexity of the problem of sequential circuit test generation by transforming the problem into one of combinational circuit test generation. However, in some cases, the cost in terms of area and/or performance and/or extra numbers of I/O pins is unaffordable.

For sequential circuit test generation, some progress has been made in the past several years. A heuristic, simulation-based test generation algorithm was presented by V.D. Agrawal, K.-T Cheng, and P. Agrawal [2]. At first, initialization sequence is generated to bring flip-flops to known states. Then, test generation for a group of faults and a single fault is performed separately. All functions are accomplished through a simulator using different cost functions. Vectors are generated to minimize the cost.

Ma, Devadas, Newton, and Sangiovanni-Vincentelli [35] described a PODEM-based deterministic approach to sequential circuit test generation, called STALLION. It first extracts a partial state transition graph (STG) of a sequential circuit. The construction of the partial STG is based on an efficient state-enumeration algorithm that aims at finding paths from the reset state to different valid states (states reachable from the reset state) in the STG. Then test sequences for line stuck-at faults can be generated using the two-phase ATPG system: fault excitation and propagation, and state justification.

Later, a new system, STEED, was proposed by Ghosh, Devadas, and Newton [18] to improve STALLION. STEED decomposes the problem of sequential circuit test generation into three subproblems: excitation vector generation, state justification, and state differentiation. Given a fault under test, it first generates a combinational excitation vector that propagates the effect of the fault to the primary outputs or the next state lines. Combinational excitation vector generation is based on a PODEM-like algorithm. A justification step is then performed to find a justification sequence for the excitation state. This step is carried out using a sequence of cube intersections on the complete or partial ON/OFF sets of the next state lines. If the effect of the fault has been propagated to the next state lines alone. the true-faulty state pair is generated by the excitation vector. A differentiation sequence for this true-faulty state pair is obtained using another sequence of cube intersections on the ON/OFF sets of the primary outputs. The three-phase ATPG system is shown to be an efficient method. STEED significantly improved STALLION in terms of computing time for the same fault coverage. Random test generation has been used as a front end of deterministic test generation. Therefore, random seed has effects on the performance of the random test generation.

Cho. Hachtel. and Somenzi [12] have recently given an efficient algorithm. VERITAS. for sequential circuit test generation. VERITAS is based on implicit state enumeration and a three-phase ATPG. The approach identifies sequential redundancies through reachability analysis of sequential circuits. It constructs the product machine of two sequential circuits to be compared. Reachability analysis is performed by traversing the finite state machine to find any difference in I/O behavior. When an output difference is detected, the information obtained by reachability analysis is used to generate a test sequence. As the product machine traversal (PMT) is quite resource-demanding, a three-phase ATPG system is used first to deal with most of the faults. PMT is used only for the faults for which the three-phase ATPG fails to generate test sequences. VERITAS further improved STEED in terms of running time, test vector length, and fault coverage. It is difficult. however, for VERITAS to handle large size sequential circuits.

These approaches above are capable of generating tests for sequential circuits with

1000-3000 gates. Due to the difficulty of test generation for sequential circuits. significant improvements are needed for very large scale sequential circuits.

Niermann and Patel [42] presented HITECT. a sequential circuit test generator without a reset state. A targeted D element technique is used to increase the number of possible mandatory assignments and reduce the over-specification of state variables. The state knowledge of previously generated vectors for state justification, without the memory overhead of a state transition diagram. is presented.

Lee and Sajula [34] presented a PODEM-based algorithm for sequential circuits. called FASTEST. The iterative array model and nine-valued logic are used. Each iteration is called a time-frame. The FASTEST algorithm determines the number of time-frames required to find a test for a fault and then expends time-frames based on the state of test generation failure.

Prinetto. Rebaudengo. and Reorda [48] developed a genetic-based algorithm. GATTO. for very large sequential circuits. The algorithm starts with a number of random generated sequences. Two operators are used in evolution process: cross-over and mutation. The cross-over operator selects two parent sequences and builds a new sequence composed of the first x_1 vectors of the first sequence and the last x_2 vectors of the second sequence (here x_1 and x_2 are two random numbers). The mutation operator randomly selects a sequence and complements a single bit within it.

Chen and Bushnell [11] proposed a test generator (SEST) for sequential circuits. They observed that test generation for different faults may share identical justification decision sequences represented by identical decision spaces. Since justification decomposition represents the collective effects of prior justification decisions. it is used to identify previously explored justification decisions.

Test cultivation is used for generating test vectors for both combinational and sequential circuits [56]. This method is based on continuous mutation of a given input sequence and on analyzing the mutated vectors to select the test set. The test cultivation algorithms are simulation-based and a test set can be cultivated for any circuit which can be simulated logically.

Marchok, El-Maleh, Maly, and Rajski [37] investigated the complexity of sequential ATPG systems. Three sequential circuit test generators are used in the investigation. It is found that an circuit attribute, termed density of encoding, is a key indicator of the complexity of structural, sequential test generation. Density of encoding is the fraction of the total number of possible states which are valid.

Pomeranz and Reddy [45] analyzed undetectable and redundant faults in sequential circuits. Faults are classied into three sets: detectable, partially detectable, and operationally redundant. The last two classes are dependent on the operation mode of the circuit. Partially detectable faults are the fault for which a test sequence does not exist: however, under certain initial conditions (or initial states) of the circuit, faulty behavior may be observed. The notion of redundancy cannot be separated from the operation mode of the circuit. Two operation modes are considered: synchronization mode and free mode.

Recently, hybrid sequential circuit test generation methods are often adopted. Rudnick and Patel [54] combined deterministic algorithms for fault excitation and propagation with genetic algorithms for state justification. Deterministic algorithms for state justification are used if the genetic approach is failed, so it allows for identification of untestable faults and to improve the fault coverage. Hsiao, Rudnick, and Patel [27] developed an ATPG system for sequential circuits, called ALTTEST, where two phases of test generation are used. The first phase uses a simulated-based genetic algorithm and the second phase uses a deterministic algorithm HITEC.

2.3 Overview of Test Compaction

Test compaction is very important in test generation for both combinational circuits and sequential circuits. since it allows reduction in test application time and test vector storage requirements of VLSI testers. The test compaction process can be performed either statically or dynamically. In static compaction, the merging process is performed after the test generation phase is completed. In dynamic compaction. test vectors are merged during the test generation phase.

Several combinational and sequential test generators aimed at generating small test sets for the stuck-at fault model have been developed. I. Pomeranz, L.N. Reddy. and S.M. Reddy [44] presented two test vector compaction methods for combinational circuits: maximal test compaction and rotating backtrace in COMPACTEST. Maximal test compaction unspecifies some primary input values specified as 1 or 0 in a test vector for a fault. even if the resulting vector is not a test vector for the fault. The rotating backtrace increases the potential of detecting additional faults by selecting different paths for backtracing.

L.N. Reddy. I. Pomeranz, and S.M. Reddy [51] developed the COMPACTEST-II system to generate compact test sets for combinational circuits. Single transition and CMOS stuck-open faults are considered. Vectors for different faults are dynamically overlapped. either fully or partially. to reduce the test set size. In ROTCO [52]. the test vectors are processed in reverse order of generation. Test vectors are allowed to be modified in the detection of faults detected by earlier vectors and the test vector length is reduced.

Kajihara, Pomeranz, Kinoshita, and Reddy [30] presented a cost-effective method for finding maximum independent fault sets in combinational circuits. A dynamic fault ordering technique was introduced using independent fault sets and a double technique that leads to the minimal test sets. Later. they [29] compacted test vectors for stuck-at faults in combinational circuits by addition and removal of test vectors. Each test vector which is added to the test vectors allows the removal of two or more test vectors. Aourid and Kaminska [4] proposed a neural network method for the set covering problem. which can be applied to the test compaction for combinational circuits.

Test vector compaction for sequential circuits is significantly more difficult than for combinational circuits. Based on sequential testability and iterative model. Ben-Hamida. Kaminska. and Savaria [5] discussed a pseudo-random vector compaction technique for sequential circuits in order to conserve the sequences of vectors that detect faults. Sequential testability measures are used to predict the optimum amount of circuit duplication which is used in test compaction.

Raghunathan and Chakradhar [49] present several accelerating techniques for dynamic test compaction for sequential circuits. These techniques are based on the identification of support sets. target fault switching. and use of dynamic equivalent and untestable fault analysis. They [10] also proposed a dynamic vector compaction and test cycle reduction algorithm to identify bottlenecks that prevent compaction and cycle reduction.

Niermann. Roy. Patel. and Abraham [43] used the compatibility of test sequences to compact test vectors for sequential circuits. The compaction technique has effect on the original fault coverage.

Pomeranz and Reddy [46] presented a procedure to generate short test sequences for sequential circuits by selecting one input combination at a time. The procedure switches between a fault-independent phase and a fault-oriented phase. Later. they [47] used some static compaction techniques. such as omission and insertion. to do dynamic test compaction for sequential circuits.

2.4 Summary

Up to now. some well-known test generation algorithms for combinational circuits have been developed and perform well for certain circuit structures. Existing ATPG systems for combinational circuits fall into two classes: structural and algebraic methods. Both Boolean satisfiability and transitive closure methods have produced good results on popular test pattern generation benchmarks [32, 9].

For sequential circuit test generation. some progress has been made in the past several years [18, 42. 12. 11]. The three-phase ATPG system is shown to be an efficient method. Due to the difficulty of test generation for sequential circuits. significant improvements are needed for very large scale sequential circuits.

Test compaction is an important part in an ATPG system for both combinational and sequential circuits. There are two kinds of methods to perform test compaction: static and dynamic test compaction [51. 30. 49].

CHAPTER 3

PRELIMINARIES

In this chapter. common terminology related to test generation is introduced. A sequential circuit is shown in Figure 3.1. The circuit consists of a combinational logic block and some feedback flip-flops. The inputs and outputs of flip-flops are the next state and present state lines. respectively. There are p primary inputs. n present state lines. n next state lines. and q primary outputs. The primary outputs are the functions of the primary inputs and present state lines. It is assumed that the present state and next state lines are neither controllable nor observable. The goal of test generation for sequential circuits is to find primary input sequences which can propagate the faults in the sequential circuit to the primary outputs.

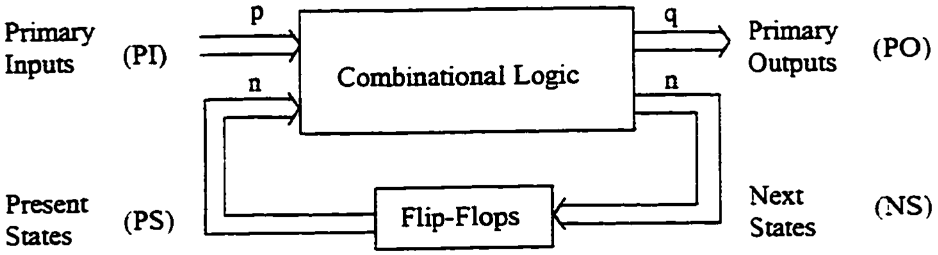


Figure 3.1. A sequential circuit.

A conventional *iterative array model* [7]. as shown in Figure 3.2. is used to illustrate the test generation process of sequential circuits. The iterative array in

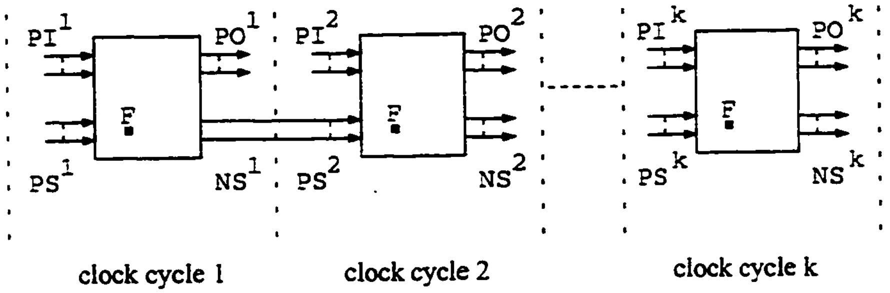


Figure 3.2. An equivalent pseudo-combinational iterative array to the sequential circuit in Fig. 2.1.

Figure 3.2 is logically equivalent to the sequential circuit shown in Figure 3.1. If an input sequence $PI^1, PI^2, \cdots, PI^k$ is applied to the sequential circuit in initial present state PS^1, it generates a primary output sequence $PO^1, PO^2, \cdots, PO^k$ and the next state sequence $NS^1, NS^2, \cdots, NS^k$ $(PS^{i+1} = NS^i, 1 \leq i < k)$.

Assume there is a fault, F, in the combinational logic block of the sequential circuit shown in Figure 3.1. The combinational block is duplicated in terms of each clock cycle, i.e., time-frame. Two iterative array models are considered in test generation: the fault-free and faulty array models. In the faulty array model, the fault under test exists in every time frame. The behavior difference between the fault-free and faulty array models is the effect of the fault on the primary outputs, since other nodes besides the primary outputs are neither observable nor controllable. The fault is propagated to the primary output along a sensitized path through each time frame.

Definition 3.1 Beginning with the present state in clock cycle 1, PS^1, we set the reset state values and wish to produce a primary input sequence, $PI^1, PI^2, \cdots, PI^k$, which, when applied to clock cycles $1, 2, \cdots, k$, propagates the effect of the fault F to the primary outputs, PO^k, during the kth clock cycle. This primary input sequence is called a *test sequence* for the fault.

Unlike combinational circuits, where only one input test vector is needed to test a fault, a sequential circuit may require a test sequence of up to 2^{n+1} input test vectors, where n is the number of memory elements (flip-flops) in the sequential circuits [7].

In sequential circuit testing, a *state* is a bit vector. Its length is equal to the number of memory elements in the sequential circuit. In general, a state is a *cube*, i.e., the values at the different bit positions may be 0, 1 or X (don't care). A *minterm* state is a state with only 0's or 1's as bit values. A cube state is a group of minterm states. A *universal cube* is a cube with all X entries.

Definition 3.2 State S_1 *implicates* state S_2, if and only if, every state contained in S_1 is also contained in S_2. That is, state S_2 *covers* state S_1.

For example, state (1, 0, 0) is a minterm state, and state (1, 0, X) is a cube state. There are two minterm states (1, 0, 0) and (1, 0, 1) in the state (1, 0, X), so state (1, 0, 0) implicates state (1, 0, X). (X, X, X) is a universal cube.

A nine-value logic (0, 1, X, D, $\overline{D}$, FD, $F\overline{D}$, TD, $T\overline{D}$) is used to describe the circuit behavior. The logic value D represents a logic value 1 for a node in a fault free circuit and a logic value 0 for the same node in the faulty circuit. The logic value $\overline{D}$ is the complement value of D. The logic value FD is the consistent value of the logic value D and the logic value 0. The logic value $F\overline{D}$ is the consistent value of the logic value $\overline{D}$ and the logic value 0. The logic value TD is the consistent value of the logic value D and the logic value 1, and the logic value $T\overline{D}$ is the consistent value of the logic value $\overline{D}$ and the logic value 1, as shown in Figure 3.3.

Definition 3.3 The logic values 0, 1, and X are referred to as the *control logic values*. The logic values D, $\overline{D}$, FD, $F\overline{D}$, TD, $T\overline{D}$ are referred to as the *fault logic values*.

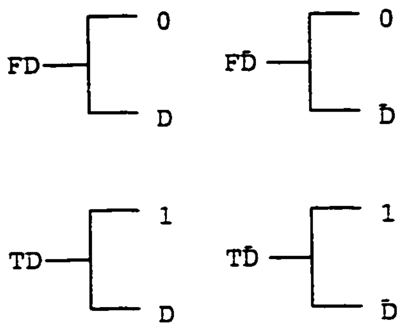


Figure 3.3. The meaning of logic values FD, $F\overline{D}$. TD. $T\overline{D}$.

The sequential circuits discussed here are assumed to have a *reset state*. All test sequences are applied to the sequential circuit with the reset state as the starting state. Some faults in the circuit may be *redundant*, i.e., their existence does not change the behavior of the circuit. There are two kinds of redundant faults. combinational redundant and sequential redundant.

Definition 3.4 A *combinational redundant* fault is a fault which cannot be propagated to the primary outputs or the next state lines. beginning from any state. with any input vector.

Definition 3.5 A *sequential redundant* fault is a fault which cannot be excited or whose effect cannot be propagated to the primary outputs using any sequence of input vectors starting from the reset state.

Definition 3.6 An *excitation vector* for a fault is a test vector that propagates the fault to either the primary outputs or the next state lines. The test vector consists of two parts, the primary input and the present state. The present state part of an excitation vector is called an *excitation state*. The primary input part of an excitation vector is an *excitation input*.

Definition 3.7 The process of finding a primary input sequence which takes a circuit from the reset state into the excitation state is called *state justification*. The corresponding input sequence is a *justification sequence*.

There are two kinds of state justification. *forward* state justification and *backward* state justification. In forward state justification, the search is done from the reset state to the excitation state; and vice versa for backward state justification. Usually. backward state justification is used to perform state justification. If the excitation vector propagates the fault to the next state lines. *state differentiation* is required.

Definition 3.8 *State differentiation* is the process of propagating the effect of a fault on the next state lines to the primary outputs. A *differentiation sequence* for a pair of states. fault-free state S^T and faulty state S^F. which differ in at least one bit. is a primary input sequence such that. if the circuit is initially in S^T. the last vector in the sequence produces a different logic value in at least one primary output than if the circuit were initially in S^F.

In sequential circuit test generation. the complete test sequence is obtained by combining the justification sequence, the excitation vector. and the differentiation sequence.

Definition 3.9 When all flip-flops in a sequential circuit are disabled. the sequential circuit becomes a *pseudo-combinational* circuit. The primary inputs and present state lines are considered as the inputs of the pseudo-combinational circuit. The primary outputs and next state lines are the outputs of the pseudo-combinational circuit.

A general pseudo-combinational circuit is shown in Figure 3.4.

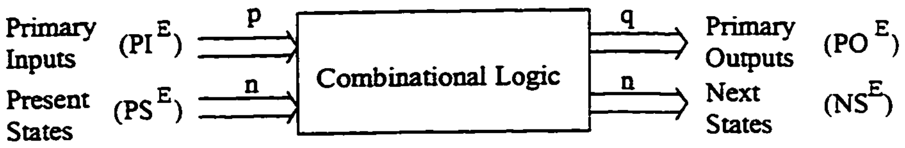


Figure 3.4. A general pseudo-combinational circuit is obtained from the corresponding sequential circuit by disabling all flip-flops.

Definition 3.10 The *input cone* of a primary output is a portion of a circuit which includes the primary output and its subtree from the primary output to the primary inputs and present state lines. Any fault site in this subtree is a *node* in the input cone. The *input cone* of a next state line is a portion of a circuit which includes the next state line and its subtree from the next state line to the primary inputs and present state lines. The *input cone* of a node is a portion of a circuit which includes the node and its subtree from the node to the primary inputs and present state lines. The *size* of a node is the number of nodes in its input cone.

The concept of the input cone is suitable for any node in a circuit and includes the input cones of primary outputs and next state lines.

Definition 3.11 Assuming a fault site in the circuit is a node in the input cones of r primary outputs. the input cones of these r primary outputs are referred to as the *primary output fault region* for the fault under test. Similarly. if a fault site in the logic circuit is a node in the input cones of s next state lines. the input cones of these s next state lines compose the *next state fault region* for the fault under test.

To illustrate the idea of an input cone. consider a simple sequential circuit s27 from the ISCAS'89 benchmarks. as shown in Figure 3.5. There is only one primary output G17. and its input cone is shown in Figure 3.6. There are three next state lines G10. G11. and G13. The input cones of the next state lines G10. G11. and G13 are shown in Figure 3.7 - 3.9. Consider a fault on G15. Since G15 is a node in the input cone of the primary output G17. its primary output fault region is the same as the input cone of G17 shown in Figure 3.6. Though G15 is a node in the input cones of the next state lines G11 and G10. if the fault is propagated to G10. it must be propagated to G11 first. Therefore, the next state line G11 only needs to be

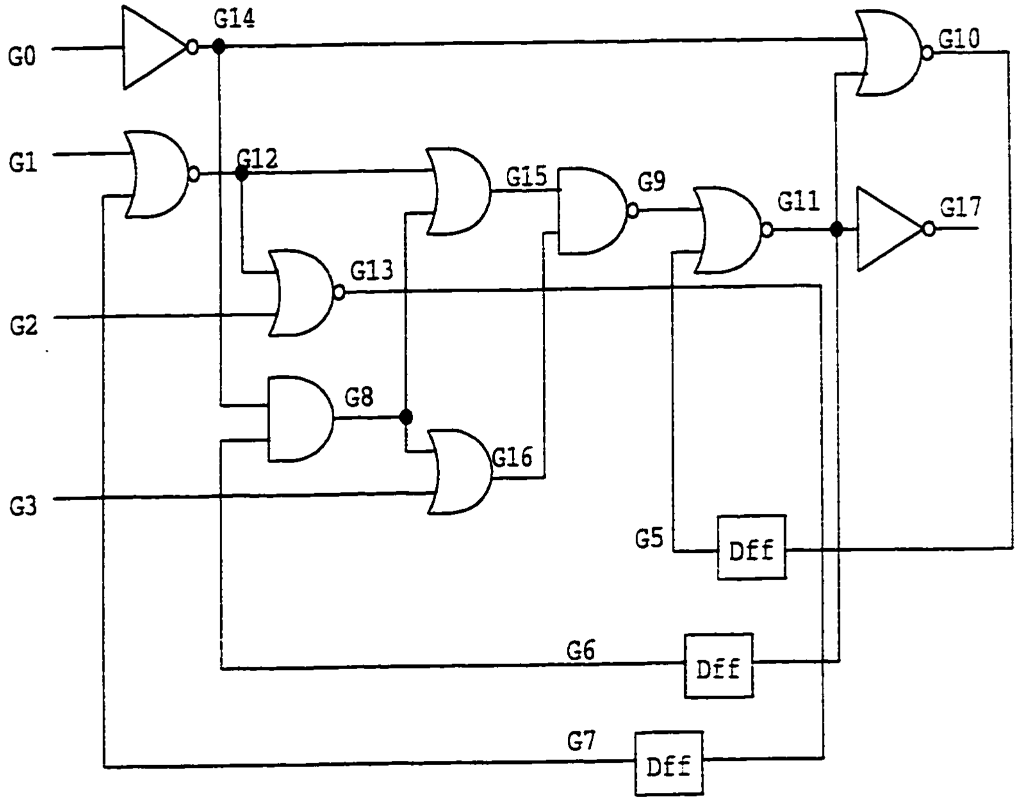


Figure 3.5. A sequential circuit s27 from ISCAS'89 benchmarks.

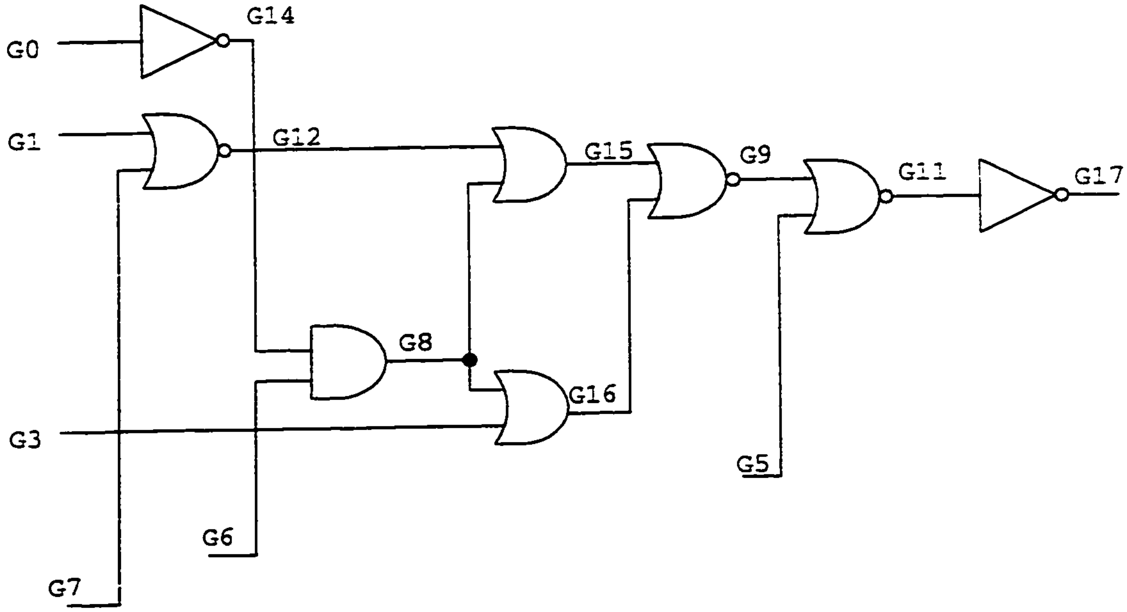


Figure 3.6. The input cone of the primary output G17 in circuit s27.

