ON DETECTION OF STUCK-OPEN FAULTS USING STUCK-AT TEST SETS IN CMOS COMBINATIONAL CIRCUITS

by

Hyung Ki Lee

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Electrical Engineering

APPROVED:

[illegible]

D. S. Ha, Chairman

[illegible]

J. G. Tront

[illegible]

S. F. Midkiff

March, 1989

Blacksburg, Virginia

6/5/89 MCR

ON DETECTION OF STUCK-OPEN FAULTS USING STUCK-AT TEST SETS IN CMOS COMBINATIONAL CIRCUITS

by

Hyung Ki Lee

D. S. Ha, Chairman

Electrical Engineering

(ABSTRACT)

The traditional line stuck-at fault model does not properly represent transistor stuck-open (SOP) faults in complementary metal oxide semiconductor (CMOS) circuits. In general, test generation methods for detecting CMOS SOP faults are complex and time consuming due to the sequential behavior of faulty circuits. The majority of integrated circuit manufacturers still rely on stuck-at test sets to test CMOS combinational circuits at the risk of some SOP faults not being detected.

In this thesis we investigate two aspects regarding the detection of SOP faults using stuck-at test sets. First, we measure the SOP fault coverage of stuck-at test sets for various CMOS combinational circuits. The SOP fault coverage is compared with that of random pattern test sets. Second, we propose a method to improve the SOP fault coverage of stuck-at test sets by organizing the test sequences of stuck-at test sets. The performance of the proposed method is compared with those of competing methods. Experimental results show that the proposed method leads to smaller test sets and shorter processing time while achieving high SOP fault coverage.

Acknowledgements

I would like to express my deepest gratitude to Dr. Dong S. Ha for his continuous encouragement and suggestions during my research for this thesis. I would also like to thank Dr. Joseph G. Tront and Dr. Scott F. Midkiff for serving on my committee.

I wish to express my appreciation to my parents in Korea. With much love, I would like to thank my wife and my daughter for their encouragement and patience.

The financial support provided by the National Science Foundation under Grant No. NSF-MIP-8809164 is also appreciated.

Table of Contents

List of Tables

List of Illustrations

I. Introduction

Complementary metal oxide semiconductor (CMOS) has become a dominant technology in very large scale integration (VLSI) circuits due to advantages such as low power consumption and high fabrication density. However, the testing of CMOS circuits is complex and time consuming. Conventional circuit testing schemes that use the gate level circuit representation and the line stuck-at fault model are known to be inadequate for testing CMOS circuits. A major difficulty in testing CMOS circuits stems from the inadequacy of the line stuck-at fault model. Transistor stuck-open (SOP) faults in which faulty transistors are turned off permanently are not modeled properly by the line stuck-at fault model in CMOS circuits [1]. A combinational circuit under the presence of SOP faults may behave as a sequential circuit. A sequence of two test patterns is required to detect a SOP fault [2-5].

Since Wadsack introduced the SOP fault model in the late 1970's [1], many test generation algorithms detecting SOP faults have been proposed [2-15]. CMOS test generation algorithms can be classified into two categories, switch level test generation algorithms [9-15] and gate level test generation algorithms [2-8]. In switch level test generation algorithms, CMOS circuits are represented in terms of switches and their interconnections. A gate level test generation algorithm attempts to take advantage of well established test generation schemes developed for the line stuck-at fault model. In general, gate level test generation algorithms are relatively simple, but give low fault coverage when compared to switch level test generation algorithms. Both types of algorithms, gate level algorithms and switch level algorithms, are still complex and time consuming, hence they may not be practical for large circuits.

The majority of integrated circuit (IC) manufacturers rely on test sets derived based on the single stuck-at fault model, called stuck-at test sets, to test CMOS circuits. There are three main reasons for using stuck-at test sets in testing CMOS circuits:

1. test generation algorithms for the detection of stuck-at faults are much simpler than those for the detection of SOP faults,
2. well established test generation algorithms developed for the detection of stuck-at faults, such as D-algorithm [16], can be used, and
3. conventional gate level circuit descriptions can be used directly for stuck-at test generation, so additional efforts to convert the circuit descriptions or to create other circuit descriptions, which are, in general, necessary for SOP fault test generation, can be avoided.

Though the use of stuck-at test sets has the advantages listed above, the cost for the advantages is low SOP fault coverage. Two important issues relate to the use of stuck-at test sets:

1. how good are stuck-at test sets for the detection of SOP faults, i.e., what is the SOP fault coverage of stuck-at test sets, and
2. how can we improve the SOP fault coverage of stuck-at test sets.