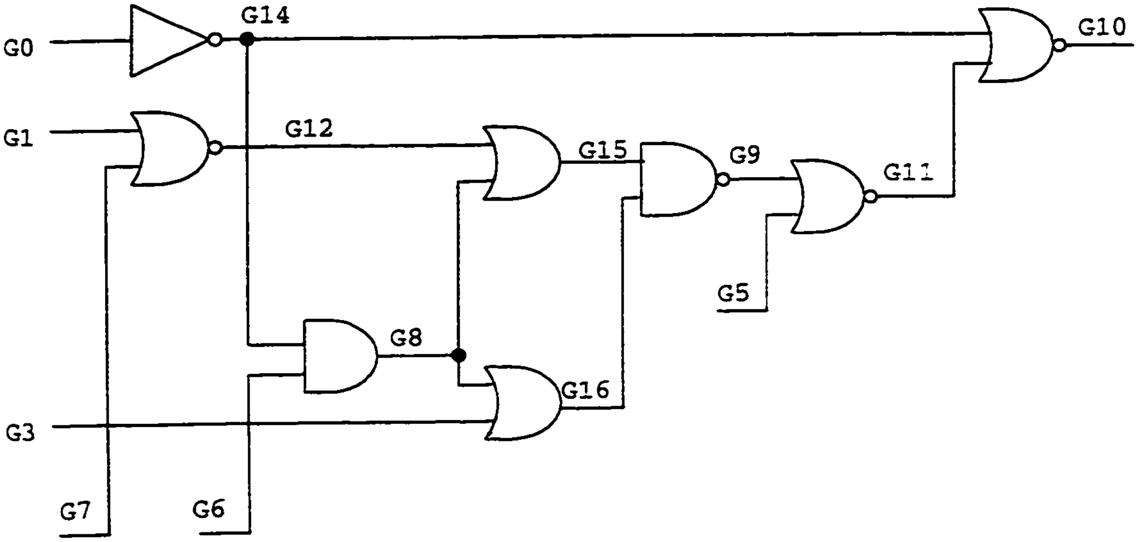


Figure 3.7. The input cone of the next state line G10 in circuit s27.

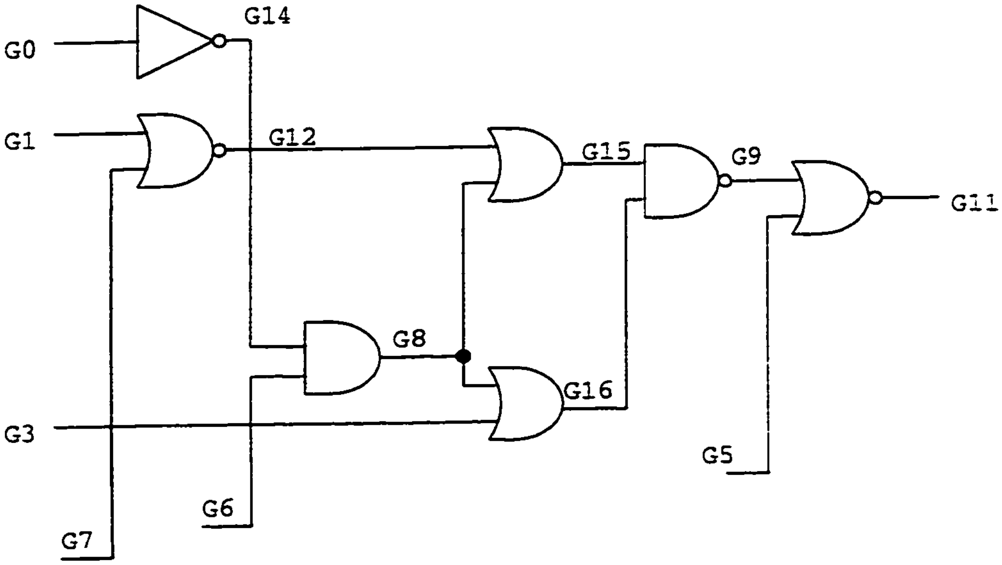


Figure 3.8. The input cone of the next state line G11 in circuit s27.

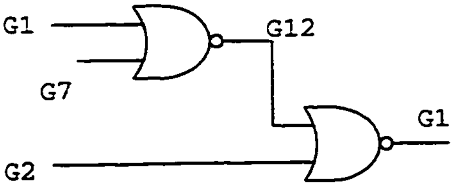


Figure 3.9. The input cone of the next state line G13 in circuit s27.

considered. The next state fault region for the fault at node G15 is the input cone of G11. as shown in Figure 3.8.

A set consists of a group of cubes. There are OFF. ON. and D sets corresponding to logic values 0. 1. and D.

Definition 3.12 The *ON set* of an output is the complete set of input values which produces the output logic value 1. The *OFF set* is the complete set of input values such that the corresponding output is at logic value 0.

The process of extracting the ON/OFF sets of the primary outputs and next state lines is called *cover extraction.*

Definition 3.13 If a sequential circuit can reach a state S_{out} during the next clock cycle from a state S_{in}. state S_{in} is said to be a *fan-in state* of state S_{out}. and state S_{out} is a *fanout state* of state S_{in}.

All fan-in states of a state can be obtained by cube intersection on the ON and OFF sets of the next state lines.

Definition 3.14 The *intersection* of two cubes c and d. denoted $c \cap d$. is the set of states that belong to both c and d.

The cube intersection is performed according to the following equation:

$$c \cap d = \begin{cases} \phi, \text{ if there exists one } k, \ c_k \cap d_k = \phi. \text{ otherwise} \\ \{(c_1 \cap d_1)(c_2 \cap d_2) \cdots (c_n \cap d_n)\}, \end{cases}$$

where $k = 1, 2, ..., n$.

Table 3.1. Cube intersection operation.

			d_i	
	$\cap$	0	1	X
c_i	0	0	$\emptyset$	0
	1	$\emptyset$	1	1
	X	0	1	X

Table 3.2. Sharp product operation.

		d_i		
	#	0	1	X
	0	ε	ϕ	ε
c_i	1	ϕ	ε	ε
	X	1	0	ε

The intersection of the three value tuple is defined in Table 3.1, where o is the empty set.

Definition 3.15 The *union* of two cubes, i.e.. $c \cup d$. is the set of states that belong to c or d.

Definition 3.16 The *sharp product* of two cubes. i.e.. $c\#d$. is the set of states that belong to c but not to d.

$$c\#d = \begin{cases} c. \text{ if there exists one } k. \ c_k\#d_k = o; \\ o. \text{ if } c_k\#d_k = \varepsilon. \text{ for all } k; \text{ else} \\ \cup_k\{c_1c_2...c_{k-1}\alpha_kc_{k+1}...c_n\}. \end{cases}$$

where $c_k\#d_k = \alpha_k \in 0.1$. $k = 1.2.....n$. And ε denotes implication.

The sharp product of the three value tuple is obtained in Table 3.2.

Definition 3.17 The *complement* of a set s, $\bar{s}$, is a set of the same dimension as the set s. such that its components have their on-sets equal to the off-sets of the corresponding components of s. The union of the set s and its complement set $\bar{s}$ equals to the corresponding universal cube and their intersection is the corresponding empty set.

Definition 3.18 A *graph* $G = (V, E)$ consists of a finite. nonempty set of *vertices* V and a set of *edges* E. If the edges are ordered pairs (v, w) of vertices, the graph is said to be *directed*; v is called the *tail* and w the *head* of the edge (v, w).

Definition 3.19 A *path* is a sequence of edges of the form (v_1, v_2), (v_2, v_3), ..., (v_{n-1}, v_n). We say that the path is *from* v_1 *to* v_n and is of *length n-1*. A *cycle* is a simple path of at least length 1 which begins and ends at the same vertex.

If a graph contains a cycle, it is cyclic: otherwise it is acyclic. A *Directed Acyclic Graph* (DAG) can be used to describe a circuit.

Definition 3.20 The *transitive closure* of G is defined as a graph $G*$ which has the same vertex set as G, but has an edge from v to w if and only if there is a path from v to w in G.

Definition 3.21 The edges V can be partitioned into equivalence classes V_i, $i \geq 1$. such that vertices v and w are equivalent if and only if there is a path from v to w and a path from w to v. The graphs $G_i = (V_i, E_i)$ are called the *strongly connected components* of G.

The goal of the satisfiability (SAT) problem [13] is to determine whether there exists an assignment of truth values to a set of variables $(x_1, x_2, ..., x_m)$ that makes the following Boolean formula satisfiable:

$$c_1 \cdot c_2 \cdot ... \cdot c_n, \tag{3.1}$$

where $\cdot$ is a logic *and* connector and $c_1, c_2, \cdots, c_n$ are n distinct clauses. Each clause consists of only literals combined by just the logic *or* (+) connector. A literal is a variable or a single negation of a variable.

CHAPTER 4

GLOBAL TEST GENERATION FOR SEQUENTIAL CIRCUITS

In this chapter. an efficient global search system for test generation of sequential circuits is presented. The system consists of two parts: a fault-independent test generation algorithm and a fault-oriented test generation algorithm. A novel backward assignment method is developed to extract the ON/OFF sets of the primary outputs and next state lines. The pseudo-combinational excitation vectors can be extracted at the same time. A method of partitioning circuits is developed to increase the efficiency of extraction. For each excitation vector. state justification is used to justify the excitation state. If the excitation vector propagates the fault to the next state lines. state differentiation is required to continuously propagate the effect of the fault on the next state lines to the primary outputs. A backward deterministic method for state differentiation is presented and the order of choosing the next state lines in state differentiation is given.

At first. observations that initiated this research work in sequential circuit test generation are given. A model of global test generation for sequential circuits is discussed. Then a novel global search test generation system for sequential circuits is presented. Three important parts in this system are discussed in the following chapters 5 - 7. They are the extraction of cover sets and combinational excitation vectors, state justification and state differentiation, and test compaction of sequential circuits.

4.1 Observations

Some existing test generation systems use random test generation as a front end of deterministic test generation to reduce test generation time [18. 12. 48]. This results in a two-step sequential circuit test generation system: random test generation and deterministic test generation. With the increasing complexity of sequential circuits. fewer percentage of faults can be detected by random test generation. A large portion of faults in many practical circuits are random pattern resistant. which causes low fault coverages for a reasonable test length. The performance of the random test generation is affected by the random seed. In addition, it may be difficult to find compact test vectors by random test generation.

Most deterministic methods for sequential circuit test generation are based on fault-oriented three-phase test generation. For each fault under test. first. a combinational excitation vector is found to propagate the fault to the primary outputs or the next state lines. Second, the combinational excitation states are backward justified. Last, if the combinational excitation vector propagates the fault to the next state lines, state differentiation is needed to forward propagate the effect of fault on the next state lines to the primary outputs

In order to perform state justification and state differentiation, cube intersections are used on the complete or partial ON/OFF sets of the primary outputs and next state lines. Usually, the extraction of cover sets is considered as a pre-processing step to the three-phase test generation method. So strictly speaking, the conventional three-phase test generation method have one pre-processing step and three phases. i.e., cover extraction, combinational excitation vector generation. state justification, and state differentiation.

4.2 The Model of Global Test Generation

The extraction of cover sets includes the ON/OFF sets of the primary outputs and the next state lines. Each vector in the ON/OFF sets produces the corresponding primary output or the next state line to be logic value 1 (ON) or 0 (OFF). We have developed a backward assignment method to perform cover extraction. In the backward assignment method, each output (the primary output or the next state line in the sequential circuit) is set to logic value D and other outputs are set to logic value X. The logic value D is justified to the inputs of the circuit. The combination of logic values in the inputs is the D sets of the corresponding output, as shown in Figure 4.1. When the logic value D is set to 0, the OFF set is obtained. When the logic value D is set to 1, the ON set is obtained.

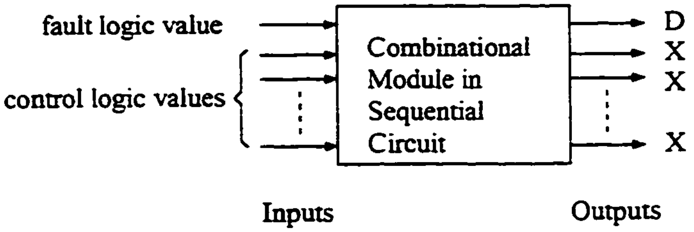


Figure 4.1. One output is set to logic value D and the combination of logic values on the inputs is the D set of the output.

From Figure 4.1, it is noticed that, in one group of assignment, one and only one input is assigned to the fault logic value. The other inputs are assigned to the control logic values. This property is determined by the backward assignment rules (we will discuss the rules in Chapter 5). Therefore, if the fault logic value is considered as a fault on the input, each vector in the cover sets is equivalent to a combinational excitation vector. For each vector in the cover sets of the primary outputs, state justification is needed to justify if the vector is reachable from the reset state. For each vector in the cover sets of the next state lines, state justification is first used to justify if the vector is reachable from the reset state. If it is reachable, state differentiation is needed to forward propagate the effect of the fault on the next state lines to the primary outputs.

Based on the backward assignment method, the extraction of cover sets and the combinational excitation vectors can be combined into one phase. The conventional three-phase test generation method (the combinational test generation, state justification, and state differentiation) and the pre-processing step (cover extraction) become the real three-phase test generation for sequential circuits, i.e., the extraction of cover sets/combinational excitation vector generation, state justification, and state differentiation.

The proposed real three-phase test generation method for sequential circuits is a global search method because each time it considers the problem of generating a test sequence for more than one fault simultaneously. During the test generation process, all faults in the circuit are considered simultaneously. The concept of considering more than one fault makes the search space universal and uses the common subsearch space for different faults.

Consider the circuit shown in Figure 4.2. There is a sensitive path between node a and node c. All faults along the path can be considered simultaneously. Also, for a fault, there may exist multi-sensitive paths. The proposed global search method uses a backward assignment method which traces different sensitive paths for the faults at the primary outputs and next state lines by justifying the faults to the primary inputs and present state lines. Therefore, the fault patterns at the primary inputs and present state lines are generated.

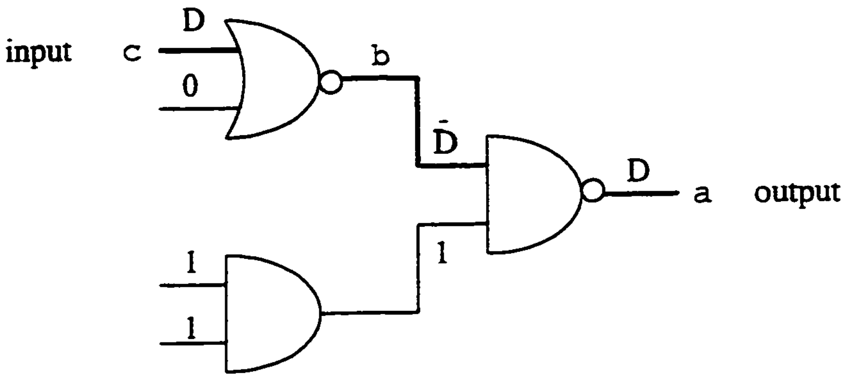


Figure 4.2. One sensitive path between one input and one output.

4.3 The Global Test Generation System for Sequential Circuits

Summarizing the above ideas. a novel global test generation system. GLOBALTEST. for sequential circuits is presented in this section. The GLOBALTEST system consists of two main algorithms: the fault-independent test generation algorithm and the fault-oriented test generation algorithm. as shown in Figure 4.3.

In fault-independent test generation. a novel backward assignment method is proposed to perform cover extraction and combinational excitation vector generation. Thus. the extraction of cover sets and combinational excitation vectors can be combined into one phase. Partitioning circuits is developed to increase the efficiency of extraction. State justification is used to justify the excitation states. If the combinational excitation vector propagates the fault to the next state lines. state differentiation is used to continuously propagate the effect of the faults to the primary outputs. To enhance the efficiency of state differentiation in the conventional three-phase ATPG. a backward deterministic method for state differentiation is proposed and the order of choosing next state lines in state differentiation is presented. The proposed fault-independent test generation algorithm for sequential circuits can de-

```
Input   : The sequential circuit and a list of faults.
Output : Test sequences for the detected faults.

procedure GLOBALTEST( )
begin
   /* the fault-independent test generation for sequential circuits */
   fault_independent_test_generation();

   /* the fault-oriented test generation procedure */
   fault_oriented_test_generation():

   output test sequences for the detected faults:
end;
```

Figure 4.3. GLOBALTEST: The global test generation system for sequential circuits.

tect a large percentage of faults. The disadvantage of this method is that it is difficult to prove redundant faults. It is therefore suitable to be used as a front end to the deterministic test generation method. Most of the existing ATPG systems use random vector test generation as the first step in test generation. Compared to random test generation. the proposed fault-independent test generation method takes a little more time but can cover more faults. Also. the random seed has no effect on the method. Random test generation is not needed in our system.

Following the fault-independent test generation algorithm. a deterministic and fault-oriented test generation algorithm is used to detect the remaining faults. In this dissertation, a revised three-phase test generation method is used. It considers one fault at a time and can determine redundant faults. The Boolean satisfiability and implication graph methods are extended to the test generation for sequential circuits. For each fault, at first, the combinational excitation vectors are generated. Then state justification is used to justify the excitation state. If the effect of the fault is propagated to the next state line. state differentiation is needed to continuously propagate the effect of the fault on the next state lines to the primary output.

The state differentiation method in the conventional three-phase ATPG method lacks efficiency in dealing with the unspecified inputs in the excitation vector and the justification sequence. STEED [18] has to try all possible assignments to the unspecified inputs before it can be concluded that a test for the fault under consideration does not exist. We have developed a backward deterministic algorithm for the state differentiation. Cubes, rather than minterm states, are used in the differentiation process. Our state differentiation algorithm searches backward to specify the cubes into the real excitation states. This has significantly reduced the differentiation time. To make state differentiation cover more faults and find shorter test sequences. our state differentiation algorithm tries the primary output first. If this fails, the algorithm tries the next state lines whose corresponding present state lines are in the input cones of the primary outputs. Finally, if this fails again, the algorithm tries the remaining next state lines.

An important goal of the ATPG system is to find a compact test set. The methods of test compaction can be classified as static and dynamic. In this dissertation, the test compaction problem is formulated as the set covering problem by the pseudo-sequence concept. An efficient set covering algorithm is proposed to compact test vectors. It attempts to select the necessary test vectors for the faults detected and eliminate other redundant test vectors. Then a local multi-variable search algorithm for test compaction is developed to further compact test vectors. The original fault coverage is not compromised.

In the following, we will discuss these two algorithms in detail.

4.4 The Fault-Independent Test Generation Algorithm

The fault-independent test generation algorithm for sequential circuits (*fault_independent_test_generation*()) consists of three procedures:

- extraction of the cover sets and the combinational excitation vectors: All flip-flops in the sequential circuit are disabled, and the sequential circuit becomes a pseudo-combinational circuit. A backward assignment method is presented to extract the ON/OFF sets of primary outputs and next state lines. The combinational excitation vectors can be generated at the same time. A method of partitioning circuits is developed to increase the efficiency.

- state justification: A justification sequence is found to take the circuit from the reset state into the excitation state.

- state differentiation: A differentiation sequence is found to propagate the effect of the fault on the next state lines to the primary outputs.

The fault-independent test generation algorithm for sequential circuits (*fault_independent_test_generation*()) is shown in Figure 4.4. At first. procedure *extract_sets_and_excitation_vectors*() is used to extract the ON/OFF sets of the primary outputs and next state lines. It can also generate the combinational excitation vectors at the same time. Procedure *state_justification*() is used to justify each excitation vector. If state justification succeeds, it is necessary to check which set the excitation vector comes from. If the excitation vector is from the ON/OFF sets of the primary outputs. a test sequence is found. If the excitation vector is from the ON/OFF sets of the next state lines. state differentiation is needed to continuously propagate the effect of the fault on the next state lines to the primary outputs. If state differentiation succeeds. a test sequence is found. When a test sequence is found. it is fault simulated by all faults under test.

In the following. we will briefly describe the extraction of the ON/OFF sets and combinational excitation vectors. state justification. state differentiation. and fault simulation.

4.4.1 Cover and Excitation Vector Extraction

The objective of cover extraction is to extract the ON/OFF sets of the primary outputs and next state lines. A backward assignment method is proposed to extract the ON/OFF sets and the combinational excitation vectors at the same time.

At first. all flip-flops in a sequential circuit are disabled. The sequential circuit becomes a pseudo-combinational circuit. The primary outputs and next state lines are

```
Input   : The sequential circuit and a list of faults.
Output : Test sequences for the detected faults.

procedure fault_independent_test_generation( )
begin
  /* phase 1: extracting the ON/OFF sets and combinational excitation
     vectors by partitioning circuits */
  excitation_vectors := extract_sets_and_excitation_vectors( ):
  for each excitational vector excitation_vector in excitation_vectors do
  begin
     /* phase 2: state justification */
     justification_sequence := state_justification(excitation_vector):
     if justification_sequence is found do
     begin
        /* judging if state differentiation is needed since the fault
           should be propagated to primary outputs */
        if the vector is from the ON/OFF sets of some next state lines do
        begin
           /* phase 3: state differentiation */
           differentiation_sequence := state_differentiation(excitation_vector):
           if differentiation_sequence is not found
              continue;
        end;
        use the test sequence to fault simulate all faults under test:
        if all faults are detected then break:
     end:
  end:
end:
```

Figure 4.4. The fault-independent test generation algorithm for sequential circuits.

considered as outputs of the pseudo-combinational circuit. The primary inputs and present state lines are considered as the inputs of the pseudo-combinational circuit.

For each output of the pseudo-combinational circuit. the combinational circuit is represented with its corresponding input cone. The ON/OFF/D sets are extracted by assigning the output line of the input cone to the logic value 0. 1. or D and using a backward assignment method to implicitly enumerate the input combinations that can set the output line to 0. 1. or D. Finally, the combination of the assignments at the inputs of the cone is the OFF, ON, or D set of the output.

Due to the connectivity of the circuit. some nodes in the circuit may be assigned more than once. Also. according to the backward assignment rules (we will discuss the rules in Section 5.1), one assignment on an output may have several groups of assignments on the corresponding inputs. Therefore with the increase of the circuit's depth. the number of assignments for each node per level may increase dramatically. The CPU time and memory space required for extracting the ON/OFF sets may also grow exponentially. A simple solution to the problem is to set a limit for the maximum number of assignments at each node [62]. When the number of assignments at a node reaches the limit. we ignore new assignments and extra assignments are discarded. Limiting the maximum number of assignments per node can dramatically decrease the extraction time. However, the ON/OFF sets obtained may be incomplete after the limit is set.

An efficient partitioning circuit method is developed. The pseudo-combinational circuit is partitioned into several smaller circuits according to some important partitioning nodes. Instead of extracting the ON/OFF sets of the original circuit. the ON/OFF sets of these smaller partitioning circuits are extracted. Since these partitioning circuits are smaller than the original circuit, it is more efficient to extract the

ON/OFF sets of these partitioning circuits. After extracting the ON/OFF sets of the partitioning circuits, it is necessary to combine the ON/OFF sets of the partitioning circuits into the ON/OFF sets of the original circuit. The detailed extraction method will be discussed in Chapter 5.

4.4.2 State Justification

Since the sequential circuit discussed here is assumed to have a reset state. all valid states begin from this reset state. State justification is used to justify if the combinational excitation state is reachable from the reset state. If the excitation state implicates the reset state, the fault can be excited by the reset state. Otherwise. the excitation state should be justified by state justification.

The iterative array model in Figure 4.5 is used to illustrate state justification. The excitation input PI^E and excitation state PS^E excite the effect of a fault under test to PO^E or NS^E. The goal of state justification is to find a primary input sequence. PI^{J1}. PI^{J2}. $\cdots$. PI^{Jk}. which places the sequential circuit into the excitation state PS^E from the reset state. If PS^{J1} is the reset state, the justification sequence PI^{J1}. PI^{J2}, $\cdots$, PI^{Jk} is found. The justification path is the set of states traversed during state justification, PS^{J1}, PS^{J2}, $\cdots$, PS^{Jk}.

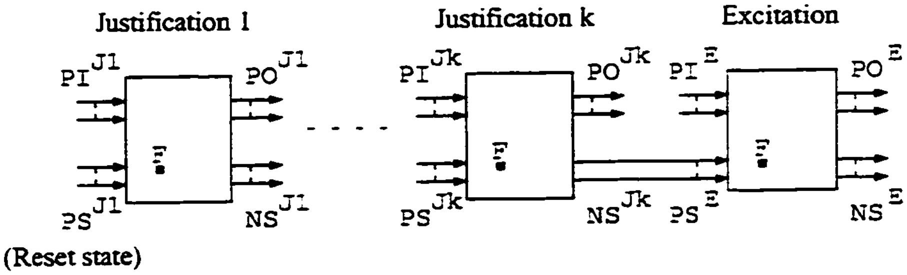


Figure 4.5. General iterative array model for state justification.

State justification can also be illustrated by the state transition graph (STG) shown in Figure 4.6. where PS^E is the excitation state. We need to find a justification path from the reset state to the excitation state PS^E.

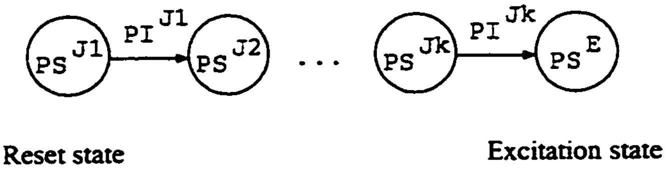


Figure 4.6. State transition graph for state justification.

State justification can be performed by two methods: forward and backward. depending on whether the search is conducted from the reset state to the excitation state or vice versa. In this dissertation, a backward state justification is adopted. All fan-in states of the excitation state are obtained by cube intersection on the corresponding ON/OFF sets of the next state lines. If the reset state implicates the fan-in states. a single vector justification sequence is found. Otherwise. the process is repeated for the fan-in states being currently justified to try to find multi-vector justification sequence.

Cubes. instead of minterm states. are used to represent the excitation state and all fan-in states in state justification. Using cubes is helpful to find shorter justification sequences. Thus the justification time is reduced and the quality of the test pattern generator increases.

4.4.3 State Differentiation

If the combinational excitation vector propagates the fault to the primary outputs and the excitation state is justified, a test sequence for the fault is successfully gen-

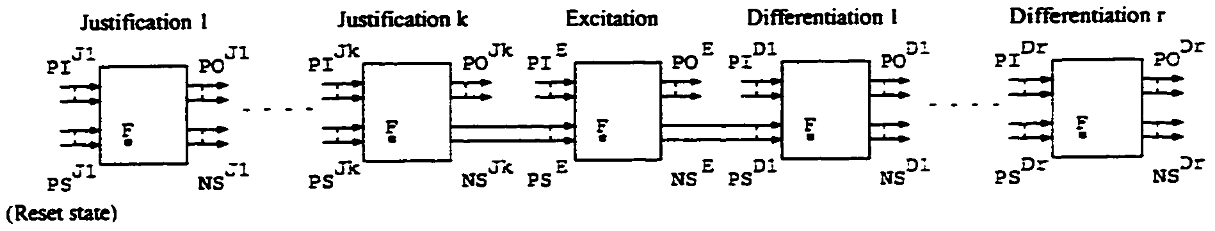


Figure 4.7. Iterative array model for state differentiation.

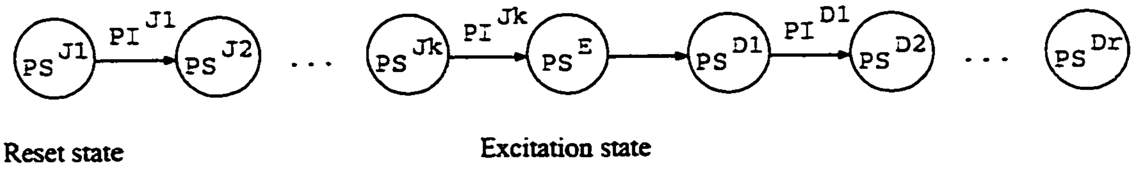


Figure 4.8. State transition graph for state justification and state differentiation.

erated. However. if the combinational excitation vector propagates the fault to the next state lines. state differentiation is required to continuously propagate the effect of the fault on the next state lines to the primary outputs.

State differentiation is illustrated by the iterative array model in Figure 4.7. The excitation input PI^E and excitation state PS^E excite the effect of a fault under test to the next state lines NS^E. and a justification path from the reset state PS^{J1} to PS^{Jk} is found. The goal of state differentiation is to find a primary input sequence. PI^{D1}. PI^{D2}. $\cdots$. PI^{Dr}. which propagates the effect of the fault on the next state lines of the excitation state clock cycle to the primary outputs of the rth differentiation clock cycle. The differentiation sequence is the primary input sequence. $PI^{D1}. PI^{D2}. \cdots. PI^{Dr}$. and the differentiation path is the set of states traversed during state differentiation. $PS^{D1}. PS^{D2}, \cdots, PS^{Dr}$. The test sequence is obtained by concatenating the justification sequence, the combinational excitation vector. and the differentiation sequence. The state transition graph can also be used to illustrate state justification and state differentiation. as shown in Figure 4.8.

In the procedure *state_differentiation*() shown in Figure 4.4. in order to reduce the time of state differentiation. random state differentiation is used first. If random state differentiation fails. a deterministic state differentiation algorithm is used.

At first. with the combinational excitation vector as the inputs to the primary inputs and present state lines. the fault-free and faulty states (S_1^T, S_1^F) on the next state lines are created by fault simulation. At the next clock cycle. the fault-free and faulty states (S_1^T, S_1^F) on the next state lines are propagated to the present state lines. By employing cube intersection on the ON and OFF sets of each primary output. a primary input vector is attempted to find which produces a different output on the corresponding primary output. beginning from the fault-free and faulty states on the present state lines separately. Such a primary input vector constitutes a single-vector differentiation sequence.

If a single-vector differentiation sequence cannot be found. all the fan-out states of the fault-free and faulty states are found via repeated cube intersection. This is performed by finding a primary input vector that produces a different output on at least one next state line for the fault-free and faulty states with the ON/OFF sets of each next state line. If the primary input vector is found. a new pair of fault-free and faulty states (S_2^T, S_2^F) are obtained. For the new fault-free and faulty states. a single-vector differentiation sequence is sought again. If found. a two-vector differentiation sequence is constructed. Otherwise. a pair of states fanning out from some fan-out state pair are picked and differentiation between this pair is attempted. The process continues until a differentiation sequence is found or there does not exist any differentiation sequence for S_1^T and S_1^F. Once the differentiation sequence is found. the entire test sequence is fault simulated to check if the fault can be detected.

In the general case, similar to state justification. state differentiation is attempted between disjoint groups of states (cube states) rather than a minterm state pair. This means that some bits in the true and faulty states are unknown. The existence of a differentiation sequence between two groups of states means that if *any* state S^T from the fault-free group is chosen, along with a *corresponding* state S^F from the faulty group, then the differentiation sequence will be able to differentiate between the states S^T and S^F. Since this is a strong requirement. it is often impossible to find a differentiation sequence between the state groups [18]. However. it does not mean that a test for the fault does not exist. In order to find a test. usually it is necessary to set some unspecified bits in the primary inputs or the present states of the justification sequence and excitation vector to either 0 or 1. A simple method can be applied where the excitation state is separated into a group of minterm states. and state justification and differentiation are performed on the minterm states. The disadvantage of this method is its long running time.

We propose an efficient backward deterministic method for state differentiation to solve this problem. After the combinational excitation state is found to propagate the fault to the next state lines with as many don't care entries as possible and is justified successfully, the backward deterministic method for state differentiation is used. When we search forward to perform state differentiation, if some unspecified bits in the present states and the primary inputs of the whole sequence are needed to be set to either 0 or 1. the backward deterministic method is used to determine the logic values of these unspecified bits and justify the new specific states by cube intersection on the ON and OFF sets of the primary outputs or the next state lines. If justification succeeds, the fault can then be propagated to the primary outputs or the next state lines. If this setting causes conflict in the unspecified bits between the fault-free and faulty states, the differentiation sequence does not exist. When the unspecified bits are on the primary inputs. they are just set to the required values. When the unspecified bits are on the present state lines. we check if the present state is justified from the next state of the last clock cycle. If the justification step needs to set some unspecified bits in the present state lines of the last clock cycle to specific logic value 1 or 0. the same process is repeated on the last clock cycle.

When we try to propagate the fault to the next state lines in state differentiation. the next state line whose corresponding present state line is in the input cones of primary outputs is chosen first. If this fails. the remaining next state lines are chosen. Such an order of choosing the next state lines is helpful to find a shorter test sequence. The method will be discussed in detail in Section 6.2.

4.4.4 Fault Simulation

When a potential test sequence for a fault is found. fault simulation is needed to check if the sequence detects the fault and other faults. In sequential circuits. the fault appears in every clock cycle. Hence. the single fault model becomes a multi-fault model. The test sequence is fault simulated by applying it to the fault-free circuit and the faulty circuit. If two groups of circuit outputs (as determined by simulation) differ. the fault is detected by the sequence. The quality of the test sequence can be estimated by

$$fault\ coverage = \text{(no. of faults detected) / (total no. of faults simulated)}.$$

Test efficiency is the percentage of detected and provably redundant faults over total number of faults. Fault simulation is an important step in an ATPG system for both combinational and sequential circuits. Much research has been done in this area and many efficient fault simulation algorithms have been developed. In general.

there are three kinds of fault simulation methods: parallel fault simulation. deductive fault simulation. and concurrent simulation.

In this dissertation, fault simulation has two main purposes:

- confirms detection of a fault for which an automatic test pattern generator (ATPG) claims that a successful test is found.
- computes fault coverage for a given test sequence.

For the first purpose. a simple event-driven fault simulation is used. The algorithm of fault simulation is shown in Figure 4.9. At first. the initial states of the fault-free circuit and the faulty circuit are set to the reset state. For each clock cycle. the corresponding test vector is used to deduct the logic values of the fault-free circuit and the faulty circuit. If the logic values at the primary outputs of the faulty circuit are different from those of the fault-free circuit. the fault is detected by the test sequence.

For the second purpose. we use HOPE as the fault simulator. HOPE is an efficient sequential circuit parallel fault simulator which simulates 32 faults at a time [33].

4.5 The Fault-Oriented Test Generation Algorithm

The test vectors generated by the fault-independent test generation algorithm can detect most of the faults in a sequential circuit. For the remaining undetected faults. we extend the Boolean satisfiability and the implication graph methods to the three-phase test generation algorithm for sequential circuits. The three-phase test generation algorithm is used as the fault-oriented test generation algorithm. The algorithm consists of three parts: combinational excitation vector generation. state justification. and state differentiation, as shown in Figure 4.10. The flow chart of the algorithm is shown in Figure 4.11.

Input : A sequence of test vectors and a fault under test.
Output : The fault is detected by the sequence or not.

```
procedure fault_simulator(a fault under test)
begin
   set the initial states of the fault-free and faulty circuits to the reset state:
   for each clock cycle of test vector do
   begin
      deduct signals values at the fault-free circuit:
      deduct signals values at the faulty circuit:
      for each primary output
         if the fault-free value is different from the faulty value
            return that the fault is detected by the test sequence:
   end:
   return that the fault can not be detected by the test sequence:
end:
```

Figure 4.9. The fault simulation algorithm.

```
Input  : The sequential circuit and a list of faults.
Output : Test sequences for the detected faults.

procedure fault_oriented_test_generation( )
begin
  for each fault under test do
  begin
    /* try to propagate the fault to primary outputs */
    if the fault site is a node in input cones of some primary outputs
    begin
      /* extracting the input cones of these primary outputs */
      cone := extract_primary_output_cone():
      /* extracting combinational excitation vectors */
      while (new excitation_vector := combin_test_generation(cone) is found)
      begin
        justification_sequence := state_justification(excitation_vector):
        if justification_sequence is found
          use the test sequence to fault simulate all faults under test:
          if fault simulation succeeds then break:
      end:
    end;
    if the test sequence is found then continue:
    /* try to propagate the fault to next state lines as it can't be
       propagated to primary outputs directly */
    if the fault site is a node in input cones of some next state lines
    begin
      cone := extract_next_state_cone();
      while (new excitation_vector := combin_test_generator(cone) is found)
      begin
        justification_sequence := state_justification(excitation_vector):
        if justification_sequence is found
          differentiation_sequence := state_differentiation(excitation_vector):
          if differentiation_sequence is found
            use the test sequence to fault simulate all faults under test:
            if fault simulation succeeds then break:
      end:
    end;
  end:
end;
```

Figure 4.10. The fault-oriented test generation algorithm for sequential circuit.

The fault-oriented test generation algorithm *fault_oriented_test_generation*( ) considers a single fault at a time. It consists of the following steps:

Step 1. The fault is attempted to be propagated to the primary outputs directly. If the fault site is a node in the input cones of some primary outputs. the corresponding input cones of these primary outputs are extracted, and go to step 2. Otherwise. go to step 5.

Step 2. A Boolean satisfiability and implication graph based test generation algorithm for pseudo-combinational circuits is used to find a (new) combinational excitation vector. If the combinational excitation vector has the present state part disjointed from the present state part of all the previously generated test vectors, go to step 3 to perform state justification. If such a new vector is not found. the fault cannot be propagated to the primary outputs directly, and go to step 5.

Step 3. State justification is performed to justify if the excitation state is reachable from the reset state. If the justification is not found. return to step 2. If found. go forward to step 4.

Step 4. Fault simulate the test sequence. If it detects the fault. return with the test sequence. Otherwise. go back to step 3.

Step 5. Since the fault can not be propagated to the primary outputs directly. the fault is attempted to be propagated to the next state lines first. If the fault site is a node in the input cones of some next state lines. the corresponding input cones of these next state lines are extracted, and go to step 6. Otherwise. the fault is redundant.

Step 6. The combinational test generation algorithm is used to find a (new) combinational excitation vector. If the combinational excitation vector has the present state part disjointed from the present state part of all the previously generated test vectors. go to step 7 to perform state justification and state differentiation. If such a new vector is not found. exit without test for the fault.

Step 7. State justification is used to justify if the excitation state is reachable from the reset state. If such a justification sequence is not found. return to step 6. If found. state differentiation is performed to propagate the effect of the fault on the next state lines to the primary outputs. If a differentiation sequence is found. go to step 8 to do fault simulation. Otherwise, go back to step 6 to find a new excitation vector.

Step 8. Fault simulate the test sequence. If it detects the fault. return with the test sequence. Otherwise, go back to step 7.

When a test sequence is found, the test sequence is used to simulate all faults in the fault list. Thus all the faults that can be detected by the test sequence are removed from the fault list.

4.5.1 Pseudo-Combinational Circuit Test Generation

Given a fault under test. the first step in the fault-oriented test generation algorithm is to generate combinational excitation vectors in the pseudo-combinational circuit. The primary output fault region for a fault under test is extracted first if the fault site is a node in the input cones of some primary outputs. If all combinational excitation vectors generated in the primary output fault region are not reachable from the reset state by state justification, the fault is needed to be propagated to the next state lines. In this case, the next state fault region for the fault should be extracted if the fault site is a node in the input cones of some next state lines. When the fault is combinational redundant, the effect of the fault cannot be propagated to either the primary outputs or the next state lines.

Figure 3.4 shows a pseudo-combinational circuit obtained from a general sequential circuit by disabling all flip-flops. The goal of test generation for a pseudo-combinational circuit is to find a combinational excitation vector (PI^E, PS^E) which excites the fault to PO^E or NS^E.

Our test generation algorithm for combinational circuits, which will be described in Chapter 5 in detail, is based on SAT model and implication graph methods [32, 9]. By incorporating necessary conditions for fault activation and path sensitization, an implication graph and SAT formula is extracted from the circuit. Any assignment which satisfies the implication graph and SAT formula constitutes a combinational excitation vector for the fault. In this dissertation, we will adopt a SAT solver in SIS developed by the University of California, Berkeley [59].

To make state justification easier, the combinational excitation vector is generated with as many don't care entries as possible, i.e., some bits remain unknown. If the excitation state is not justified, a new excitation vector is generated. To assure that the newly generated excitation vectors are not used previously, all new vectors should be disjointed from all previous states.

It is noted that a fault often affects a portion of a circuit. To reduce search space in combinational test generation, the corresponding fault region is considered instead of the whole circuit. Since the related fault region is usually smaller than the original circuit, the excitation vector can be generated efficiently.

4.5.2 Determination of Redundant Faults

The determination of redundant faults is a difficult problem in test generation for sequential circuits. Low fault coverage on certain sequential circuits does not mean that the test generation system is not suitable for the circuits if we can prove that the detected faults are close to the maximum possible number of detectable faults. In general. the determination of redundant faults may need a large amount of CPU time. since the search space should be exhausted before the fault is said to be redundant.

There are two kinds of redundant faults: combinational redundant faults and sequential redundant faults. Faults in sequential circuits are classied into three sets: detectable. partially detectable. and operationally redundant. [45] The last two classes are dependent on the operation mode of the circuit.

Combinational redundant faults are relatively easier to determine compared to the sequential redundant faults. If a fault can not be propagated to the primary outputs or the next state lines, no matter what inputs vectors are excited. beginning from any states. the fault is said to be a combinational redundant fault. Therefore. if none of the combinational excitation vectors can be found for a fault in the corresponding pseudo-combinational circuit. the fault is said to be a combinational redundant fault.

The sequentially redundant faults can be determined using theorem 1 in [35]. This theorem states that if all excitation states are not reachable from the reset state in the fault-free circuit. the fault is sequentially redundant. To determine a sequential redundant fault, all combinational excitation vectors should be generated. If none of the excitation states are justifiable. the fault is sequentially redundant.

4.6 Summary

Observations that initiated the research work in test generation for sequential circuits have been presented in this chapter. A novel global test generation system. GLOBALTEST. for sequential circuits is presented. At first. a fault-independent test generation algorithm for sequential circuits is used and detects most detectable faults. It can be used to replace random test generation as a front end test generator. Then a fault-oriented test generation algorithm for sequential circuits is executed to deal with the remaining faults and determine the redundant faults.

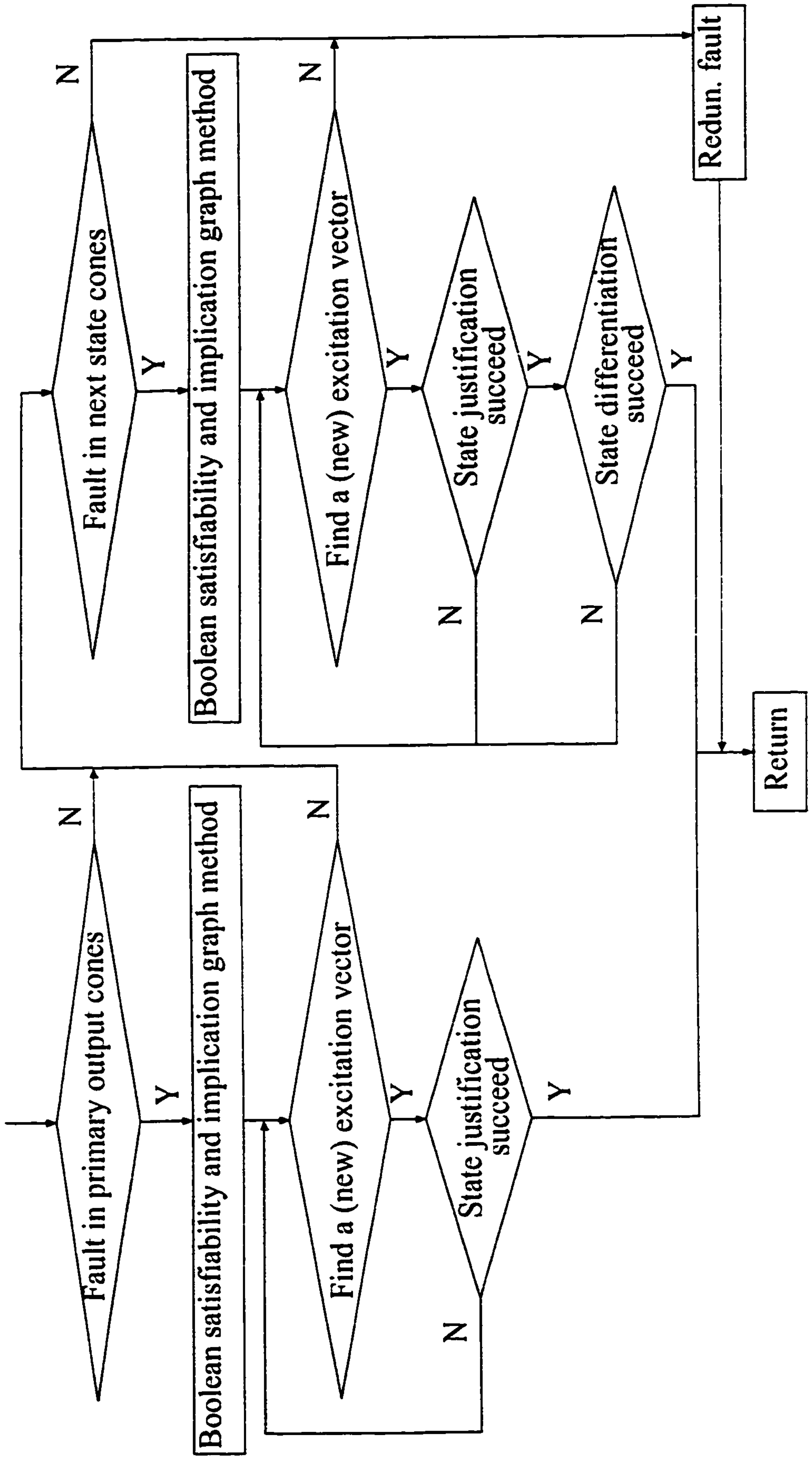

Fault in primary output cones
Y
N
Boolean satisfiability and implication graph method
Find a (new) excitation vector
Y
N
State justification succeed
Y
N
Fault in next state cones
Y
N
Boolean satisfiability and implication graph method
Find a (new) excitation vector
Y
N
State justification succeed
Y
N
State differentiation succeed
Y
N
Redun. fault
Return

Figure 4.11. The flow chart of the fault-oriented test generation algorithm for sequential circuits.

CHAPTER 5

EXTRACTION OF COVER SETS AND COMBINATIONAL EXCITATION VECTORS

In state justification and state differentiation. the ON/OFF sets of primary outputs and next state lines are used to perform cube intersections. The process of extracting the ON/OFF sets of the primary outputs and next state lines is called cover extraction. To generate a test sequence for a fault in sequential circuits, combinational excitation vectors are generated which propagate the fault to the primary outputs or the next state lines. In the conventional three-phase ATPG system. cover extraction and combinational excitation vector generation are performed separately. Cover extraction is the pre-processing step and combinational excitation vector generation is the first phase.

In this chapter, an efficient backward assignment method for cover extraction and excitation vector generation is presented. By the backward assignment method. cover extraction and excitation vector generation can be combined into one phase. A circuit partitioning method is developed to make the extraction efficient.

At first. the backward assignment rules (referred to as B-rules) are introduced. The consistency and algorithm constraints are presented. Then the circuit partitioning method is discussed. The new backward assignment procedure is presented. Finally. a combinational test generation algorithm based on Boolean satisfiability and implication graph is given. This algorithm is used for the fault-oriented test generation algorithm for sequential circuits.

5.1 Backward Assignment Rules (B-rules)

At first. all flip-flops in a sequential circuit are disabled. The sequential circuit becomes a pseudo-combinational circuit. For each output of the pseudo-combinational circuit. the corresponding circuit is represented as a separate input cone of each output. The ON/OFF sets of each output are extracted by assigning the corresponding output of the cone to logic value 1 or 0 and using the backward assignment method to implicitly enumerate the input combinations that can set the output to 1 or 0. The combination of the assignments at the inputs of the cone is the ON or OFF set of the output.

The objective of the B-rules is to justify the assignment of a logic value at the output of a circuit to each node in the corresponding input cone during the backward assignment procedure. The B-rules for AND. NAND. OR. NOR. XOR. and XNOR gates are shown in Figures 5.1 - 5.6. respectively.

A nine-value logic (0. 1. X. D. $\overline{D}$. FD. $F\overline{D}$. TD. $T\overline{D}$) (refer to the definitions on page 26) is used to describe the circuit behavior. Consider the 3-input NAND gate shown in Figure 5.2. Suppose that the ON set is extracted. From the K-map. if any input of the NAND gate is logic value 0. the output of the NAND gate is logic value 1. Therefore. when the output is set to logic value 1. one of the inputs must be 0. Three groups of inputs for the ON set of the NAND gate are obtained: $(0.X.X)$. $(X.0.X)$. and $(X.X.0)$. Similarly. we can obtain the OFF set. D set. $\overline{D}$ set. FD set. $F\overline{D}$ set. TD set. and $T\overline{D}$ set for the NAND gate. If the output of a gate is X (don't care). all inputs are don't care. In this case, we just skip the output and leave the inputs to keep the original values. Since the logic value 0 or 1 (OFF or ON set)

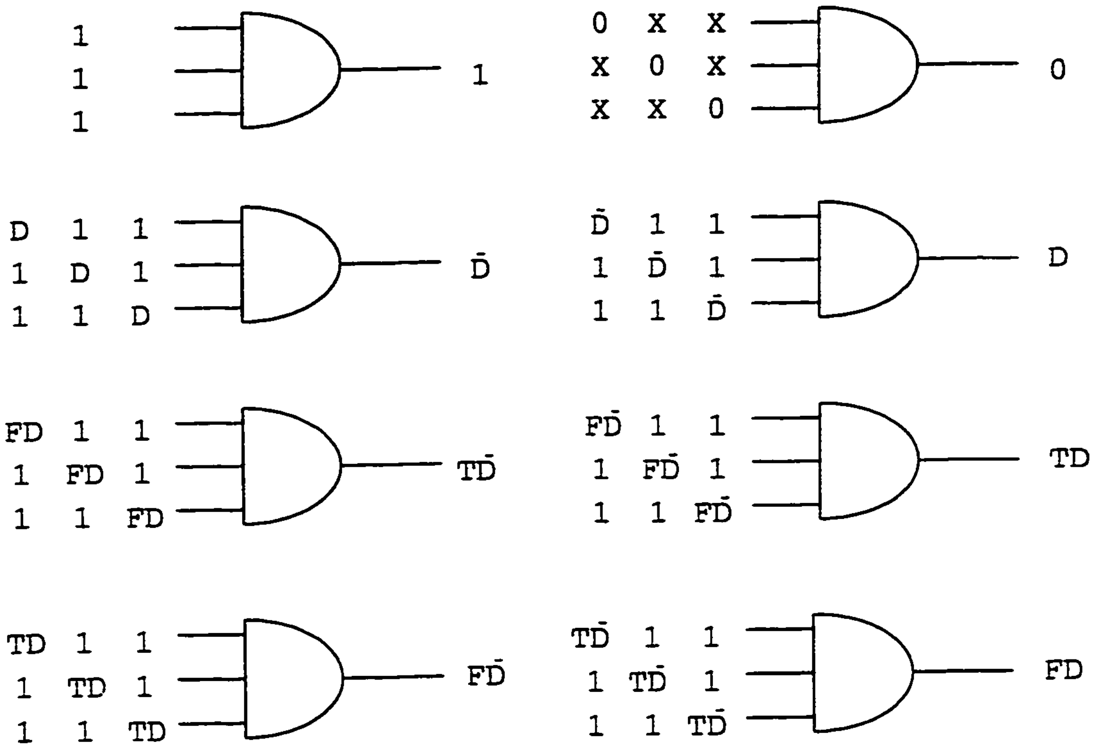


Figure 5.1. The backward assignment rules (B-rules) of AND gate.

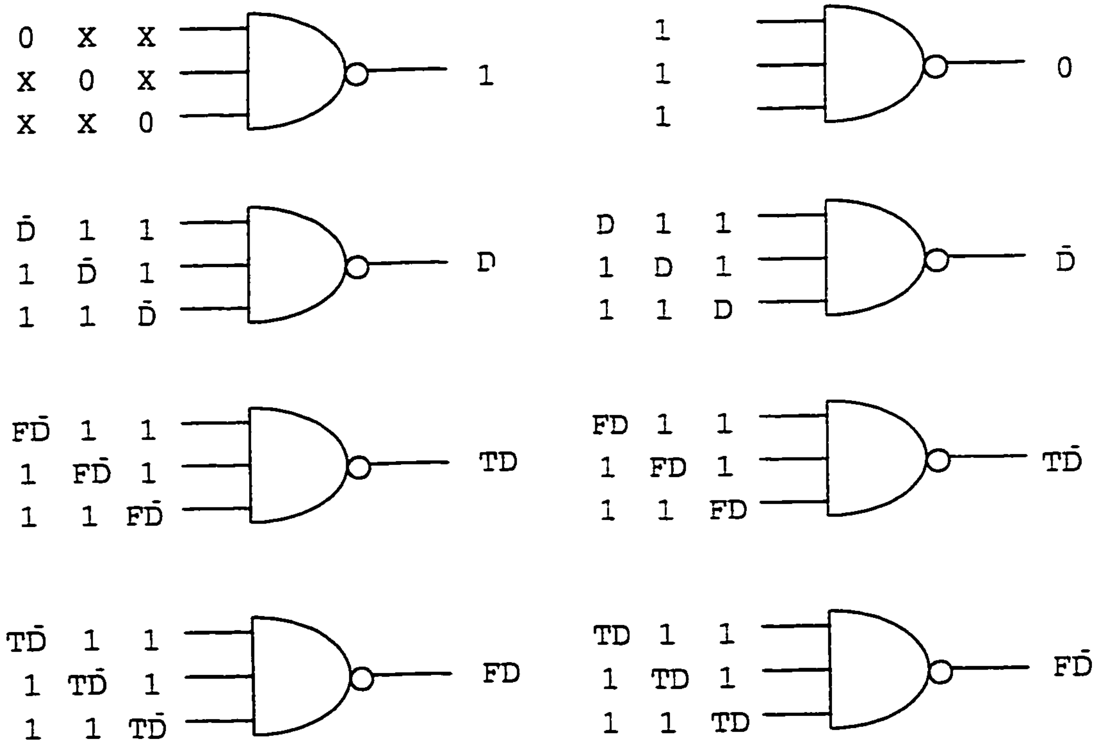


Figure 5.2. The backward assignment rules (B-rules) of NAND gate.

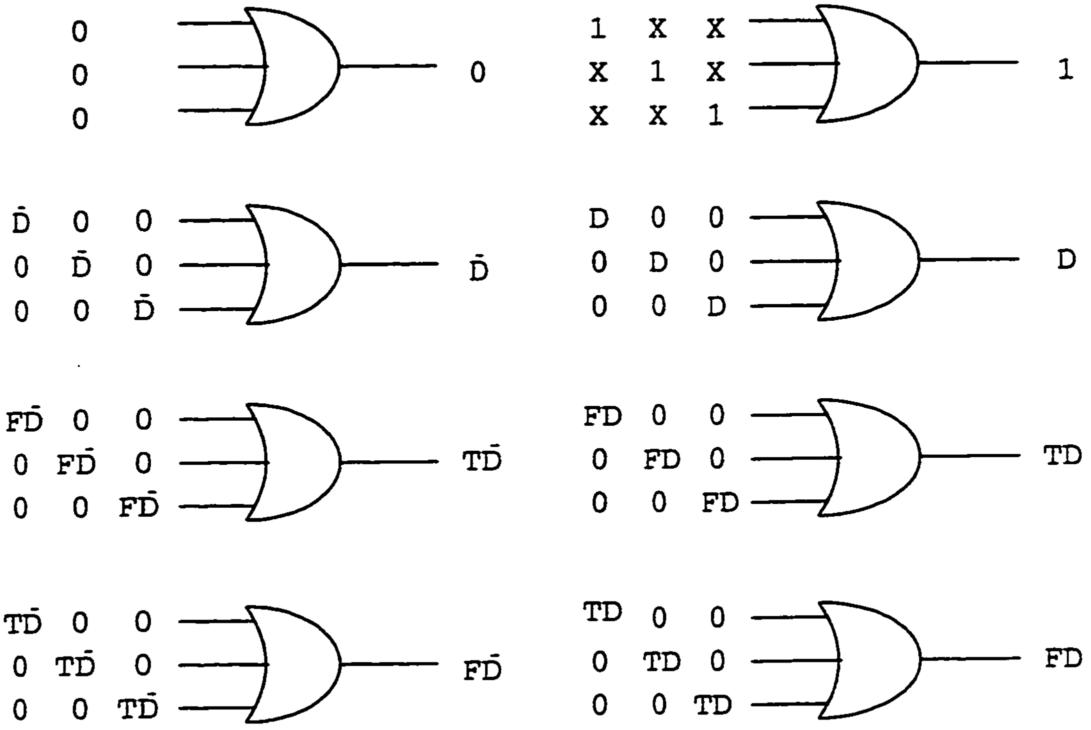


Figure 5.3. The backward assignment rules (B-rules) of OR gate.

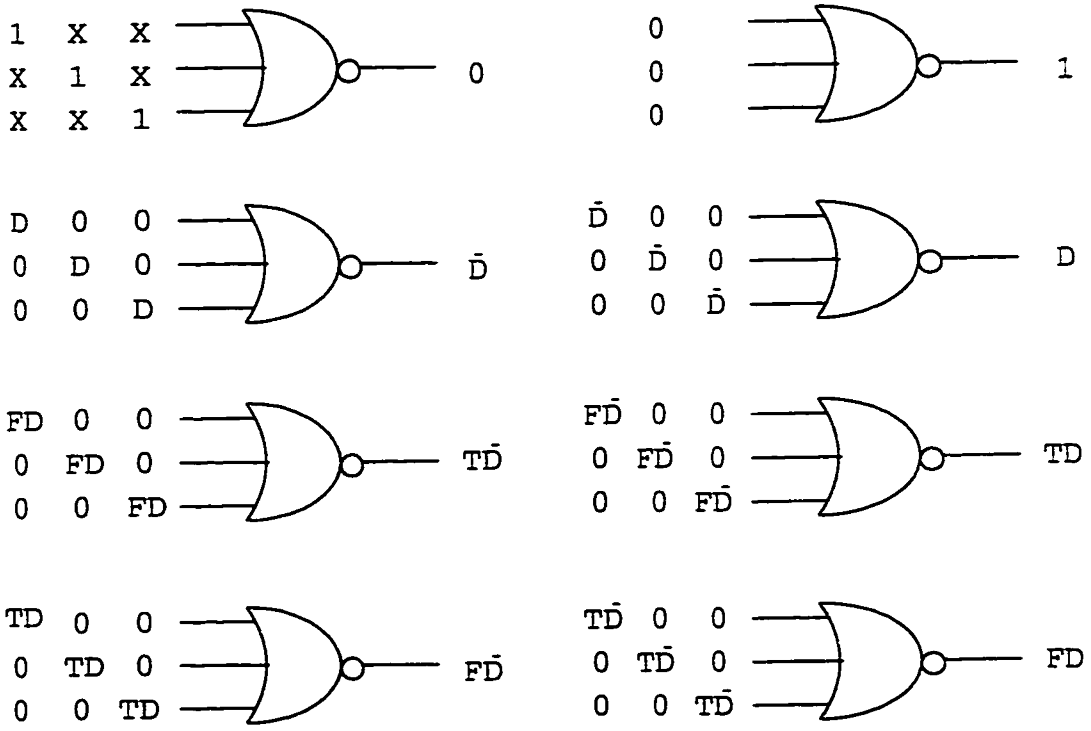


Figure 5.4. The backward assignment rules (B-rules) of NOR gate.

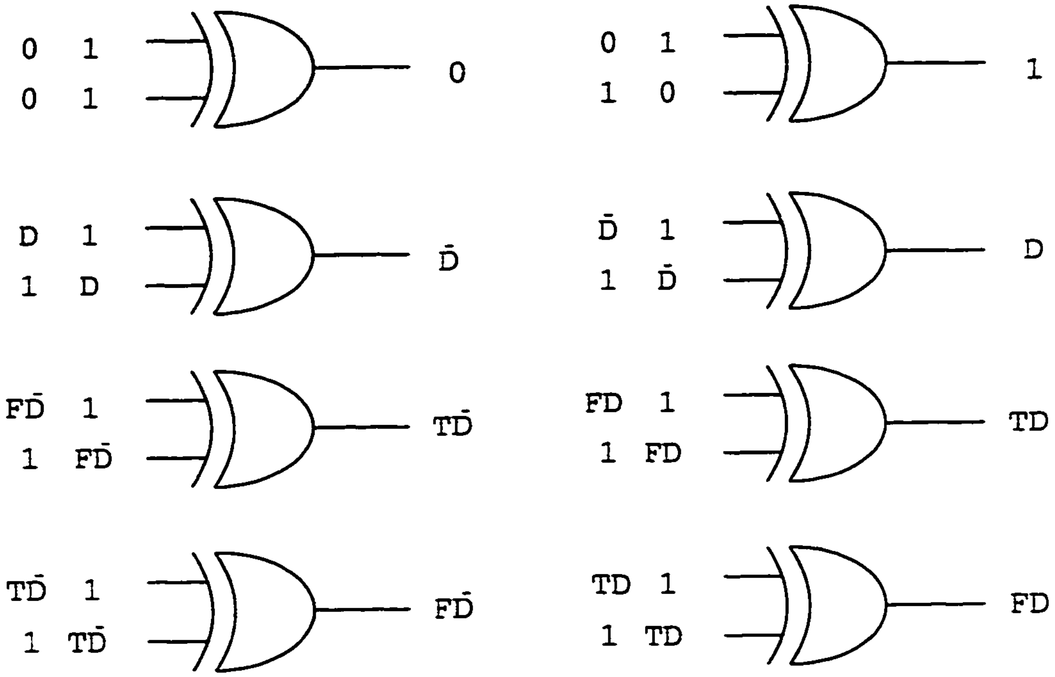


Figure 5.5. The backward assignment rules (B-rules) of XOR gate.

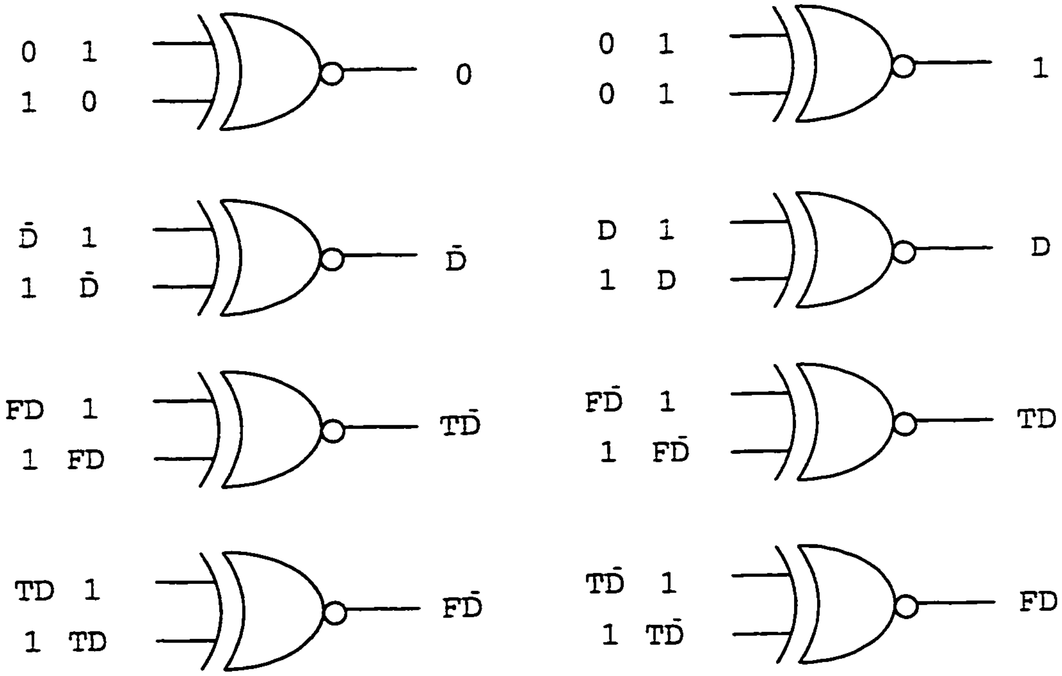


Figure 5.6. The backward assignment rules (B-rules) of XNOR gate.

is a control logic value. all inputs for the OFF/ON set are assigned the control logic values.

The situation is different when a fault logic value set is extracted. For example. consider the D set of the 3-input NAND gate in Figure 5.2. Three groups of inputs for the D set of the NAND gate are obtained: $(\overline{D}, 1, 1)$. $(1, \overline{D}, 1)$, and $(1, 1, \overline{D})$. It is shown that one and only one input in the D set is assigned to the fault logic value. Other inputs in the D set are assigned the control logic values. Therefore. it can be concluded that for the fault logic value set of any gate. there is one and only one input assigned to the fault logic value and the other inputs are assigned the control logic values. Actually. the conclusion can be extended to the whole circuit. From the conclusion. a lemma can be obtained:

Lemma 1 In each vector in the fault logic value sets. such as D and $\overline{D}$ sets. one and only one input has the fault logic value. and the other inputs are assigned the control logic values. In each vector of the control logic value sets. such as OFF and ON sets. all inputs are assigned the control logic values.

The proof of Lemma 1 is quite easy. since all B-rules of different gates are given in Figures 5.1 - 5.6 separately. 1) We want to extract the fault logic value sets. such as D set. Consider a combinational module from a sequential circuit. as shown in Figure 5.7. One output is set to the logic value D and other outputs are set to don't care values X. The backward assignment rules are used to justify the logic values at outputs to inputs. According to the backward assignment rules. since only one node at each group of assignment is set to the fault logic value. one and only one input of the circuit has the fault logic value. 2) We want to extract the control logic value sets. All outputs of the circuit are set to the control logic values. so according to the backward assignment rule, all inputs of the circuit are assigned control logic values.

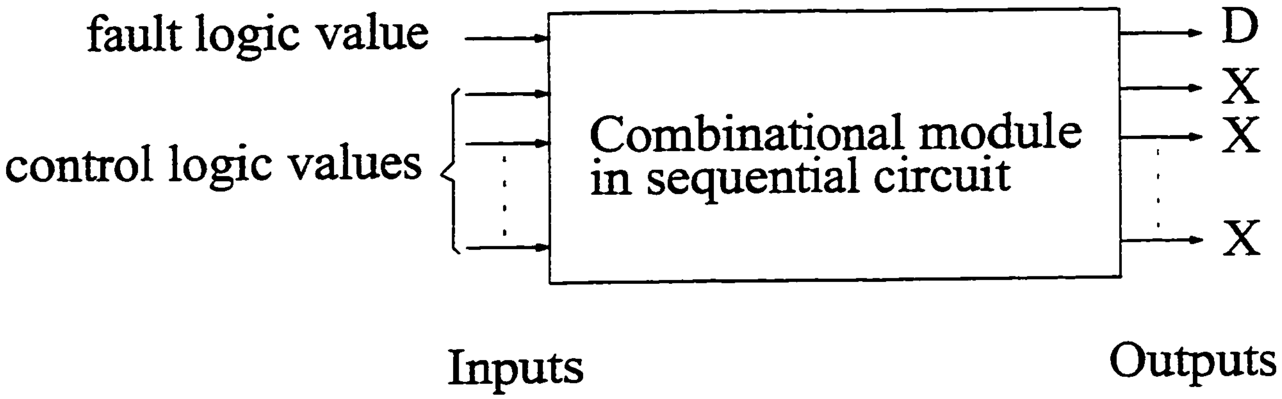


Figure 5.7. Extraction of D set of an output in a combinational circuit.

5.2 Consistency and Constraints

The proposed backward assignment method is to propagate logic values at the primary outputs and next state lines to the inputs of the circuit. For each gate in the circuit. the B-rules are used to justify logic values at the output to its inputs. Since some nodes may have multiple fanout nodes. these nodes may be assigned logic values more than once by their multiple fanout nodes. Therefore. it is necessary to check at each level of assignment for these nodes that have been assigned new values.

The consistency constraint is proposed to ensure that the assignments of logic values are correct. as shown in Figure 5.8. If these logic values are in conflict with each other – for example, one fanout requires the node to be logic value 0. and another fanout requires the node to be logic value 1 – the assignment is in conflict and should be discarded. If logic value v_1 assigned by one fanout implies logic value v_2 assigned by another fanout. the consistent logic value v_1 should be chosen.

When the B-rules are used to justify logic values at the outputs to the inputs of the circuit. the combination of the inputs may exceed one. For example. in the ON set of the 3-input NAND gate. three groups of inputs are obtained. In this case. every time. one group of assignments is used as the outputs of backward stage gates. Therefore. with the increase of the circuit's depth. the number of assignments for each node per level may increase dramatically. Also. the CPU time for extracting the ON/OFF sets may increase greatly. A simple method is to set a limit for the maximum number of the assignments at each node [62]. Limiting the maximum number of assignments per node can dramatically decrease the extraction time. However. the ON/OFF sets obtained may be incomplete.

In this dissertation, circuit partitioning is proposed to solve this problem. This

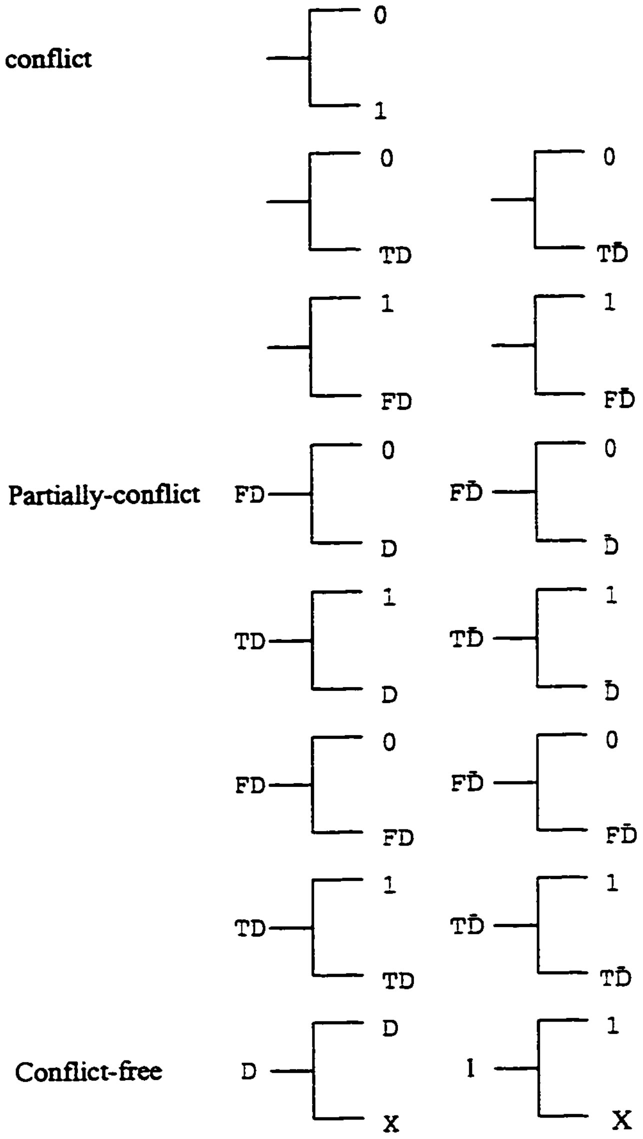


Figure 5.8. The consistency constraint of logic values.

method assures that the complete or near complete ON/OFF sets are obtained. Also. logic minimization is used to compress the assignments after the number of assignments reach a limit. It makes the storage of ON/OFF sets memory efficient. In the following section, a method of partitioning sequential circuits is introduced.

5.3 Partitioning Sequential Circuits

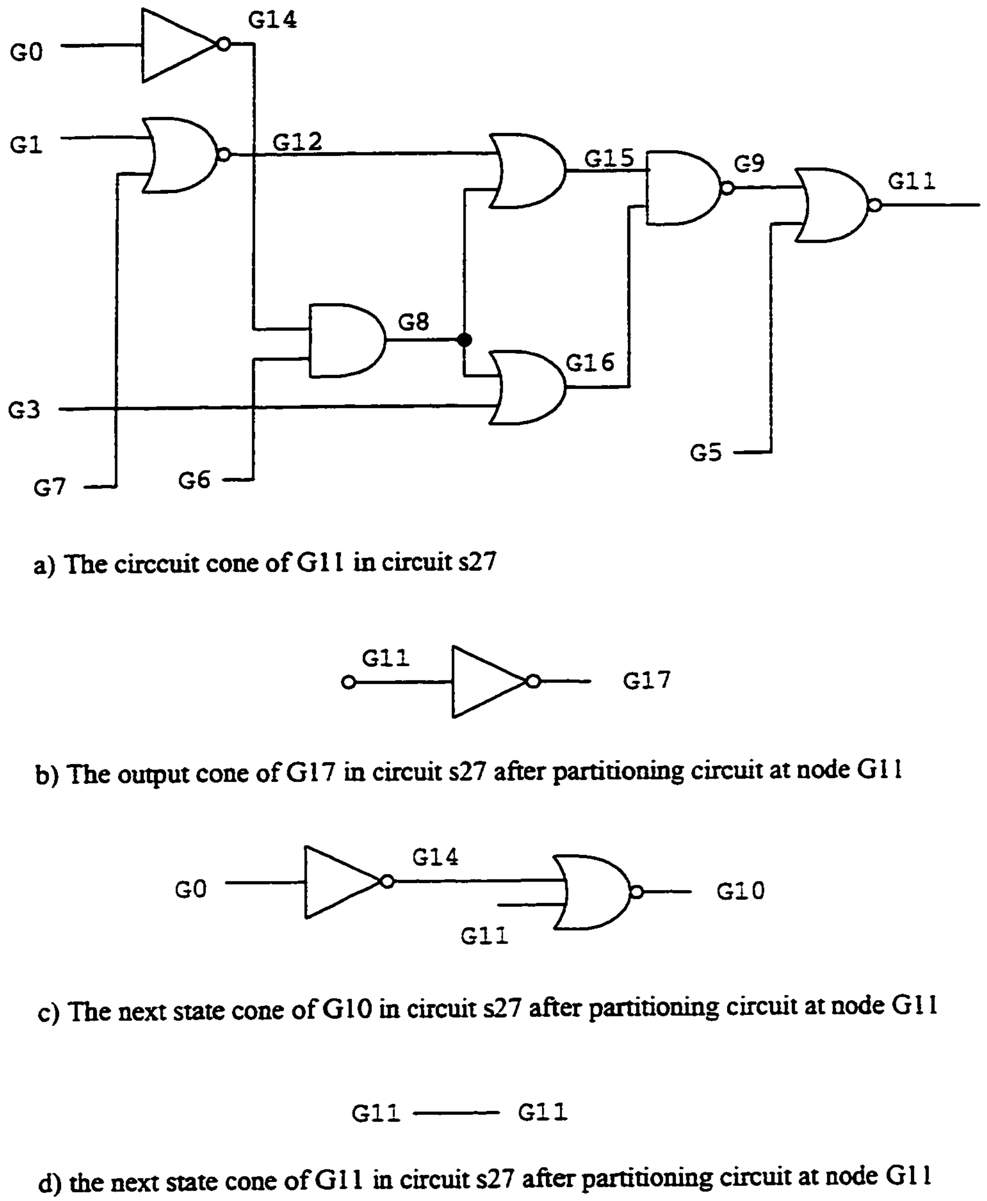
Usually. in order to get the cover sets of each output. we need to obtain their corresponding circuit cones and then extract the cover sets of each output from its circuit cone. For large circuits, the extraction process may be time-consuming. We developed an efficient method to extract the cover sets by partitioning the sequential circuit. The partitioning method is illustrated by a circuit s27 from ISCAS'89 benchmark circuits. as shown in Figure 3.5. There are one primary output *G17* and three next state lines *G10. G11*, and *G13*. They are considered as the outputs of the pseudo-combinational circuit after the flip-flops are disabled.

The corresponding input cones of the primary output and next state lines are shown in Figures 3.6. 3.7. 3.8, and 3.9. From these input cones, it is found that some parts of circuit are repeatedly extracted. For example, the input cone of *G11* appears in the input cone of the primary output *G17* and the input cones of the next state lines *G10* and *G11*. So the input cone of *G11* is extracted three times.

The pseudo-combinational circuit can be partitioned according to some important nodes in the circuit. These nodes are called partitioning nodes. Node *G11* has three fan-out lines: *G17*. *G10*. and *G6*. The cover sets of node *G11* should be extracted first and are represented as $C_{G11}(G0, G1. G3, G5. G6. G7)$. After that, the node *G11* can be considered as a virtual input node to other related outputs. The extraction of cover sets of the output nodes *G11*, *G10*, and *G17* is simple. The corresponding input cones for nodes *G11*, *G10*, and *G17*. as shown in Figure 5.9. are smaller than the previous ones. Previously, the input cone of the node *G10* contains 8 gates: *G10*. *G14*, *G11*, *G9*, *G15*, *G16*, *G8*, and *G12*. After the node *G11* is considered as an input node, the input cone of the node *G10* has only 2 gates. i.e., *G10* and *G14*. Also. the size of the input cone of the node *G17* is reduced from 8 gates to 1 gate. Therefore. the size of input cones is reduced after partitioning and the extraction efficiency is increased greatly.

To extract the ON/OFF sets of the primary outputs and next state lines. first the ON/OFF sets of the partitioning node *G11* are extracted from its input cone. Then the ON/OFF sets of the primary output *G17* and next state lines *G10*. *G11*. and *G13* are extracted from their corresponding input cones after the partitioning node *G11* is considered as the virtual input node to the input cones. After the extraction. the obtained cover sets need further processing since the partitioning node is not the real input node. The cover sets of node *G10* are represented as $C_{G10}(G0.G11)$. where node *G0* is the input node to the input cone and node *G11* is the virtual input node (partitioning node) to the input cone. Cube intersection is needed to extend the cover sets of the node *G11* into the cover sets $C_{G10}(G0.G11)$ of the node *G10*. If the node *G11* in the sets $C_{G10}(G0.G11)$ has logic value 0 (or 1). the OFF set (or ON set) of the node *G11* is picked. The cube intersection of the OFF set (or ON set) of the node *G11* and the sets $C_{G10}(G0.G11)$ gives the real cover sets of the node *G10*. Therefore. the real cover sets of the node *G10* are obtained as $C_{G10}(G0.G1.G3.G5.G6.G7)$. The nodes *G0*. *G1*, *G3*. *G5*. *G6*. and *G7* are the input nodes to the circuit.

It is important to choose adequate partitioning nodes in the circuit. A simple and efficient method is presented to determine partitioning nodes. The weight of each node is defined as the product of its size and its fan-out number. The size of a node



a) The circcuit cone of G11 in circuit s27

b) The output cone of G17 in circuit s27 after partitioning circuit at node G11

c) The next state cone of G10 in circuit s27 after partitioning circuit at node G11

d) the next state cone of G11 in circuit s27 after partitioning circuit at node G11

Figure 5.9. The input cones of the primary output and next state lines in circuit s27 after the node *G11* is considered as a virtual input node.

is defined as the number of nodes in the input cone of the node. It is found that if the size of a partitioning node is too large, it is still time-consuming to extract the cover sets of the partitioning node. Therefore, a limit is set on the size of each partitioning node. In our experiments, the limit is set to 70 to 100. When the size of a node exceeds the limit, its weight is set to zero. Each time, the node with the maximum weight is chosen as the partitioning node. After a partitioning node is chosen, at first the partitioning node is taken as a virtual output node and its cover set is extracted. Then, another partitioning node is chosen and all previously chosen partitioning nodes are taken as the virtual input nodes to its input cone.

5.4 Generation of Combinational Excitation Vectors from Cover Extraction

Each time, one primary output or next state line is set to logic value D. Other primary outputs and next state lines are set to don't care values. The B-rules are used to justify the logic values to the inputs of the circuit and the D set of the primary output or next state line is obtained. According to Lemma 1, one and only one input in each vector of the D set is assigned to the fault logic value, and the other inputs are assigned to the control logic values. If the fault logic value is considered as a fault in the input, each vector is equivalent to a combinational excitation vector.

For example, one vector in the D set of the next state line $G13$ is $(X, 1, \overline{D}, X, X, X, X)$ for nodes $G0$, $G1$, $G2$, $G3$, $G5$, $G6$, and $G7$ (see Figure 3.5). When D is set to 0, one vector in the OFF set of line $G13$ and one combinational excitation vector are generated, i.e., $(X, 1, 1, X, X, X, X)$. It is obvious that the combinational excitation vector can detect a fault s-a-0 at the input $G2$. When D is set to 1, one vector in the ON set of line $G13$ and one combinational excitation vector are generated, i.e., $(X, 1, 0, X, X, X, X)$. It can detect a fault s-a-1 at the input $G2$.

Therefore, from Lemma 1, the ON/OFF sets and the combinational test vectors can be extracted from the D set in one phase by the proposed backward assignment method.

5.5 The Algorithm of Cover Extraction and Combinational Excitation Vector Generation

Three methods can be used to extract the ON/OFF sets of the primary outputs and next state lines in a pseudo-combinational circuit.

- One output of the circuit is set to 1 or 0, and the backward assignment method is used to extract the ON or OFF set of the output separately.
- The ON set and OFF set for the same output are complementary, which means that the union of the ON set and OFF set for the same output should be the corresponding universal cube. So when the ON set is extracted, the OFF set can be easily obtained by disjointing the ON set from the corresponding universal cube. The limitation of this method is that the ON or OFF set must be complete. Otherwise, it may cause errors in the extracting the OFF or ON set by disjointment.
- One output of the circuit is set to the logic value D. The backward assignment method is used to extract the D set of the output. When D is equal to 1 or 0, the ON or OFF set is obtained. Also, the combinational excitation vectors can be generated from the D set.

Since the third method is time-efficient and also can generate combinational excitation vectors, the third method is used to extract the ON/OFF sets of the primary outputs and next state lines combined with the circuit partitioning method. A high level description of the cover extraction and combinational excitation vector generation algorithm by the backward assignment method is shown in Figure 5.10. The procedure *extract_sets_and_excitation_vectors*() first chooses partitioning nodes and partitions the circuit according to the partitioning nodes. It selects a partitioning node and assigns it logic value D. The backward assignment procedure is used to extract the D set. Since the previous partitioning nodes may be considered as virtual input nodes to the following partitioning nodes. the D sets of the following partitioning nodes are obtained by cube intersection on the D sets of the previous partitioning nodes. After extracting the D sets of the partitioning nodes. the D sets of the primary outputs and next state lines are extracted. Finally. the ON/OFF sets and the combinational excitation vectors are generated from the D sets.

Consider the circuit s27 from ISCAS'89 sequential benchmark circuits as an example. as shown in Figure 3.5. Assume that *extract_sets_and_excitation_vectors*( ) chooses the node $G11$ as the partitioning node and assigns it logic value D. The input cone of *G11* needs to be considered. as shown in Figure 5.9. Node $G11$ now represents the only currently assigned node in the node list of assignment and is assigned to logic value D. According to the B-rules. both inputs of the NOR gate $G11$ are assigned logic values $(0, \overline{D})$ and $(\overline{D}, 0)$. At this point. a new level of assignment list includes two nodes $G9$ and $G5$. Since node $G5$ is an input. its value is left unchanged and it is removed from the node list. From node $G9$. the procedure assigns values to nodes $G15$ and $G16$. The backward assignment process repeats until all values at the intermediate nodes are justified to the input nodes by using the B-rules. The D set $C_{G11}(G0, G1, G3, G5, G6, G7)$ of node $G11$ is extracted.

Then the procedure selects the primary output $G17$ and assigns logic value D to it. We only need to consider the input cone of $G17$. as shown in Figure 5.9. The D

```
Input   : A sequential circuit's netlist.
Output : The ON/OFF sets of the primary outputs and next state lines and
          the combinational excitation vectors.

procedure extract_sets_and_excitation_vectors( )
begin
   choose partitioning nodes according to the weight of each node:
   partition the circuit according to the partitioning nodes:
   for each partitioning node. primary output. and next state line do
   begin
      arrange the partitioning node. primary output. or next state line in the
         list of node assignments:
      assign logic value $D$ to it:
      while the list of node assignments is not empty do
      begin
         for each node in the list of node assignments
            execute the backward-assignment function:
         refresh the list of node assignments:
      end:
      get the $D$ sets by cube intersection to extend the cover sets of
         the previous partitioning nodes into the cover sets of inputs:
      get the ON/OFF sets and combinational test vectors from the $D$ sets:
   end:
   return the ON/OFF sets of the primary outputs and next state lines
      and the combinational excitation vectors:
end;
```

Figure 5.10. The Algorithm of Cover Extraction and Combinational Excitation Vector Generation.

set $C_{G17}(G11)$ of the primary output $G17$ is extracted by considering the node $G11$ as a virtual input to the input cone of $G17$. Cube intersection is performed to get the D set $C_{G17}(G0, G1, G3, G5, G6, G7)$ from $C_{G17}(G11)$ and $C_{G11}(G0, G1. G3. G5. G6, G7)$. The input node $G2$ is not a node in the input cone of the primary output $G17$. so its value is set to X in the D set of the primary output $G17$. Finally. the D set of the primary output $G17$ is $C_{G17}(G0, G1, G2. G3. G5, G6, G7)$.

The process repeats until the D sets of all primary outputs and next state lines are extracted. From the D sets, the ON/OFF sets and combinational excitation vectors are generated. The ON/OFF sets are represented as bit vectors which are similar to those used in ESPRESSO [6].

5.6 Pseudo-Combinational Circuit Test Generation for Fault-Oriented ATPG System

The combinational excitation vector generation discussed above is used for the fault-independent test generation algorithm for sequential circuits. Besides the fault-independent test generation algorithm. a fault-oriented test generation algorithm for sequential circuits is also used in the proposed ATPG system. In the fault-oriented test generation algorithm, a fault-oriented pseudo-combinational test generation method is used. The method considers one fault at a time.

In this dissertation, a Boolean satisfiability and implication graph formula is extracted that defines the structure of the related circuit and incorporates necessary conditions for fault activation and path sensitization. Then a SAT solver procedure is used to satisfy the formula.

5.6.1 Circuit Representation

In the 1960s and early 1970s, an algebraic or symbolic manipulation method called Boolean difference, differing from structural methods, was developed. This method did not achieve the popularity of the structural methods because of its complexity of computation. Since the combinational test generation method based on Boolean satisfiability was introduced in [32], this method has received more and more attention. In the following, the method of Boolean difference is described briefly.

Given a function $f(x) = f(x_1, x_2, ..., x_i, ... x_n)$ which describes the behavior of a combinational circuit, where $x_1, ..., x_n$ are the inputs of the circuit, the Boolean difference of $f(x)$ with respect to its ith input variable is defined as

$$\frac{df}{dx_i} = f(x_1, ..., x_i, ..., x_n) \oplus f(x_1, ... \overline{x_i}, ..., x_n) \qquad (5.1)$$

Then

$$\left[X_i^a \cdot \frac{df}{dx_i}\right]_T = 1 \qquad (5.2)$$

is the necessary and sufficient condition of fault x_i stuck at a detected by vector T, where $a = 1$ or 0, $X_i^1 = \overline{X_i}$, and $X_i^0 = X_i$. Equation 5.2 implies that the fault under test is first excited to the logic value opposite to the stuck-at value, and then the change of the logic value at the fault location can be observed at the primary outputs. In short, test generation for combinational circuits can be viewed as a search of an n-dimensional 0-1 space defined by the variables x_i $(1 \leq i \leq n)$ for points that satisfy the above equation.

When all flip-flops in a sequential circuit are disabled, the sequential circuit becomes a pseudo-combinational circuit. The digital combinational circuit can be represented as a set of unary, binary, ternary, and M-ary ($M > 3$) relations.

At first, the pseudo-combinational circuit is represented as the conjunctive normal form, i.e., CNF (also known as product of sums). As an example, a two input AND gate is used to illustrate how to get a CNF formula from a circuit, as shown in Figure 5.11.

The CNF formula of the AND gate is:

$$CNF = (\overline{Z} + X) \cdot (\overline{Z} + Y) \cdot (\overline{X} + \overline{Y} + Z) \tag{5.3}$$

If and only if the values of the variables are consistent with the truth table of the AND gate, Equation 5.3 equals to 1.

Figure 5.11 illustrates the CNF formulae for the basic gates (one or two inputs). In the CNF formula, one sum is called a *clause* and each term in a clause is called a *variable*. Clauses with one, two, or three variables are unary, binary, or ternary clauses, respectively. It is convenient to extend the basic CNF formulae in Figure 5.11 to gates which have more than two inputs. For example, the CNF formula for a NAND gate with three inputs X, Y, and W is shown in Figure 5.12.

The implication graph of the two-input AND gate can be obtained from the unary and binary clauses, as shown in Figure 5.13. There are two binary clauses $(\overline{Z} + X)$ and $(\overline{Z} + Y)$ in the AND gate. When the formula is satisfied, each clause should be satisfied. For instance, to meet the clause $(\overline{Z} + X)$, when $X = 0$, Z should be 0; when $Z = 1$, X should be 1. Two implications, $\overline{X} \rightarrow \overline{Z}$ and $Z \rightarrow X$, are obtained from the clause, where → means implication. The ternary or M-ary ($M > 3$) clauses cannot be transferred to the implication graph. But if one or more variables in the

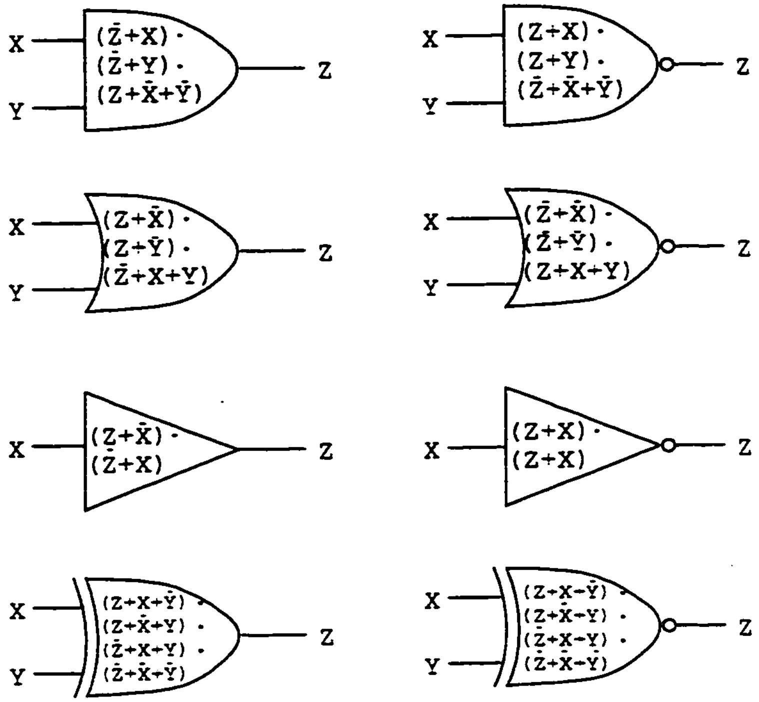


Figure 5.11. The CNF formulae of basic gates.

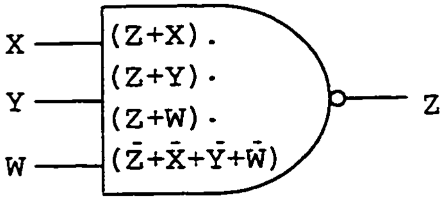


Figure 5.12. The CNF formula of 3-input NAND gate.

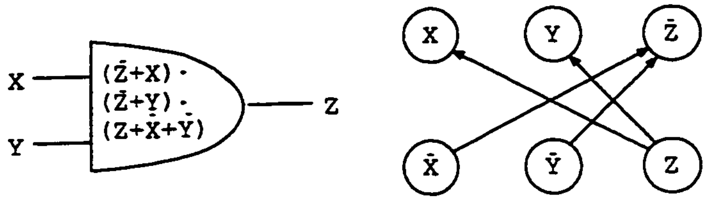


Figure 5.13. Implication graph of an AND gate.

clauses are known or assumed, the clauses become binary or unary clauses.

Consider a simple circuit S1 shown in Figure 5.14 and we want to derive a test for the fault G s-a-0. By extracting each formula for each gate in the circuit using the above method, the CNF formula for the output of the circuit is:

$$CNF = (G+\overline{A})\cdot(G+\overline{B})\cdot(\overline{G}+A+B)\cdot(E+C)\cdot(\overline{E}+\overline{C})\cdot(F+G)\cdot(F+E)\cdot(\overline{F}+\overline{G}+\overline{E}) \qquad (5.4)$$

The faulted circuit is produced by copying the original circuit, renaming all related variables, and disconnecting the faulted site (all faulted signals are labeled with "′"), as shown in Figure 5.15. Because of the fault G s-a-0, the signal G is always at logic value 0 no matter what values are at the inputs A and B. The signal G is disjointed into two signals: fault-free G and faulted D'. In order to detect the fault, G' has logic value of s-a-0 and G must have logic value 1.

Since the unfaulted and faulted circuits have the same behavior except those nodes that are affected by the fault, only the nodes that lie on a path between the fault site and a circuit output need to be renamed. The CNF formula for the faulted circuit is

$$CNF = (E+C)\cdot(\overline{E}+\overline{C})\cdot(\overline{G'})\cdot(F'+G')\cdot(F'+E)\cdot(\overline{F'}+\overline{G'}+\overline{E}) \qquad (5.5)$$

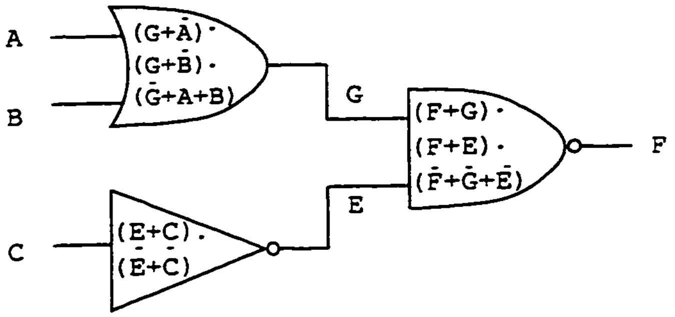


Figure 5.14. Formula extraction of a simple circuit S1.

It is not necessary to include the OR gate G in the CNF formula for the faulted circuit because of the implied discontinuity at the fault site.

According to Boolean difference, in order to detect the fault at G. the unfaulted and faulted circuits are put together and an XOR gate is added to their outputs. The final circuit is shown in Figure 5.16. In order to cover the fault G s-a-0. the output of the XOR gate BD should be 1. If the CNF formula equals to 1. a solution is found. Otherwise. no test exists. The formula of the final circuit is:

$$CNF = (G + \overline{A}) \cdot (G + \overline{B}) \cdot (\overline{G} + A + B) \cdot (E + C) \cdot (\overline{E} + \overline{C}) \cdot (F + G) \cdot (F + E) \cdot (\overline{F} + \overline{G} + \overline{E}) \cdot (\overline{G'}) \cdot (F' + G') \cdot (F' + E) \cdot (\overline{F'} + \overline{G'} + \overline{E}) \cdot (BD) \cdot (\overline{F} + F' + BD) \cdot (F + \overline{F'} + BD) \cdot (\overline{F} + \overline{F'} + \overline{BD}) \cdot (F + F' + \overline{BD}) \quad (5.6)$$

The problem of combinational circuit test generation is formulated as one of finding a consistent signal logic assignment which satisfies the above formula.

5.6.2 Signal Dependencies

On the basis of the CNF formula of the circuit, two kinds of signal dependencies are used: fixation and contradiction. If a path $x \rightarrow \overline{x}$ is found in the implication

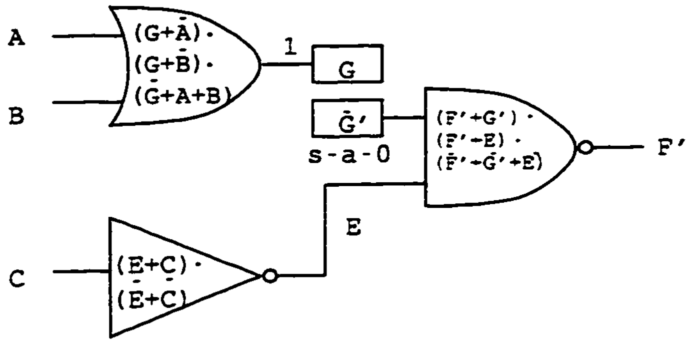


Figure 5.15. Formula extraction of the simple circuit S1 with a fault *G* s-a-0.

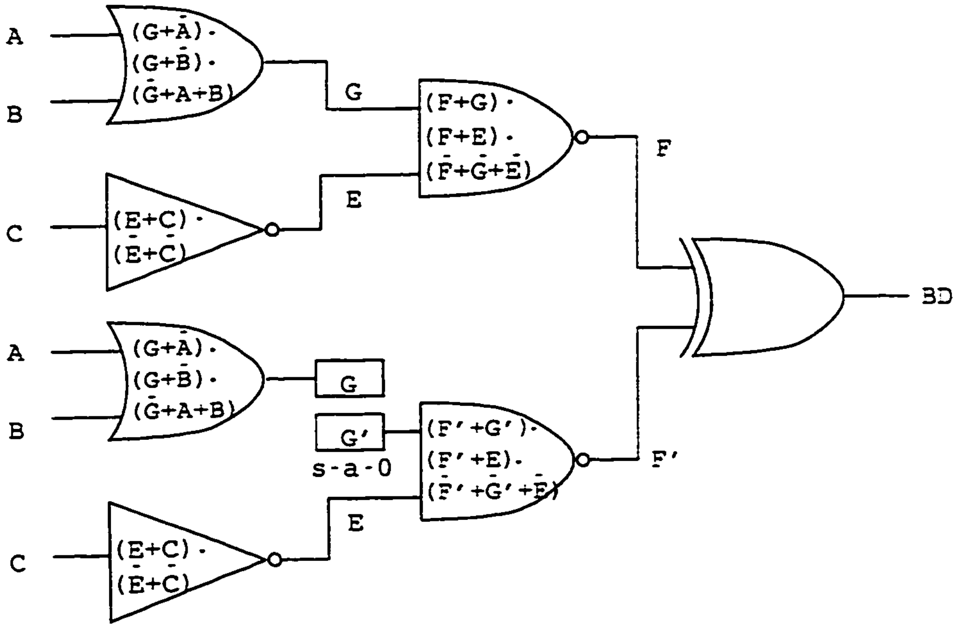


Figure 5.16. The XOR of the unfaulted and faulted circuits should be 1.

graph. it implies that x should be 0. Similarly. if a path $\overline{x} \rightarrow x$ is found. x should be 1. If both paths $x \rightarrow \overline{x}$ and $\overline{x} \rightarrow x$ are present in the implication graph at the same time. the contradiction exists and no solution exists. By using this method. we can find whether or not a variable is set to a given value. A breadth-first search algorithm is used to find a path between a variable x and its complement $\overline{x}$. The procedure of signal dependency computation is quite simple. as shown in Figure 5.17.

If a contradiction occurs in the signal dependency. it means that some variable(s) must be simultaneously assigned to logic values 0 and 1. In this case, there is no solution for this variable assignment. If signal values which have been determined satisfy the Boolean equation. the solution is found. Otherwise. the partial set of signal values determined may reduce some of the ternary relations to binary relations. These new binary relations are included in the implication graph and new signal dependencies should be determined. The process continues until no ternary or M-ary relations reduce to binary relations.

5.6.3 Pruning the Search Tree

Several methods are used to prune the search tree. According to our experience. the more constraints the variables have. the smaller the search tree. This is because when some variables are assigned to logic value 0 or 1. their relations with other variables may help in determining other unassigned variables' logic values easily.

If a fault can propagate to one or more outputs of the circuit, there must be at least one sensitized path (similar to D algorithm) from the fault site to the output. In this path, the fault-free and faulted values must be different. Suppose that if we add an XOR gate whose inputs are the fault-free and faulted values. the output of the XOR gate must be one. The concept is similar to the active line variables used

```
Input   : The directed graph $G = (V. E)$ and the assignment array of signals.
Output : The signal dependencies of the graph.

procedure signal_dependency( )
begin
   for each variable $v$ and its complement $\overline{v}$ do
   begin
      if a path from $v$ to $\overline{v}$ is found
         if $v$ is assigned to 1
            /* contradiction */
            return no solution:
         else $v$ is assigned to 0:
      if a path from $\overline{v}$ to $v$ is found
         if $v$ is assigned to 0
            /* contradiction */
            return no solution:
         else $v$ is assigned to 1;
   end:
   return the signal dependencies of the implication graph:
end:
```

Figure 5.17. The procedure of signal dependency computation.

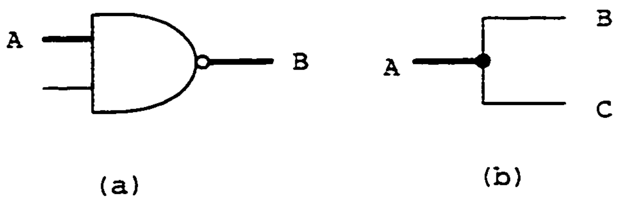


Figure 5.18. (a). If A is sensitized. B must be sensitized: $(\overline{EX_A} + EX_B)$. (b). If A is sensitized. either B or C must be sensitized: $(\overline{EX_A} + EX_B + EX_C)$.

in [32]. Letting A be the fault-free value. A' the faulted value. and EX the output of an XOR gate whose inputs are A and A'. we obtain these two clauses in ternary relation $(\overline{EX} + A + A') \cdot (\overline{EX} + \overline{A} + \overline{A'})$. If this path is active. that means $EX = 1$. A and A' must be different.

If a sensitized variable A has a single output B. the clause $(\overline{EX_A} + EX_B)$ is added. which means that if A is the sensitized variable, B is sensitized. Also. if the sensitized variable A has two outputs B and C. then the clause $(\overline{EX_A} + EX_B + EX_C)$ should be added. That means that if the variable A is sensitized. either the variable B or C must be sensitized. Figure 5.18 shows two examples of these clauses.

On the other hand. some vertices in the directed graph may belong to a *strongly connected component.* So these vertices can be considered as one vertex. When the value of a vertex is obtained. the other vertices in this strongly connected component can be easily determined.

Because of the duality of the implication graph. if some vertices belong to a strongly connected component. the corresponding complemented vertices must belong to another strongly connected component. For example. in the circuit S1. there is a path from C to $\overline{E}$ and a path from $\overline{E}$ to C. Vertices C and $\overline{E}$ belong to a strongly connected component, and their complemented vertices $\overline{C}$ and E must belong to another strongly connected component. After all implication relations with vertices C and $\overline{C}$

are transformed to vertices $\overline{E}$ and E. the vertices C and $\overline{C}$ can be deleted from the implication graph. By finding strongly connected components, the implication graph is condensed. The algorithm of finding strongly connected components in a directed graph can be found in [3].

Consider the simple circuit S1 with the fault G s-a-0 shown in Figure 5.15. In order to excite the fault. the logic values of G and G' must be different. The faulted value G' is 0, so the unfaulted value G should be 1. A clause (G) is added to the formula in Equation 5.6. G has one fanout F and an XOR gate has been added to the unfaulted line F and faulted line F'. After the sensitized path is considered. the CNF formula obtained for the fault is

$$CNF = (G+\overline{A})\cdot(G+\overline{B})\cdot(\overline{G}+A+B)\cdot(E+C)\cdot(\overline{E}+\overline{C})\cdot(F+G)\cdot$$
$$(F+E)\cdot(\overline{F}+\overline{G}+\overline{E})\cdot(\overline{G'})\cdot(F'+G')\cdot(F'+E)\cdot(\overline{F'}+\overline{G'}+\overline{E})\cdot(BD)\cdot$$
$$(\overline{F}+F'+BD)\cdot(F+\overline{F'}+BD)\cdot(\overline{F}+\overline{F'}+\overline{BD})\cdot(F+F'+\overline{BD})\cdot(G) \quad (5.7)$$

From the clause $(\overline{G'})$, we know that in order to satisfy the CNF formula. G' must be set to 0. The logic value of G' can be used to simplify the CNF formula. Since the clause $(\overline{F'}+\overline{G'}+\overline{E})$ is satisfied due to the logic value of G'. the clause is omitted. The clause $(F'+G')$ becomes (F'). From the new clause (F'), F' must be set to 1. The logic value of F' is used again to simplify the formula. The process continues until the formula can not be simplified any further. The final simplified CNF formula is:

$$CNF = (A+B)\cdot(E)\cdot(\overline{G'})\cdot(F')\cdot(\overline{F})\cdot(BD)\cdot(G) \quad (5.8)$$

An efficient SAT solver from SIS [59] is used to solve the above SAT formula. The branch and bound method is used in the SAT solver.

5.6.4 Combinational Circuit Test Generation Procedure

The test generation procedure for combinational circuits based on the ideas presented above is as follows.

1. Derive the CNF representation of the combinational circuit with the fault. The unary and binary clauses are saved in the implication graph, and ternary and M-ary ($M > 3$) clauses are saved in the satisfiability form (SAT form).

2. Simplify the implication graph by using signal dependencies. If a contradiction is found, the fault is combinational redundant. If variable values satisfy the Boolean equation, a solution is found without backtracking. Otherwise, only a partial set of variables is determined. These determined variables are used to reduce some ternary and M-ary ($M > 3$) clauses into binary clauses.

3. Find strongly connected components in the implication graph. A condensed implication graph is obtained.

4. The SAT solver from SIS is used to solve the problem.

5. If the variable assignments satisfy the Boolean equation, return the combinational excitation vector. If the excitation vector can constitute a test sequence for the fault in the sequential circuit, the test for the fault is found. Otherwise, a new combinational excitation vector should be found.

5.7 Summary

In this chapter, the backward assignment rules (B-rules) are presented to extract the ON/OFF sets of the primary outputs and next state lines. By using Lemma 1.

the backward assignment method can also generate the combinational excitation vectors. The consistent constraint is proposed to ensure that the backward assignments are logically correct. To make extraction efficient. an novel method is developed to partition circuits. In addition, a new fault-oriented combinational test generation algorithm based on the Boolean satisfiability and implication graph is proposed in this chapter. A Boolean satisfiability and implication graph formula is extracted that defines the structure of the related circuit and incorporates necessary conditions for fault activation and path sensitization. Then a SAT solver procedure is used to satisfy the formula.

CHAPTER 6

STATE JUSTIFICATION AND STATE DIFFERENTIATION

In test generation for combinational circuits, when a combinational excitation vector is found for a fault. the fault is detected. The situation is different for sequential circuit test generation. State justification and state differentiation are needed to justify the combinational excitation state and continuously propagate the effect of a fault on the next state lines to the primary outputs.

State justification and state differentiation play on important role in the proposed three-phase ATPG system. GLOBALTEST. In this chapter. state justification and state differentiation algorithms are described in detail.

6.1 State Justification

For a fault under test in a sequential circuit. in the three-phase ATPG system. a combinational excitation vector is found to propagate the fault to the primary outputs or the next state lines. State justification attempts to justify the combinational excitation state. If the excitation state covers the reset state. the fault can be excited from the reset state. Otherwise, state justification is used to justify the excitation state and find a justification sequence from the reset state to the excitation state S^E.

Initially. the state justification procedure tries to find a single-vector justification sequence from the reset state to the excitation state. The entire fan-in states S^E_{in}

of the excitation state S^E can be obtained by cube intersections on the ON/OFF sets of the next state lines. The cubes of fan-in states are chosen according to the excitation state. If a present state line in the excitation state has logic value 1(0). the ON set (OFF set) of the corresponding next state line is picked. If a present state line has logic value X. the next state line is ignored and nothing is picked. The cube intersection of the ON and OFF sets of the next state lines gives the fan-in states of the excitation state S^E. The ON/OFF sets of the next state lines include both primary input and present state parts. The present state vectors are used to check if they cover the reset state and to get their fan-in states if the next state justification is needed. The primary input vectors are used to generate a test sequence if the fault is detected. If the present states cover the reset state. the single-vector justification sequence is obtained.

If the single-vector state justification fails. a two-vector justification sequence is sought. This is performed by attempting to justify the fan-in states S^E_{in} via a single vector justification sequence. If the state justification procedure succeeds. a two-vector justification sequence is found. Otherwise. a three-vector justification sequence is attempted. The process is repeated for the fan-in states of the state currently justified.

When the fan-in states are obtained. these states should be disjointed from the previously used states to prevent cycles. The state justification procedure *state_justification*() is shown in Figure 6.1. Figure 6.2 shows the procedure of obtaining fan-in states of present state *get_fanins*().

Fault-free state justification is performed here since the cover sets are extracted in fault-free circuits. It may not always generate the correct justification sequence. Therefore. fault simulation is needed when a test sequence is found.

```
Input   : The excitation state state and ON/OFF sets of next state lines.
Output : A justification sequence from reset state to state if found:
          else return NOT-FOUND.

procedure state_justification(state)
begin
   /* put the primary input part of state into PI Stack */
   push state into PI Stack;
   fanins := get_fanins(state);
   for each fan-in state fanin in fanins
      if fanin covers the reset state
         return the state justification sequence saved in PI Stack:

   for each fan-in state fanin in fanins do
   begin
      state_justification(fanin):
      if the justification sequence is found
         return the state justification sequence:
   end:
   pop state from PI Stack():
   return (NOT-FOUND):
end:
```

Figure 6.1. State justification procedure.

Input   : The present state *state* and ON/OFF sets of next state lines.
Output : All fan-in states of *state* except those included in used states
(exist_state).

```
procedure get_fanins(state)
begin
   first_mark := TRUE;
   for each present state line that is a 1 or 0 do
   begin
      if first_mark is TRUE
      begin
         fanins := ON or OFF set of corresponding next state line;
         first_mark := FALSE;
      end;
      else
         fanins := fanins $\cup$ (ON or OFF set of corresponding next state line)
   end;
   /* do sharp produce to remove used cubes from fanins */
   sharp_product(fanins, exist_state);
   /* logic minimization */
   minimization(fanins);
   add_fanins_to_exist(fanins);
   return fanins;
end;
```

Figure 6.2. The procedure of obtaining fan-in states of present state.

Consider the fault $G2$ s-a-0 in the circuit s27 shown in Figure 3.5. One of the combinational excitation vectors is $(X, X, 1, X, X, X, 1)$ for $G0$, $G1$, $G2$, $G3$, $G5$, $G6$, and $G7$. The corresponding excitation state is $(X, X, 1)$ for $G5$, $G6$, and $G7$. The states of $G5$ and $G6$ are the logic value X, so we can ignore them. As line $G7$ is at logic value 1, we pick up the ON set of $G13$. The ON set of $G13$ is $(X, 1, 0, X, X, X, X)$ and $(X, X, 0, X, X, X, 1)$ for $G0$, $G1$, $G2$, $G3$, $G5$, $G6$, and $G7$. From the first vector in the ON set, we know that when $G1$ and $G2$ have logic values $(1, 0)$, $G13$ has a logic value 1. In the next clock cycle, $G7$ would be logic value 1. As the reset state $(0, 0, 0)$ implicates the states of $G5$, $G6$, and $G7$ (X, X, X) in the first vector of the ON set, the excitation state is reachable from the reset state. The initial justification process is given in Table 6.1, where 0/1 means that 0 is the unfaulted value and 1 is the faulted value, etc.

Table 6.1. Initial state justification process.

	primary inputs				present states			next states			PO
gates	G0	G1	G2	G3	G5	G6	G7	G10	G11	G13	G17
justification vector	X	1	0/0	X	X	X	X	X	X	1/1	X
excitation vector	X	X	1/0	X	X	X	1	X	X	0/1	X

Since the sequential circuit starts from the reset state, we should set the initial states of G5, G6 and G7 to (0, 0, 0). After fault simulation, the justification sequence is a valid justification sequence. The final state justification is shown in Table 6.2.

Table 6.2. Final state justification process.

	primary inputs				present states			next states			PO
gate	G0	G1	G2	G3	G5	G6	G7	G10	G11	G13	G17
justification vector	X	1	0/0	X	0	0	0	X	0	1/1	1
excitation vector	X	X	1/0	X	X	0	1	X	0	0/1	1

6.2 State Differentiation

In our algorithm. the flip-flops are disabled and the sequential circuits are converted into pseudo-combinational circuits. For pseudo-combinational circuits. faults can be divided into three kinds:

1) The fault site is a node in the input cones of the primary outputs. and a combinational excitation vector can be found for the fault.

2) The fault site is a node in the input cones of the primary outputs. but a combinational excitation vector cannot be found.

3) The fault site is not a node in any input cones of the primary outputs. but is a node in the input cones of the next state lines.

Consider the circuit shown in Figure 6.3. The fault C s-a-0 can be propagated to the primary output K and belongs to the first kind of fault. For the fault B s-a-1. though B is a node in the input cone of the primary output K. the fault cannot be propagated to the primary output K directly. The fault should be propagated to the next state line I first (the effect of the fault on I would be propagated to the present state line J in the next clock cycle). and then the effect of the fault on J is propagated to the primary output K. Therefore. the fault belongs to the second kind of fault. For the fault $F \rightarrow I$ s-a-0. as I is not a node in the input cone of the primary output

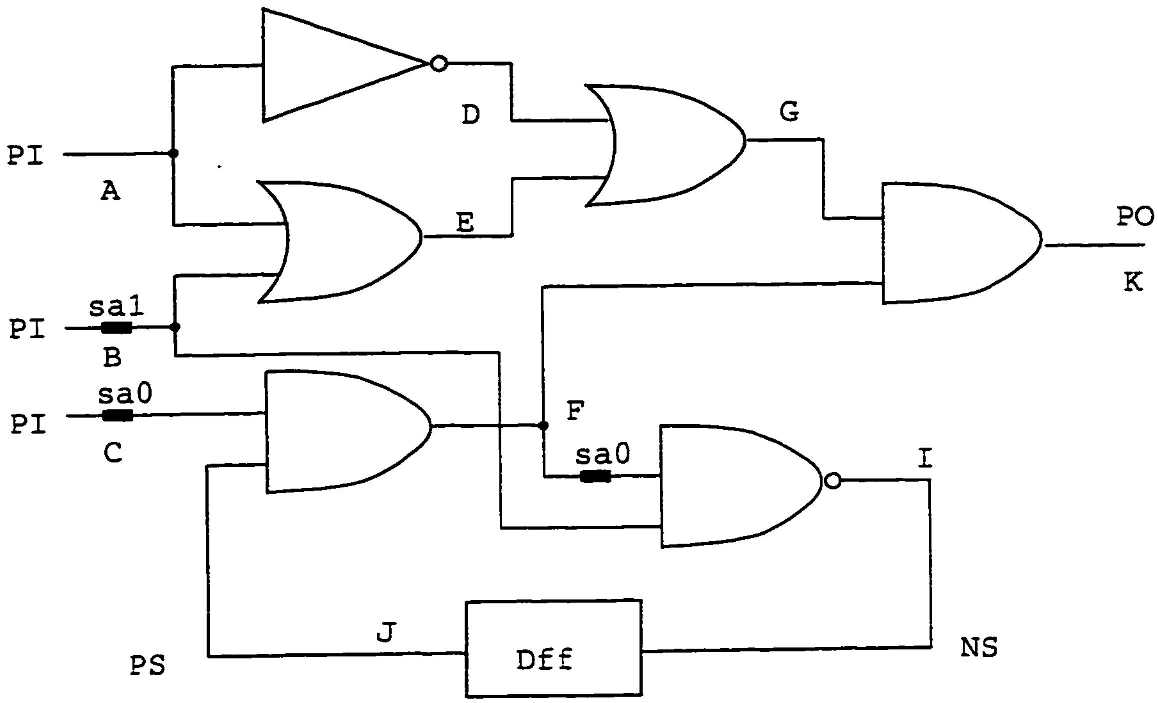


Figure 6.3. Three kinds of faults defined in our algorithm.

K, the fault belongs to the third kind of fault.

For the latter two kinds, these faults should be propagated to the next state lines first by using the test generation procedure for combinational circuits. If a combinational excitation state is found and justified, state differentiation is used to continuously propagate the effects of these faults on the next state lines to the primary outputs. If state differentiation succeeds, a differentiation sequence is found. Otherwise, the combinational excitation vector cannot constitute a test sequence for the sequential circuit.

When the excitation vector propagates the fault to the next state lines, the true state S_1^T is the state in the fault-free circuit and the faulty state S_1^F is the state in the faulty circuit. S_1^T and S_1^F are guaranteed to differ in at least 1 bit. Since the effect of the fault has been propagated to S_1^F, it is assumed that S_1^T and S_1^F are states in the fault-free circuit. The purpose of state differentiation is to find a differentiation sequence which causes S_1^T and S_1^F to have at least one different bit at the primary outputs.

To make the state differentiation procedure more time-efficient, a random differentiation sequence is used as a first step. Some random vectors are added to the sequence starting from the reset state to the excitation state, and the unspecified primary inputs in the whole sequence are assigned randomly specified logic values. The whole sequence is used to fault-simulate the fault. If the sequence can detect the fault, a test sequence is found. Otherwise, a deterministic state differentiation method is used.

The idea of deterministic state differentiation is presented as follows: According to the ON and OFF sets of every primary output, we search for a primary input vector which exists in both the ON and OFF sets where the present state parts of the ON (or OFF) set and the OFF (or ON) set cover S_1^T and S_1^F separately. If such a primary input vector is found, the primary input vector constructs a single-vector differentiation sequence. The procedure of single-vector state differentiation is shown in Figure 6.4. Otherwise, multi-vector differentiation sequences should be searched.

In multi-vector state differentiation, first we try to find a two-vector differentiation sequence, then a three-vector sequence and so on. We attempt to propagate the true state S_1^T and the faulty state S_1^F to the next state lines by using a method similar to the one used in single-vector state differentiation. Instead of using the ON/OFF sets of the primary outputs in single-vector state differentiation, the ON/OFF sets of the next state lines are used. If the new true and faulty states are not found, quit without a solution. Otherwise, the single-vector state differentiation method is used to find a single-vector differentiation sequence on the new true and faulty states (S_2^T, S_2^F) again. If found, a two-vector differentiation sequence is constructed. Otherwise, a three-vector differentiation sequence is attempted. The procedure of multi-vector state differentiation is shown in Figure 6.5.

Each time a pair of new true and faulty states is found, these states are disjointed from the used true and faulty states. Thus, cycles during state differentiation are prevented.

Since test generation for combinational circuits produces an excitation vector with as many don't care entries in the primary inputs and present state lines as possible, if we just use the above state differentiation procedure, in most cases a differentiation sequence may not be found even if it exists. This is because it is necessary to set some unspecified inputs in the test sequence to either 1 or 0. However, some primary inputs and present states obtained by the combinational test generation, state justification,

```
Input  : The true and faulty states $S^T$ and $S^F$. and the ON/OFF sets of
         primary outputs.
Output : A single-vector differentiation sequence if found:
         else return NOT-FOUND.

procedure single_vector_state_differ($S^T$, $S^F$)
begin
   for each primary output do
   begin
      /* find a primary input vector existed in the ON and OFF sets of
         the output */
      if (PI_vector := find_PI(ON-set. OFF-set)) is found do
      begin
         /* judge if $S^T$ implies the ON-set and $S^F$ implies the OFF-set */
         if Judge_implication($S^T$, ON-set, $S^F$, OFF-set) is TRUE
            return(PI_vector);
         /* judge if $S^T$ implies the OFF-set and $S^F$ implies the ON-set */
         if Judge_implication($S^T$, OFF-set. $S^F$. ON-set) is TRUE
            return(PI_vector);
      end;
   end:
   return (NOT-FOUND);
end;
```

Figure 6.4. The procedure of single-vector state differentiation.

```
Input    : The true and faulty states $S^T$ and $S^F$, and the ON/OFF sets of
           primary outputs and next state lines.
Output : A multi-vector differentiation sequence if found:
           else return NOT-FOUND.

procedure multi_vector_state_differ($S^T$, $S^F$)
begin
  /* find all (new) excitation vectors fanouts which propagate $S^T$ and $S^F$
     to the next state lines */
  get_next_differ_state($S^T$. $S^F$. fanouts):
  for each fan-out state fanout in fanouts do
  begin
     /* create new true and faulty states */
     create_new_states(fanout. $S_i^T$. $S_i^F$):
     /* use single state differentiation method */
     if single_vector_state_differ($S_i^T$, $S_i^F$) succeeds
        return (FOUND):
  end:
  for each new true and faulty states $S_i^T$ and $S_i^F$ in fanout
     if multi_vector_state_differ($S_i^T$, $S_i^F$) succeeds
        return (FOUND):
  return (NOT-FOUND):
end:
```

Figure 6.5. The procedure of multi-vector state differentiation.

and state differentiation may have some don't care entries. Therefore. in order to detect the fault, these don't care entries in the primary inputs and present state lines have to be determined.

In STEED [18], all possible assignments to the unspecified inputs have to be made before it can be concluded that a test for the fault under consideration does not exist. There exists 2^n possible assignments for n unspecified inputs. Considering that each possible assignment may need to perform state justification. the real search space is much larger than 2^n.

We have proposed a backward deterministic method for state differentiation. This method can help in finding the differentiation sequence and determining the don't care entries. When the fault is attempted to propagate to a primary output or next state line in state differentiation. if some present state lines in the current clock cycle are don't care entries while the same bits in the ON and OFF sets of the primary output or next state line are deterministic logic values (e.g.. 0 or 1). it is known that. in order to obtain a 'differentiation sequence. these present state lines must be set to the deterministic logic values. After these present state lines are set to the same deterministic logic values as in the ON and OFF sets. a new problem arises. i.e.. whether the new specific present state is still justified from the previous clock cycles.

To solve the problem. in the backward deterministic method a revised state justification algorithm is presented to justify the specific present states. Because a justification sequence has been found from the reset state to the excitation state. in the revised state justification algorithm, it is just needed to specify some don't care entries in the justification sequence. When the specific present state requires that some of the don't care entries of the present state lines of the last clock cycle be set to deterministic logic values, the revised state justification procedure is used again to justify the modified present state of the last clock cycle. The process continues until no more specified states need to be justified. If the specific present state is reached from the previous clock cycles, the state differentiation process continues. Otherwise. a differentiation sequence cannot be found for the true and faulty states.

When some don't care entries of the primary inputs need to be set to deterministic logic values. we just set them according to the ON/OFF sets and justification is not required. The backward deterministic procedure for single-vector state differentiation is shown in Figure 6.6.

When the backward deterministic procedure for single-vector state differentiation fails to find a single-vector differentiation sequence, a backward deterministic procedure for multi-vector state differentiation is used. The procedure is similar to the multi-vector state differentiation procedure shown in Figure 6.5. The only two differences are:

- The backward deterministic procedure for single-vector state differentiation. as shown in Figure 6.6, is used to replace the single-vector state differentiation procedure shown in Figure 6.4.
- In order to propagate the fault to the next state lines. if some don't care bits in the state need to be set to specific values. we set them to the required values and then justify if the new specific state is reachable from the previous states.

To explain the idea of backward deterministic state differentiation. we continue to consider the fault G2 s-a-0 in circuit s27 shown in Figure 3.5 as an example. The fault has been propagated to the next state line G13 and the justification sequence has been found. From Table 6.2. the true and faulty states are $(X.0.0)$ and $(X.0.1)$

```
Input   : The true and faulty states $S^T$ and $S^F$. and the ON/OFF sets of
          primary outputs.
Output : A single-vector differentiation sequence if found:
          else return NOT-FOUND.

procedure single_vector_back_state_differ($S^T$. $S^F$)
begin
   for each primary output do
   begin
      /* find a PI vector existed in the ON and OFF sets of the output */
      if (PI_vector := find_PI(ON-set, OFF-set)) is found do
      begin
         /* get intersections: $S^T \cap$ ON-set, and $S^F \cap$ OFF-set */
         if intersections($S^T$, ON-set, $S^F$, OFF-set) is not empty do
         begin
            /* judge if some bits in $S^T$ and $S^F$ are X. while the same bits in
               both intersections are deterministic values */
            if some bits are needed to be set to specific values do
            begin
               /* set these bits to the specific values. and then justify the new
                  deterministic state is reachable from the previous states */
               set_new_state();
               if new_state_justification() succeeds return (PI_vector):
            end:
            else return PI_vector:
         end:
         /* get intersections: $S^T \cap$ OFF-set. $S^F \cap$ ON-set */
         if intersections($S^T$. OFF-set, $S^F$. ON-set) is not empty do
         begin
            if some bits are needed to be set to specific values
               set_new_state():
               if new_state_justification() succeeds return (PI_vector):
            else return PI_vector;
         end;
      end;
   end;
   return (NOT-FOUND);
end;
```

Figure 6.6. The backward deterministic procedure for single-vector state differentiation.

for lines G5, G6 and G7. The ON and OFF sets of primary output G17 are shown in Table 6.3.

Table 6.3. The ON and OFF sets of primary output G17.

	primary inputs				present states		
gate	G0	G1	G2	G3	G5	G6	G7
ON-set	1	X	X	X	X	X	1
	1	1	X	X	X	X	X
	X	1	X	X	X	0	X
	X	X	X	X	X	0	1
	1	X	X	0	X	X	X
	X	X	X	0	X	0	X
	X	X	X	X	1	X	X
OFF-set	X	0	X	1	0	X	0
	0	X	X	X	0	1	X

The intersection of the primary input part on the first vectors of the ON set and the OFF set is not empty, i.e., $(1, 0, X, 1)$. The present state in the first vector of the ON set is $(X, X, 1)$, and its intersection with the faulty state $(X, 0, 1)$ is not empty, i.e., $(X, 0, 1)$. The present state lines in the first vector of the OFF set are $(0, X, 0)$, and its intersection with the true state $(X, 0, 0)$ is not empty, i.e., $(0, 0, 0)$. In order to propagate the fault to the primary output, the first bit should be set to logic value 0. After setting, the differentiation vector becomes $(1, 0, X, 1, 0, 0, 0)$. For the primary inputs, we just set them to the new logic values. But the new differentiation state $(0, 0, 0)$ should be justified if it is reachable from the previous clock cycle.

From the partial OFF set of next state line G10, $(0, X, X, X, X, X, X)$, we know that if line G0 in the previous clock cycle is set to 1, the new state would be reached from the previous clock cycle. The original vector in the previous clock cycle is the excitation vector, $(X, X, 1, X, X, 0, 1)$ and its intersection with the OFF set of next state line G10 is not empty, i.e., $(0, X, 1, X, X, 0, 1)$. The excitation vector should be set according to the intersection. As all bits needed to be set are in the primary inputs. we just change the original logic value X to the new deterministic value. The differentiation sequence is found. The final test sequence is composed of the justification sequence, the excitation vector, and the differentiation sequence shown in Table 6.4. After finding the test sequence, we use it to fault simulate the fault G2 s-a-0. and the result is the same with Table 6.4.

Table 6.4. The process of exciting the fault G2 s-a-0 to primary output G17.

	primary inputs				present states			next states			PO
gate	G0	G1	G2	G3	G5	G6	G7	G10	G11	G13	G17
justification vector	X	1	0/0	X	0	0	0	X	0	1/1	1
excitation vector	0	X	1/0	X	X	0	1	0	0	0/1	1
differentiation vector	1	0	X/0	1	0	0	0/1	0/1	1/0	0/1	0/1

It is shown above that, in state differentiation, the fault is initially attempted to be propagated to the primary outputs. If this fails, the fault is attempted to be propagated to the next state lines. Then, in the following clock cycles, the effect of the fault on the next state lines is attempted to be propagated to the primary outputs. An important problem in multi-vector state differentiation is deciding which next state line is attempted first. The next state lines in a sequential circuit can be divided into two kinds:

- The corresponding present state lines of the next state lines are the nodes in the input cones of some primary outputs.

- The corresponding present state lines of the next state lines are not the nodes in the input cones of some primary outputs. They are the nodes in the input cones of some next state lines.

To find a shorter test sequence, the first kind of the next state lines is attempted first. If a differentiation vector cannot be found for the next state lines, the second kind of the next state lines is attempted.

In our experiments on the ISCAS'89 benchmark circuits, an interesting phenomenon about state differentiation is found, as shown in Figure 6.7. When a combinational excitation vector for a fault is found which excites the fault to one of the next state lines, a number of differentiation vectors are needed to propagate the effect of the fault on the next state lines to some next state lines of the second kind. In this case, the state of the total next state lines is changed so that in some next clock cycle, the effect of the fault on the next state lines of the second kind can be propagated to the next state lines of the first kind. Finally, the effect of the fault on the next state lines of the first kind is propagated to the primary outputs.

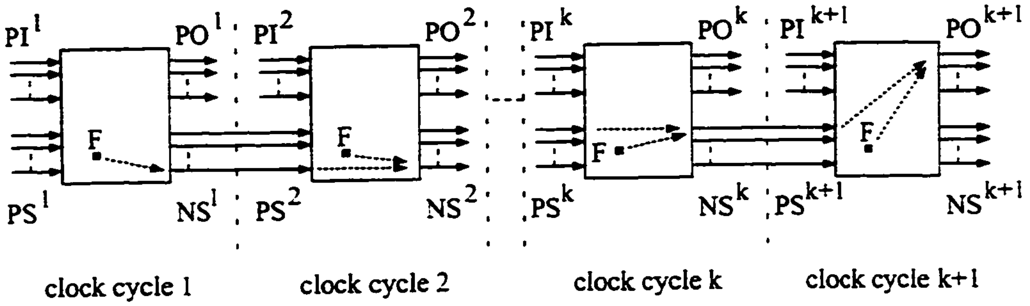


Figure 6.7. A general state differentiation process.

Consider the state differentiation process for a fault $CLRB \rightarrow UC\_17VD$ s-a-0 in circuit s382 from ISCAS'89 benchmark circuits, as shown in Table 6.2.

The first column gives the primary input vector (*PI*) and the next 21 columns give the states of 21 next state lines (*NS1* to *NS21*). Initially, the combinational excitation vector (*vector*) excites the fault to the next state line *NS15*. The next state line *NS15* belongs to the second kind of the next state lines, so it is needed to propagate the fault to the first kind of next state lines before the fault is propagated to one of the primary outputs. After some primary input vectors, the effect of fault is propagated to the next state line *NS3*. Since the next state line *NS3* belongs to the first kind of the next state lines, the effect of the fault on the line *NS3* can be propagated to the primary outputs.

6.3 Summary

In this chapter, two important steps in our ATPG system, state justification and state differentiation, are described in detail. State justification and state differentiation are efficiently performed using cube intersection on the ON/OFF sets of the primary outputs and next state lines. To increase the efficiency of the existing state differentiation in dealing with the unspecified inputs in the excitation vector and justification sequence, a backward deterministic method for state differentiation is presented. In addition, the order of choosing the next state lines in state differentiation is important to the quality of the test sequence.

Table 6.5. State differentiation process for the fault $CLRB \rightarrow UC\_17VD$ s-a-0 in circuit s382 from ISCAS'89 sequential benchmark circuits.

PI	NS1	NS2	NS3	NS4	NS5	NS6	NS7	NS8	NS9	NS10	NS11	NS12	NS13	NS14	NS15	NS16	NS17	NS18	NS19	NS20	NS21
vector	0/0	0/0	0/0	1/1	1/1	1/1	1/1	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/1	0/0	0/0	0/0	0/0	0/0	0/0
000	0/0	0/0	0/0	1/1	1/1	1/1	1/1	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/1	0/0	0/0	0/0	0/0	0/0	1/1
000	0/0	0/0	0/0	1/1	1/1	1/1	1/1	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/1	0/0	0/0	0/0	0/0	1/1	1/1
000	0/0	0/0	0/0	1/1	1/1	1/1	1/1	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/1	0/0	0/0	0/0	1/1	0/0	0/0
000	0/0	0/0	0/0	1/1	1/1	1/1	1/1	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/1	0/0	0/0	0/0	1/1	0/0	1/1
000	0/0	0/0	0/0	1/1	1/1	1/1	1/1	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/1	0/0	0/0	0/0	1/1	1/1	0/0
											. . . . .										
000	0/0	0/0	0/0	1/1	1/1	1/1	1/1	0/0	0/0	0/0	0/0	1/1	0/0	0/1	1/0	0/0	1/1	0/0	1/1	1/1	1/1
000	0/0	0/0	0/0	1/1	1/1	1/1	1/1	0/0	0/0	0/0	0/0	1/1	0/0	0/1	1/0	0/0	1/1	1/1	0/0	0/0	0/0
000	0/0	0/0	0/0	1/1	1/1	1/1	1/1	0/0	0/0	0/0	0/0	1/1	0/0	0/1	1/0	0/0	1/1	1/1	0/0	0/0	1/1
000	0/0	0/0	0/0	1/1	1/1	1/1	1/1	0/0	0/0	0/0	0/0	1/1	0/1	0/0	1/0	1/0	0/0	0/0	0/0	0/0	0/0
000	0/0	0/0	0/1	1/1	1/1	1/1	1/0	0/0	0/0	0/0	0/0	1/1	0/1	0/0	1/0	1/0	0/0	0/0	0/0	0/0	1/1
000	state differentiation succeeds																				

CHAPTER 7

TEST COMPACTION OF SEQUENTIAL CIRCUITS

7.1 Introduction

Test compaction is very important in test generation, since it allows to reduce test application time and test storage requirement of VLSI testers. The goal of test compaction for sequential circuits is to generate compact test sequences. It is especially important for circuits using a scan design. as the test application time increases proportionally to the product of the number of tests and the number of flip-flops in the scan chain. Therefore, it is preferable to have a test set with shorter length. provided the fault coverage is not compromised.

Usually, the test vectors generated by an ATPG system are not compact. It is necessary to include test compaction methods into ATPG systems. such as COMPACTEST [44], ROTCO [52], and COINS [47]. The first two methods are used for combinational circuit test compaction. The third one is for sequential circuit test compaction. Test compaction for sequential circuits is much more difficult than for combinational circuits.

The methods of test compaction can be classified as static and dynamic. Static methods attempt to reduce the length of test vectors in a given test set [57, 29]. Static test compaction is usually applied as a postprocessing step. after the test sequences have been generated by an ATPG system. Dynamic methods consider test compaction during the generation of the test set [19. 30. 10]. When a test sequence for a fault is generated, dynamic test compaction attempts to generate short test sequences. It is indicated that dynamic test compaction is very hard and computationally expensive to implement in sequential circuit test generation [43]. The advantage of static compaction is that, since the test set is available. it is easier to find optimal or near optimal solutions among the test sets.

The existing techniques for static test compaction of sequential circuits attempt to merge two or more test sequences into one test sequence [43], and some vectors of one sequence may be omitted and then inserted into another sequence. Thus the test sequences in a test set are significantly modified. This may affect the original fault coverage and also be time-consuming. In this chapter. a static test compaction algorithm is proposed. Test compaction is performed on a given test sequence set for all detected faults. The test compaction problem is formulated as the set covering problem. An efficient set covering algorithm is proposed to compact test vectors. It attempts to select the necessary test vectors for the faults detected and eliminate other redundant test vectors. Then a local reduction and expansion algorithm is presented to further compact test vectors. The original fault coverage is not compromised. Experimental results on the ISCAS'89 benchmark circuit. discussed in section 8.3. showed that this test compaction algorithm can efficiently and quickly provide very good performance.

The rest of the chapter is organized as follows. The formulation of test compaction as a set covering problem is described in Section 7.2. In Section 7.3. a multispace search test compaction algorithm is presented. Section 7.4 summarizes this work.

7.2 The Formulation of Test Compaction Problem to Set Covering Problem

The set covering problem is an optimization problem that models many resource-selection problems. It is an NP-complete problem [17]. Given an instance $(Z.F)$. where Z is a finite set and a family F consists of sets S_1. S_2, $\cdots$. S_n. such that every element of Z belongs to at least one subset in F. The problem is to find a minimum-size subfamily C (cover) of k sets S_{i_1}, S_{i_2}, $\cdots$. S_{i_k} such that

$$\bigcup_{j=1}^{k} S_{i_j} = \bigcup_{j=1}^{n} S_j.$$

Consider an example of the set covering problem. as shown in Figure 7.1. There are 12 black points in Z and 6 sets in F. The minimum-size set cover is $C = \{S_1.S_2.S_3\}$.

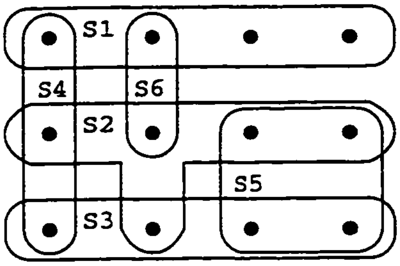


Figure 7.1. An example of the set covering problem.

The weighted set covering problem is a generalization of the set covering problem. Each set S_i on the family F has an associated weight w_i and the weight of a cover C is $\sum_{S_i \in C} w_i$. We need to find a minimum-weight cover. i.e..

$$\min\,(\sum_{j=1}^{k} w_{i_j})$$

subject to:

$$\bigcup_{j=1}^{k} S_{i_j} = \bigcup_{j=1}^{n} S_j.$$

When $w_i = 1$ for all i, the weighted set covering problem becomes the set covering problem.

The test set for a sequential circuit consists of test sequences, where each test sequence consists of a number of test vectors and can detect some faults in the sequential circuit. Each fault is detected by at least one test sequence. All undetected faults are not needed to be considered in test compaction.

The *test compaction* problem for sequential circuit test generation can be defined as: Given a sequential circuit, the ATPG system generates a test set with the shortest length to ensure the maximum fault coverage. For static test compaction, the ATPG system has generated a set of redundant test vectors for sequential circuits. So the static test compaction problem for sequential circuits can be simplified as: Given a set of test sequences, it is needed to select a test set with the shortest length among the given test vectors to ensure the maximum fault coverage. The problem is whether it is possible to remove some redundant test vectors without compromising the original fault coverage.

Example *ex1*: To show this, consider a simple example *ex1*, as shown in Table 7.1. There are three test sequences and six faults. For the first sequence, 2(1) means that the first two vectors can detect the fault 1, and so on. It is obvious that some

Table 7.1. An example *ex1* of test compaction for sequential circuits with three test sequences and six faults.

Sequences	Vectors and faults detected		
1	2(1)	5(2, 3)	6(4)
2	3(4)	4(1, 5)	5(2, 3)
3	2(6)	7(5)	

vectors in the three sequences are redundant. The problem is how to find a test set with the shortest length.

It is noted that it is not always necessary to apply the whole test sequence to detect every fault. For example, to detect fault 1. the first two vectors in the first sequence are enough. The first four vectors in the second sequence can also detect fault 1.

In order to formulate the test compaction problem. a subsequence concept is proposed. A subsequence is a portion of a test sequence. Each test sequence can be divided into several subsequences according to the faults detected. Sequence 1 can be divided into three subsequences: S_{1-1}, S_{1-2}, and S_{1-3}. Subsequence S_{1-1} consists of the first two vectors of sequence 1. Its length is 2 and can detect fault 1. Subsequence S_{1-2} consists of the first five vectors of sequence 1 and detects faults 1. 2. and 3. and so on. Therefore, we can get Table 7.2 from Table 7.1. It is known that subsequence S_{1-1} is included in subsequence S_{1-2}, and so on. All faults detected by subsequence S_{1-1} can be detected by subsequence S_{1-2}. □

Table 7.2. The original there sequences in the example *ex1* can be divided into eight subsequences.

Sequences	Length	Faults detected
1-1	2	1
1-2	5	1. 2. 3
1-3	6	1, 2, 3, 4
2-1	3	4
2-2	4	1, 4, 5
2-3	5	1, 2, 3, 4, 5
3-1	2	6
3-2	7	5, 6

A sequence consists of a number of subsequences and all related subsequences are considered to be contained in the sequence. There is at most one subsequence of one sequence which can be chosen for the chosen sequence set. The reason for this is quite straightforward. If two or more subsequences of one sequence are chosen, all subsequences with the shorter length are covered by the subsequence with the longest length. All faults detected by the subsequence with the shorter length can be detected by the corresponding subsequence with the longest length. Therefore, only the subsequence with the longest length is kept in the chosen sequence set. When one subsequence seq_1 of one sequence is needed to be replaced by another subsequence seq_2 of the same sequence, the subsequence seq_2 can be considered to be obtained by addition/removal of some test vectors to/from the sequence seq_1.

According to the subsequence concept, each subsequence can be considered as a set. The faults detected by the subsequence refer to the elements of the set X. The subsequences are considered as the sets in F. The length of the subsequence is regarded as the weight of the sequence. So the test compaction problem can be formulated as the weighted set covering problem. The set representation of the example *ex1* is shown in Figure 7.2, where $F1, F2, \cdots, F6$ are six faults.

It is necessary to choose a subfamily of subsequences with the minimum number of vectors while the total testable faults are still tested by the chosen subsequences.

In the next section, a set covering algorithm is presented to solve the test compaction problem. Then a local reduction and expansion algorithm is developed to further compact test vectors.

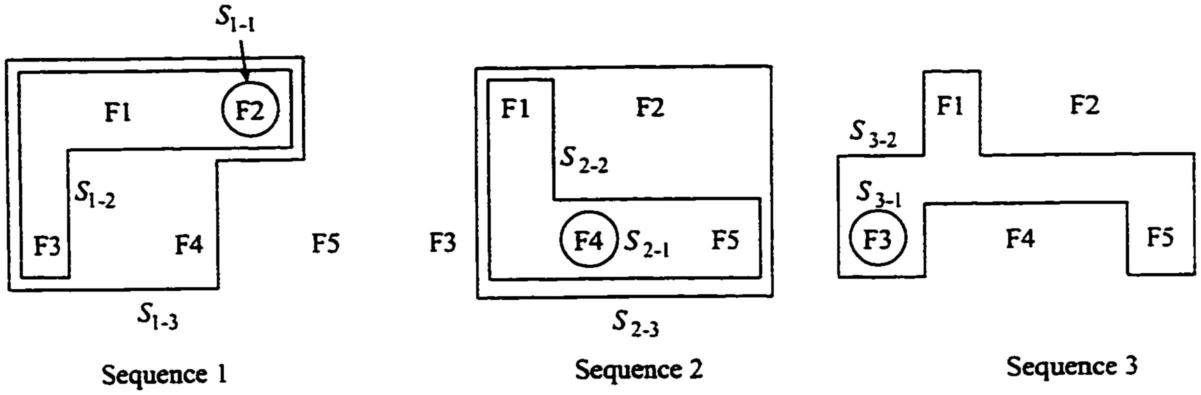


Figure 7.2. The formulation of test compaction for sequential circuits as the set covering problem.

7.3 A Multispace Search Algorithm for Test Compaction of Sequential Circuits

Multispace search is a new optimization approach developed to handle difficult situations in search and optimization. It can improve the performance of the traditional value search methods [23. 22. 25].

In multispace search. any active component related to the given problem structure can be manipulated and thus can be formulated as an independent search space. For a given optimization problem. for its variables. values. constraints. objective functions. and key parameters (that affect the problem structure). we define a variable space. a value space (i.e.. the traditional search space). a constraint space. an objective function space. a parameter space. and other search spaces. respectively. The totality of all the search spaces constitutes a *multispace* (see Figure 7.3).

The basic idea of multispace search is simple. Instead of being restricted in the value space. the multispace is taken as the search space. In the multispace, components other than values can be manipulated and optimized as well. During the search, a multispace search algorithm not only alters values in the value space. it also walks

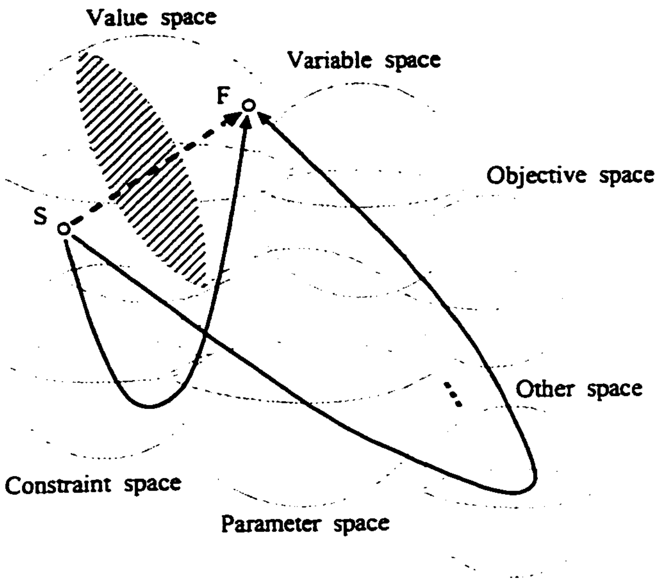


Figure 7.3. In the value space, a traditional search process (dashed line) cannot pass a "wall" of high cost search states (hatched region). It fails to reach the final solution state, *F*. A multispace search process (solid lines) scrambles across different search spaces. It could bypass this "wall" through the other search spaces.

across the variable space and other active spaces, dynamically changes the problem structure related to variables, constraints, parameters, and other components, and systematically *constructs* a sequence of intermediate problem instances. Each intermediate problem instance is solved by an optimization algorithm and the solution found is used as the initial solution to the next intermediate problem instance to be constructed in the next iteration. By interplaying value search and structured operations, multispace search solves the sequence of intermediate problem instances incrementally constructs the final solution, through a sequence of structured intermediate problem instances. Only at the *last* moment of search, the reconstructed problem structure approaches the original problem structure, and thus the final value assignment represents the solution to the given search problem.

The proposed multispace search algorithm for test compaction of sequential circuits (HICOMPACT) mainly consists of two parts: a fast set covering algorithm for test compaction and a local reduction and expansion algorithm for test compaction.

An initial solution of test compaction is obtained in the fast set covering algorithm for test compaction. The test vectors obtained with this algorithm may not have the shortest length, but it is a good starting point for the succeeding local reduction and expansion algorithm.

It is noted that, each time a corresponding subsequence, instead of the original whole test sequence, is chosen according to a fault. When two subsequences corresponding to the same test sequence are chosen, they are merged in the longer subsequence.

An effective *local reduction and expansion* (LRE) technique was developed based on multispace operations. LRE is effective to reduce the length of test sequences.

Since the number of faults affected may be more than one. multiple faults should be considered. To avoid being stuck at local minima, we will always consider a new test sequence whose length was not reduced. After a sequence multispace operations. if the new length of the chosen sequence set is reduced, the new chosen sequence set is taken as the new solution. The process repeats until no more improvement is found.

7.4 Summary

The test compaction problem is very important in test generation. since it allows to reduce test application time and test storage requirement of VLSI testers. In this chapter, an efficient algorithm was presented to solve the problem. The test compaction problem is formulated to the set covering problem by introducing the subsequence concept. A fast set covering algorithm is presented to solve the test compaction problem for sequential circuits. Then a local reduction and expansion algorithm is developed to further compact the test set. The algorithm is capable of find satisfactory solutions for ISCAS'89 sequential benchmark circuits. The performance is achieved because of an efficient set covering algorithm and an efficient local reduction and expansion algorithm for test compaction.

CHAPTER 8

EXPERIMENTAL RESULTS

In this chapter. the ISCAS'89 sequential benchmark circuits are used to evaluate the proposed ATPG system for sequential circuits. At first. the ISCAS'89 benchmark sequential circuits used in this dissertation are briefly described. Then the ISCAS'89 benchmark circuits are used to evaluate the proposed global test generation system for sequential circuits. called GLOBALTEST. Finally, the experimental results of the test compaction algorithm for sequential circuits called HICOMPACT are given.

8.1 ISCAS'89 Benchmark Circuits

In order to accurately evaluate a test generation system. real circuit examples should be used. Benchmark circuits constitute good circuit examples to evaluate a test system and are also suitable for comparing results with other test generation systems. The ISCAS'89 sequential benchmark circuits [8] have been used to evaluate the proposed test system GLOBALTEST and test compaction algorithm HICOMPACT. None of the ISCAS'89 benchmark examples have a specified reset state. It is assumed that a vector of all zeros is the reset state, as in [12. 18]. Table 8.1 shows a subset of the ISCAS'89 benchmark circuits used in this research. The five columns give the name and the numbers of primary inputs, primary outputs, flip-flops. and gates of each circuit.

Table 8.1. ISCAS'89 sequential benchmark circuit characteristics.

circuit	pi	po	dff	gate
s298	3	6	14	119
s344	9	11	15	160
s349	9	11	15	161
s382	3	6	21	158
s386	7	7	6	159
s400	3	6	21	162
s444	3	6	21	181
s510	19	7	6	211
s526	3	6	21	193
s641	35	24	19	379
s713	35	23	19	393
s820	18	19	5	289
s953	16	23	29	418
s1196	14	14	18	529
s1238	14	14	18	510
s1423	17	5	74	657
s1494	8	19	6	647
s5378	35	49	179	2779

8.2 Evaluation of the Proposed Test Pattern Generator

The test generation algorithm for sequential circuits described in the previous chapters has been implemented in the program GLOBALTEST. It consists of about 25 000 lines of C code and runs in a UNIX environment. Table 8.2 gives the statistics of running GLOBALTEST for sequential circuit test generation. Experiments were performed on a SUN Sparc 20 workstation. For each circuit, the total number of faults (*total faults*), the number of detected faults (*det. fault*) and the number of faults that were proven redundant (*red. fault*) are given. The fault coverage (*coverage*) includes detected faults. The test efficiency (*efficiency*) includes detected and provably redundant faults. The next column (*test time*) reports the execution times for test generation in CPU seconds and includes test compaction time. The total number of test vectors in test sequences is given in the column *compact test vec.* The test compaction time (*compact time*) is CPU time needed to perform test compaction. The test compaction time only takes a small portion of total test generation time.

The experimental results of the proposed system GLOBALTEST are compared with those of STEED [18] and VERITAS [12] on the ISCAS'89 benchmark circuits. as shown in Table 8.3. All these system have assumed a vector of all zero to be the reset state for the circuits. The test efficiency (*effici.*) includes detected and provably redundant faults. The total number of test vectors (*vectors*) and the CPU time (*time*) are also compared. The CPU times of STEED and VERITAS were on DEC 5000/200 DECstation 5000 model 200 uses a 64 KB instruction cache and 64 KB data cache. It has a performance rating of 18.5 SPECmarks [50]. The System Performance Evaluation Cooperative (SPEC) is a vendor-sponsored benchmarking project. A performance index called the SPECmark is defined as the geometric mean of the relative performance of the ten separate programs in the SPEC suite.

Table 8.2. Real execution performance of our algorithm GLOBALTEST on a SUN Sparc 20 with the ISCAS'89 sequential benchmark circuits with reset.

circuit	total faults	det. fault	red. fault	coverage (%)	efficiency (%)	test time(s)	compact test vec.	compact time(s)
s298	308	273	35	88.64	100	2.4	99	0.23
s344	342	337	5	98.54	100	5.5	38	0.21
s349	350	343	7	98.00	100	7.6	45	0.34
s382	399	379	20	94.99	100	252	1536	0.23
s386	384	314	70	81.77	100	9.8	168	0.03
s400	424	397	27	93.63	100	663	1850	0.22
s444	474	439	35	92.62	100	975	1871	0.14
s510	564	564	0	100	100	14.6	210	0.31
s526	555	466	89	83.96	100	1023	2637	0.16
s641	467	408	59	87.37	100	25.7	398	0.55
s713	581	480	101	82.62	100	1001	216	0.64
s820	850	814	35	95.76	99.88	46	542	8.11
s953	1079	1069	10	99.07	100	20.2	436	0.94
s1196	1242	1239	3	99.76	100	95	469	5.45
s1238	1355	1283	72	94.69	100	78	389	1.85
s1423	1515	1007	14	66.47	67.39	1565	1232	0.59
s1494	1506	1455	51	96.61	100	238	598	6.39
s5378	4603	3375	915	73.32	92.66	3761	515	16.5

Table 8.3. Test generation comparison with STEED and VERITAS on ISCAS'89 benchmark circuits.

circuit	GLOBALTEST			STEED			VERITAS		
	effici. (%)	vectors	time (s)	effici. (%)	vectors	time (s)	effici. (%)	vectors	time (s)
s298	100	99	2.4	99.0	280	5	100	119	4
s344	100	38	5.5	100	125	5	100	48	4
s349	100	45	7.6	100	120	5	100	56	4
s382	100	1536	252	95.2	1633	1320	100	1028	195
s386	100	168	9.8	100	238	4	100	168	3
s400	100	1850	663	95.8	409	1200	100	1091	195
s444	100	1871	975	95.6	994	1992	100	1026	152
s510	100	210	14.6	99.8	733	7	100	584	7
s526	100	2637	1023	91.0	2037	1060	100	1457	207
s641	100	398	25.7	93.1	327	10200	100	134	15
s713	100	216	1001	93.1	315	10440	100	139	21
s820	99.88	542	46	100	1304	120	100	785	40
s953	100	436	20.2	100	1050	29	100	578	40
s1196	100	469	95	98.7	545	4080	100	376	41
s1238	100	389	78	99.0	576	3600	100	389	52
s1423	67.39	1232	1565	56.4	4026	10800	-	-	-
s1494	100	598	238	100	1374	147	100	1040	103
s5378	92.66	515	3761	99.3	1037	12000	-	-	-

STEED employs a random vector test generation technique to detect some of the easily testable faults. The random seeds have effects on the experimental results. GLOBALTEST uses a two-step test generation method: fault-independent test generation and fault-oriented test generation, so the results of GLOBALTEST do not depend on the choices of random seeds. From Table 8.3, for most sequential circuits, GLOBALTEST obtains higher fault coverages than those of STEED with more compact test sets and less CPU time.

VERITAS can get good results in most small-to-middle size circuits. However, it fails to detect faults in large size circuits in ISCAS'89 benchmarks. The proposed test generation system GLOBALTEST can perform test generation on large size sequential circuits.

From Tables 8.2 to 8.3, the performance of our algorithm can be evaluated as follows:

- The proposed algorithm GLOBALTEST outperforms the ATPG system STEED in terms of time, fault coverage, and test length for most ISCAS'89 benchmark circuits.

- The global test generation system GLOBALTEST can perform test generation on larger sequential circuits while the system VERITAS fails.

- With the increase of the circuit size and the number of flip-flops, the algorithm GLOBALTEST provides an efficient performance. The proposed test generation system has successfully generated tests for sequential circuits with a reasonable amount of CPU time and has obtained close to maximum fault coverage.

- For some large circuits, when complete covers cannot be enumerated, the partial cover is generated and the algorithm can work on it.

- The proposed fault-independent test generation algorithm can be used as a front-end algorithm to efficiently perform test generation for sequential circuits.
- The proposed fault-oriented algorithm. based on the Boolean satisfiability and implication graph. is useful as a deterministic algorithm for sequential circuit test generation.

8.3 Experimental Results of Test Compaction

The test compaction algorithm for sequential circuits HICOMPACT was implemented in *C* language. The ISCAS'89 benchmark circuits were used to evaluate the performance of the HICOMPACT algorithm. It is assumed that a vector of all zeros was the reset state for the ISCAS'89 sequential benchmark circuits. We used the proposed global test generation algorithm GLOBALTEST for sequential circuits as the test generator. All experiments were conducted on a SUN Sparc 20 workstation.

The experimental results of the HICOMPACT algorithm with the ISCAS'89 benchmark circuits are shown in Table 8.4. For each sequential circuit. the total number of faults (*total faults*). the number of detected faults (*det. fault*). and the number of redundant faults (*red. fault*) are given. The fault coverage (*coverage*) only includes the detected faults and the test efficiency (*efficiency*) includes the detected faults and redundant faults. The compacted test vector length and the test compaction time are given in the columns *test vec.* and *compact time*.

From Table 8.4, it is shown that as the circuit size and number of flip-flops increase. our algorithm still shows efficient performance. It has successfully compacted test vectors for sequential circuits within a small amount of CPU time.

The performance of HICOMPACT is compared with several other test generation

Table 8.4. Real execution performance of our algorithm HICOMPACT on a SUN Sparc 20 with the ISCAS'89 sequential benchmark circuits.

circuit	total faults	det. fault	red. fault	coverage (%)	efficiency (%)	test vec.	compact time (s)
s298	308	273	35	88.64	100	99	0.23
s344	342	337	5	98.54	100	38	0.21
s349	350	343	7	98.00	100	45	0.34
s382	399	379	20	94.99	100	1536	0.23
s386	384	314	70	81.77	100	168	0.03
s400	424	397	27	93.63	100	1850	0.22
s444	474	439	35	92.62	100	1871	0.05
s510	564	564	0	100	100	210	0.31
s526	555	466	89	83.96	100	2637	0.16
s641	467	408	59	87.37	100	398	0.55
s713	581	480	101	82.62	100	216	0.64
s820	850	814	35	95.76	99.88	542	8.11
s953	1079	1069	10	99.07	100	436	0.94
s1196	1242	1239	3	99.76	100	469	5.45
s1238	1355	1283	72	94.69	100	389	1.85
s1423	1515	1007	14	66.47	67.39	1232	0.59
s1494	1506	1455	51	96.61	100	598	6.39
s5378	4603	3375	915	73.32	92.66	515	16.5

procedures in Table 8.5. The fault coverage (*coverage*) and the total number of test vectors (*test vec.*) are compared between these systems. The fault coverage only includes the detected faults. The lengths of test vectors generated by HICOMPACT are shorter that those of STEED and VERITAS for the most benchmarks. For example. for the circuit s510. the reduction in test vector length is over two times compared to STEED and VERITAS [18. 12].

It is obvious that the results of HICOMPACT on ISCAS'89 sequential benchmark circuits depend on the test set generated by the test generator. To exactly evaluate HICOMPACT. some random instances of the set covering problem are generated. The execution performance of HICOMPACT on these random instances of the set covering problem is shown in Table 8.6. For each problem instance. the number of sequences. the number of faults, and density are given in columns *no. of sequence. no. of faults.* and *density*, respectively. The density of the set covering problem is defined as

$$density = \frac{\text{total number of sequences for all faults}}{\text{no. of sequences} \times \text{no. of faults}}.$$

The minimum and maximum lengths of the sequences are given in column *min/max length*. The cost is the total number of test vectors needed to detect the faults. The optimal cost is obtained by an exhaustive search algorithm for the set covering problem. The exhaustive search algorithm tries all possible combinations of sequences to obtain the optimal solution. Therefore. it takes quite a long time to get the solution. For example. for the instance of 20 sequences. 20 faults. and 50% density. it takes about 4 hours to get the optimal solution on a SUN Sparc 20 workstation. A greedy search algorithm. similar to that in [14], is used to give *greedy cost*. The cost and CPU time by HICOMPACT are given in columns *HICOMPACT cost* and *time*. All these problem instances can be solved in less than 0.001 seconds in a SUN Sparc 20

Table 8.5. Test generation comparison with other procedures on the ISCAS'89 sequential benchmark circuits.

circuit	HICOMPACT		STEED [18]		VERITAS [12]	
	coverage (%)	test vec.	coverage (%)	test vec.	coverage (%)	test vec.
s298	88.64	99	87.66	280	88.64	119
s344	98.54	38	98.5	125	98.54	48
s349	98.00	45	97.9	120	97.89	56
s382	94.99	1536	90.73	1633	94.99	1028
s386	81.77	168	81.8	238	81.77	168
s400	93.63	1850	89.86	409	93.63	1091
s444	92.62	1871	88.39	994	92.62	1026
s510	100	210	99.82	733	100	584
s526	83.96	2637	76.58	2037	83.96	1457
s641	87.37	398	86.5	327	87.37	134
s713	82.62	216	81.58	315	82.74	139
s820	95.65	542	95.83	1304	95.88	785
s953	99.07	436	99.1	1050	99.07	578
s1196	99.76	469	98.71	545	99.76	376
s1238	94.69	389	93.96	576	94.69	389
s1423	66.47	1232	56.43	4026	-	-
s1494	96.61	598	96.62	1374	96.61	1040
s5378	73.32	515	69.00	1037	-	-

Table 8.6. Real execution performance of HICOMPACT on a SUN Sparc 20 with randomly generated set covering problem instances. The cost is the total number of test vectors needed to detect the faults.

no. of	no. of	density	min/max	optimal	greedy	HICOMPACT	
sequences	faults	(%)	length	cost	cost	cost	time(s)
10	10	25	1/20	16	16	16	<0.001
10	10	50	1/20	6	6	6	<0.001
10	10	75	1/20	5	11	5	<0.001
15	15	25	1/20	18	21	20	<0.001
15	15	50	1/20	8	9	8	<0.001
15	15	75	1/20	5	11	7	<0.001
20	20	25	1/20	23	23	23	<0.001
20	20	50	1/20	5	5	5	<0.001
20	20	75	1/20	2	6	2	<0.001

workstation.

It is shown that, in Table 8.6, the greedy search procedure can only find good solutions in some instances. HICOMPACT improves the results of the greedy search procedure. In most problem instances, HICOMPACT can find the optimal or near optimal solutions.

8.4 Summary

ISCAS'89 sequential benchmark circuits are used to evaluate the proposed test generation algorithm GLOBALTEST and the test compaction algorithm HICOMPACT. Two algorithms are capable of finding satisfactory solutions for ISCAS'89 benchmarks. Experimental results show that faults that require long test sequences are handled efficiently and finite state machines with a large number of states are tested using a reasonable amount of CPU time. For test generation, the performance is achieved because of a global test generation algorithm. For test compaction, the performance is achieved because of an efficient set covering algorithm and an efficient local multi-variable search algorithm.

CHAPTER 9

CONCLUSIONS AND FUTURE WORK

With the rapid advances in integrated circuit technology. it is possible to fabricate digital circuits with a very large number of devices on a single VLSI chip. The increase in size and complexity of circuits placed on a chip. with little or no increase in the number of input/output (I/O) pins. drastically reduces the controllability and observability of the logic on the chip. More logic must be accessed with almost the same number of I/O pins. thus making testing of chips much more difficult. Since VLSI engineering has been used in a wide range of applications. the need for testing is becoming more and more important.

Test generation for sequential circuits has been recognized as a difficult problem. Different approaches have been used to deal with the testing problem. either by randomly generating test sequences or by using deterministic test generation methods.

The existing test generation algorithms for sequential circuits can generate test sequences for large sequential circuits. However, with increasing circuit complexity, either test generation time increases exponentially or it cannot produce test sequences due to the exponential increase of reachable states.

In this dissertation, we propose a new approach for designing test generation algorithms with better time complexity and fault coverage. A global search approach has been developed for the test generation of large sequential circuits. The approach justifies a fault at a primary output or next state line in order to trace different sensitive pathes from the primary inputs and present state lines to the primary output or next state line. During the global test generation process. many faults are considered as candidates to be tested simultaneously. It consists of two algorithms: a fault-independent test generation algorithm and a fault-oriented test generation algorithm.

A new and efficient backward assignment method is presented to perform cover extraction by partitioning circuits. It can extract both the ON/OFF sets of the primary outputs and next state lines and the combinational excitation vectors. Thus it makes the conventional three-phase ATPG system into the exact three-phase. i.e.. extraction of cover sets and combinational excitation vectors. state justification. and state differentiation.

After a combinational excitation vector is obtained. state justification is used to find a justification sequence from the reset state to the excitation state. If the effect of the fault is propagated only to the next state lines. state differentiation is needed to continuously propagate the fault-effect to the primary outputs. To enhance the efficiency of state differentiation. a backward deterministic algorithm for state differentiation is developed and the order of choosing the next state lines in state differentiation is studied.

Since the fault-independent test generation algorithm cannot determine the redundant fault. a fault-oriented test generation algorithm is used to detect the remaining faults and determine the redundancy of the faults.

The fault-oriented test generation algorithm considers one fault at a time. A Boolean satisfiability and implication graph method is extended to test generation for sequential circuits. By disabling all flip-flops in the sequential circuits. the sequential circuits become pseudo-combinational circuits. A combinational circuit test generation algorithm based on Boolean satisfiability and implication graph is presented to obtain combinational excitation vectors. Then state justification and state differentiation are used again to justify the excitation states and to continuously propagate the effect of the fault to the primary outputs.

Test compaction is an important part in an ATPG system. The test compaction problem is formulated as a set covering problem and a fast and efficient set covering algorithm is used to solve the test compaction problem. A local reduction and expansion algorithm for test compaction is then developed to further compact the test set.

The proposed test generation algorithm for sequential circuits was implemented and used on the ISCAS'89 sequential benchmark circuits. The results on large sequential circuits suggest that our algorithm outperforms other existing test generation algorithms. The overall test system has yielded a high fault coverage and provided time efficient procedures to generate tests for large size sequential circuits.

Our algorithm can efficiently perform test generation for sequential circuits. It has obtained close to the maximum fault coverage on the most ISCAS'89 sequential benchmark circuits. Consequently. as was pointed out in [9]. the parallelization of transitive closure computation. though not attempted in the present work, is easily possible. The proposed test generation system, we hope, can be developed into a parallel test generation system. Since the testing problem for VLSI engineering is getting more and more difficult with the advent of VLSI. parallelism of test generation systems may be a possible solution.

REFERENCES

[1] V. D. Agarwal, S. K. Jain, and D. M. Singer. Automation in design for testability. In *Proc. Custom Integrated Circuit Conf.*, pages 21–23. Rochester. NY. May 1984.

[2] V. D. Agrawal, K.-T. Cheng, and P. Agrawal. Contest: A concurrent test generator for sequential circuits. In *Proc. 25nd Design Automat. Conf.*. pages 84–89. June 1988.

[3] A. V. Aho. J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Reading, Addison-Wesley, MA. 1974.

[4] M. Aourid and B. Kaminska. Neural networks for the set covering problem: An application to the test vector compaction. In *IEEE International Conference on Neural Networks*, pages 4645–4649. 1994.

[5] N. BenHamida, B. Kaminska, and Y. Savaria. Pseudo-random vector compaction for sequential testability. In *IEEE International Symposium on Circuits and Systems*. pages 63–66, 1994.

[6] R. K. Brayton, G. D. Hachtel, Curt McMullen. and A. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis.* Kluwer Academic. Hingham. MA, 1984.

[7] M. A. Breuer and A. D. Friedman. *Diagnosis and Reliable Design of Digital Systems.* Computer Science, New York, 1976.

[8] F. Brglez, D. Bryan, and Kozminski. Combinational profiles of sequential benchmark circuits. In *Proc. IEEE Int. Symp. Circuits and Systems.*. pages 1929–1934. May 1989.

[9] S. T. Chakradhar, V. D. Agrawal, and S. G. Rothweiler. A transitive closure algorithm for test generation. *IEEE Trans. on CAD.* 12(7):1015–1027. July 1993.

[10] S.T. Chakradhar and A. Raghunathan. Bottleneck removal algorithm for dynamic compaction and test cycles reduction. In *Proceedings European Design Automation Conference.* pages 98–104, 1995.

[11] X. Chen and M.L. Bushnell. Sequential circuit test generation using dynamic justification equivalence. *Journal of Electrical Testing: Theory and Application.* 8(1):9–33, Feb. 1996.

[12] H. Cho, G. D. Hachtel, and F. Somenzi. Redundancy identification/removal and test generation for sequential circuits using implicit state enumeration. *IEEE Trans. on CAD,* 12(7):935–945, July 1993.

[13] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third ACM Symposium on Theory of Computing.* pages 151–158. 1971.

[14] T.H. Cormen, C.E. Leiserson. and R.L. Rivest. *Introduction to Algorithms.* The MIT Press. Cambridge. Massachusetts. 1990.

[15] H. Cox and J. Raajski. On necessary and nonconflicting assignments in algorithmic test pattern generation. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems,* 13(4):515–530. Apr. 1994.

[16] H. Fujiwara and T. Shimono. On the acceleration of test generation algorithms. *IEEE Trans. Comp.,* C-32:1137–1144, Dec. 1983.

[17] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. freeman, 1979.

[18] A. Ghosh, S. Devadas. and A. R. Newton. Test generation and verification for highly sequential circuits. *IEEE Trans. on CAD,* 10(5):652–667, May 1991.

[19] P. Goel. An implicit enumeration algorithm to generate tests for combinational logic circuits. *IEEE Trans. Comp.*, C-30:215–222, Mar. 1981.

[20] J. Gu. Local search for satisfiability (SAT) problem. *IEEE Trans. on Systems. Man, and Cybernetics*, 23(4):1108–1129, Jul. 1993. and 24(4):709. Apr. 1994.

[21] J. Gu. Global optimization for satisfiability (SAT) problem. *IEEE Trans. on Knowledge and Data Engineering*, 6(3):361–381, Jun. 1994.

[22] J. Gu. *Multispace Search: A New Optimization Approach (Summary).* In *Lecture Notes in Computer Science, Vol. 834*, pages 252–260. 1994.

[23] J. Gu. Optimization by multispace search. Technical Report UCECE-TR-90-001. Dept. of Electrical and Computer Engineering, Univ. of Calgary, Jan. 1990.

[24] J. Gu. *Design Efficient Local Search Algorithms. In* F. Belli and F.J. Radermacher. editors, *Lecture Notes in Artificial Intelligence, Vol. 604: IEA/AIE*, pages 651–654. Springer-Verlag, Berlin, Jun. 1992.

[25] J. Gu. *Constraint-Based Search.* Cambridge University Press, New York. to appear.

[26] J. Gu, Q.P. Gu. and D.-Z. Du. Convergence properties of optimization algorithms for the satisfiability (SAT)problem. *IEEE Trans. on Computers*, 45(2):209–219. Feb. 1996.

[27] M.S. Hsiao, E.M. Rudnick, and J.H. Patel. Alternating strategies for sequential circuit ATPG. In *European Design and Test Conference*. pages 368–374. 1996.

[28] O. H. Ibarra and S. K. Sahni. Polynomially complete fault detection problems. *IEEE Trans. on Comp.*, C-24:680, Mar. 1975.

[29] S. Kajihara and I. Pomeranz aand S.M. Reddy. On compacting test sets by addition and removal of test vectors. In *VLSI Test Symposium*, pages 202–207, Apr. 1994.

[30] S. Kajihara, I. Pomeranz, K. Kinoshita, and S.M. Reddy. Cost-effective generation of minimal test sets for stuck-at faults in combinational logic circuits. In *Proc. 30th Design Automat. Conf.*, pages 102–106, 1993.

[31] H. Kubo. A procedure for generating test sequences to detect sequential circuit failures. *NEC Res. and Dev.*, 12:69–78, Oct. 1968.

[32] T. Larrabee. Test pattern generation using Boolean satisfiability. *IEEE Trans. on CAD*, 11(1):4–15, Jan. 1992.

[33] H. K. Lee and D. S. Ha. Hope: An efficient parallel fault simulator for synchronous sequential circuits. In *29th ACM/IEEE Design Automation Conference*, pages 336–340, 1992.

[34] S.Y. Lee and K.K. Saluja. Efficient test vectors for ISCAS sequential benchmark circuits. In *IEEE International Symposium on Circuits and Systems*, pages 1511–1514, Chicago, IL, 1993.

[35] H. T. Ma, S. Devadas, A. R. Newton, and A. Sangiovanni-Vincentelli. Test generation for sequential circuits. *IEEE Trans. on CAD*, 7(10):1081–1093, Oct. 1988.

[36] S. Mallela and S. Wu. A sequential test generation system. In *Proc. Int. Test Conf.*, pages 57–61, Philadelphia, PA, Oct. 1985.

[37] T.E. Marchok, A. El-Maaleh, W. Maly, and J. Rajski. A complexity analysis of sequential ATPG. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 15(11):1409–1423, Nov. 1996.

[38] J. Markus. *Electronics Dictionary.* McGraw-Hill, New York, 1978.

[39] A. Miczo. The sequential ATPG: A theoretical limit. In *Proc. Int. Test Conf.*, pages 143–147, Oct. 1983.

[40] A. Miczo. *Digital Logic Testing and Simulation.* Harper and Row, Publishers, New York, 1986.

[41] P. Muth. A nine-valued circuit model for test generation. *IEEE Trans. Computers*, C-25:630–636, June 1976.

[42] T. Niermann and J.H. Patel. HITEC: A test generation package for sequential circuits. In *European Design Automation Conference*, pages 214–218, 1991.

[43] T.M. Niermann, R.K. Roy, J.H. Patel, and J.A. Abraham. Test compaction for sequential circuits. *IEEE Trans. on CAD*, 11(2):260–267, Feb. 1992.

[44] I. Pomeranz, L.N. Reddy, and S.M. Reddy. COMPACTEST: A method to generate compact test sets for combinational circuits. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 12(7):1040–1049, July 1993.

[45] I. Pomeranz and S.M. Reddy. Classification of faults in synchronous sequential circuits. *IEEE Trans. on Computers*, 42(9):1066–1077, Sep. 1993.

[46] I. Pomeranz and S.M. Reddy. On generating compact test sequencefor synchronous sequential circuits. In *Proc. European Design Automation Conference*, pages 105–110, 1995.

[47] I. Pomeranz and S.M. Reddy. Dynamic test compaction for synchronous sequential circuits using statis compaction techniques. In *Proc. of Annual Symposium on Fault Tolerant Computing*, pages 53–61, Sendai, Japan, 1996.

[48] P. Prinetto, M. Rebaudengo, and M.S. Reorda. An automatic test pattern generator for large sequential circuits based on genetic algorithms. In *International Test Conference*, pages 240–249, 1994.

[49] A. Raghunathan and S.T. Chakradhar. Acceleration techniques for dynamic vector compaction. In *ICCAD*, pages 310–317, 1995.

[50] A. Ralston and E.D. Reilly. *Encyclopedia of Computer Science*. Van Nostrand Reinhold, New York, 1993.

[51] L.N. Reddy, I. Pomeranz, and S.M. Reddy. COMPACTEST-II: A method to generate compact two-pattern test sets for combinational logic circuits. In *IEEE International Conference on Computer-Aided Design*, pages 568–574. Los Alamitos. CA, 1992.

[52] L.N. Reddy, I. Pomeranz, and S.M. Reddy. ROTCO: A reverse order test compaction technique. In *Proc. Euro. ASIC*, pages 189–194. 1992.

[53] J. P. Roth. Diagnosis of automata failures, a calculus and a method. *IBM J. Res. Dev.*. 10:278–291. July 1966.

[54] E.M. Rudnick and J.H. Patel. Combining deterministic and genertic approaches for sequential circuits test generation. In *32nd Design Automation Conference*. pages 183–188. San Francisco. CA, June 12-16 1995.

[55] G. Russell and I.L. Sayers. *Advanced Simulation and Test Methodologies for VLSI Design*. Van Nostrand Reinhold (International), London, 1989.

[56] D.G. Saab, Y.G. Saab, and J.A. Abraham. Automatic test vector cultimation for sequential circuits using generatic algorithms. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 15(10):1278–1285, Oct. 1996.

[57] M.H. Schulz, E. Trischler, and T.M. Sarfert. SOCRATES: A highly efficient automatic test pattern generation system. *IEEE Trans. on CAD*, 7(1):126–136, Jan. 1988.

[58] F. F. Sellers, M. Y. Hsiao, and L. W. Bearnson. Analyzing errors with Boolean difference. *IEEE Trans. Computers*, C-17:676–683, July 1968.

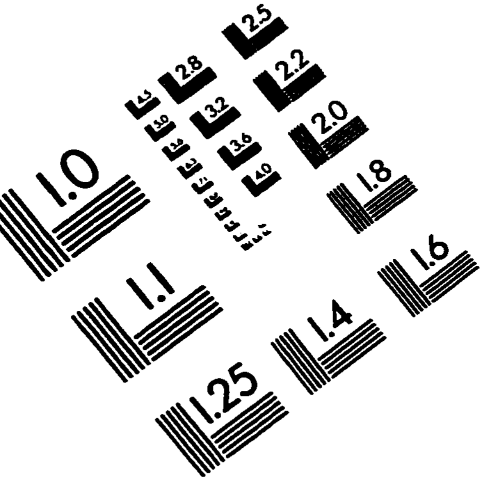
[59] E.M. Sentovich, K.J. Singh, C. Moon, H. Savoj, R.K. Brayton, and A. Sangiovanni-Vincentelli. Sequential circuit design using synthesis and optimization. In *Proc. Int. Conf. Computer Design*, pages 328–333, 1992.

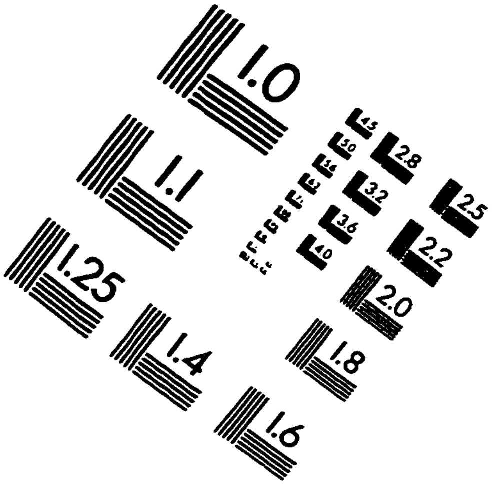
[60] S. Shteingart, A. W. Nagle, and J. Grason. Rtg: Automatic register level test generator. In *Proc. 22nd Design Automat. Conf.*, pages 803–807, Las Vegas, June 1985.

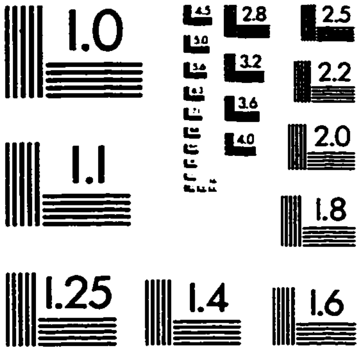
[61] J. J. Thomas. Automated diagnostic test program for digital networks. *Computer Design*, pages 63–67, Aug. 1971.

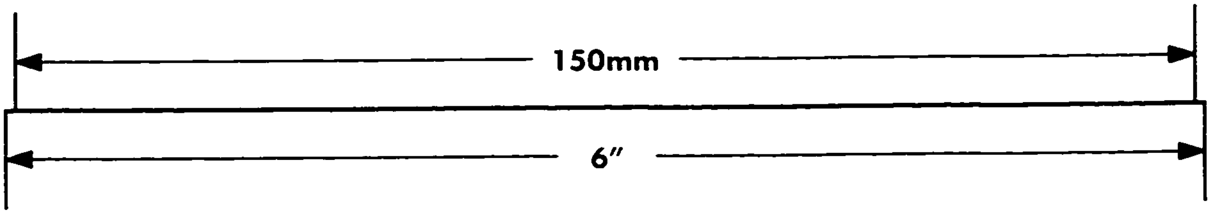
[62] A. S. Yousif. A novel search approach for test generation. Master's thesis, Dept. of Electrical and Computer Engineering, The University of Calgary, Calgary, AB T2N 1N4, September 1992.

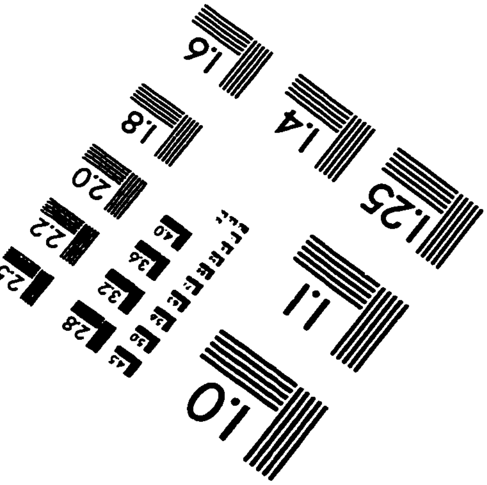
IMAGE EVALUATION
TEST TARGET (QA-3)

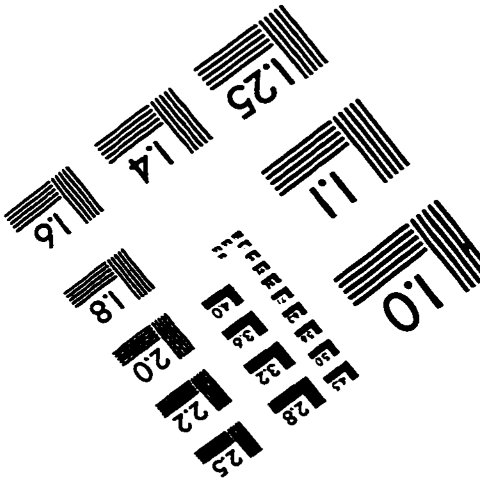












APPLIED IMAGE . Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989