Although stuck-at test sets are widely used for testing CMOS circuits, to our knowledge, there has been only one report on the effectiveness of the method. Woodhall et al. presented empirical data on the SOP fault escaping rate of a stuck-at test set [17]. The observed SOP fault escaping rate for 4,522 die examined was 0.121%. However, the experiment does not give the SOP fault coverage of the stuck-at test set, i.e., it does not tell how many SOP faults are covered by the stuck-at test set. Moreover, the results are based on one type of combinational circuit implemented on many die.

In this thesis, we present the SOP fault coverage of stuck-at test sets for various CMOS combinational circuits. The SOP fault coverage of stuck-at test sets is compared with that of random pattern test sets. The next issue of this thesis is improvement of the SOP fault coverage of stuck-at test sets. Several methods of using stuck-at test sets in detecting SOP faults have been proposed [2-4]. In El-ziq's method [2], a stuck-at test set of the circuit under test is applied first to detect some SOP faults. Test patterns are generated for the remaining undetected SOP faults. Unlike the above method, Chandramouli proposed a method of organizing the test sequence of stuck-at test sets [4]. The method requires the application of a sequence of two or three stuck-at test patterns for each primary input or fan-out branch. El-ziq also proposed a method to organize the test sequence of a stuck-at test set [3]. The method sorts the stuck-at faults for all primary inputs and fan-out branches in a specific order and generates stuck-at test patterns to detect the faults in the given sequence. The method is essentially the same as Chandramouli's method except the consideration of CMOS complex cells. Both methods, El-Ziq's method [3] and Chandramouli's method [4], are based on the path sensitization scheme developed for detection of stuck-at faults. Unlike the above methods, we propose a method which considers the faults on the inputs of each gate. The performance of the proposed method is compared with those of competing methods. Experimental results show that the proposed method gives small test set sizes and short processing time while achieving high SOP fault coverage.

In Chapter II, background on testing CMOS circuits is presented with emphasis on the detection of SOP faults using stuck-at test sets. Chapter III presents the analysis of undetected SOP faults by the stuck-at test set and the proposed method. Chapter IV reports the experimental results and observations made from the experiments. Finally, Chapter V concludes this thesis.

II. Background

2.1 Overview

Since Wadsack introduced the SOP fault model for CMOS circuits in the late 1970's [1], there has been extensive research on testing CMOS circuits in which SOP faults are considered. In this chapter, the background of the proposed research is described with emphasis on the detection of SOP faults using stuck-at test sets. Section 2.2 briefly presents conventional circuit testing methods. Section 2.3 and Section 2.4 review previous studies on CMOS circuit testing and the basic scheme of SOP fault testing, respectively. Section 2.5 describes the equivalence relation between SOP faults and stuck-at faults. Section 2.6 presents SOP fault testing using stuck-at test sets. Finally, Section 2.7 describes the research of this thesis.

2.2 Review of Conventional Circuit Testing

Traditionally, digital circuits have been implemented using basic logic gates such as NAND, AND, NOR, OR and NOT. The circuits are represented by the basic logic gates and their interconnections. The line stuck-at fault model has been commonly used for testing the circuits. Under the line stuck-at fault model, it is assumed that faults occur on the lines of the gate inputs or the gate outputs. The stuck-at 1 (s-a-1) fault on line i implies that the line i is stuck permanently at logic value 1. The stuck-at 0 (s-a-0) fault on line i implies that the line i is stuck permanently at logic value 0.

During the last two decades, various test generation algorithms for detecting line stuck-at faults have been proposed. For example, the D-algorithm [16], PODEM [18] and FAN [19] are such algorithms. Through experiences in testing digital circuits, these algorithms and the line stuck-at fault model have been verified to be effective in testing conventional logic circuits using as TTL and nMOS technology.

2.3 Review of CMOS Circuit Testing

The conventional circuit testing methods based on the line stuck-at fault model are no longer adequate to test CMOS circuits [1-15,20-26]. The inadequacy of the conventional testing methods in testing CMOS circuits is mainly due to:

1. inaccurate circuit modeling and
2. inaccurate fault modeling.

CMOS complex gates and transmission gates are not properly represented using basic logic gates. The stuck-at fault model does not adequately represent physical failures of CMOS circuits either. A fault where a transistor is permanently on (off), called the transistor stuck-on (open) fault, is not modeled properly using the stuck-at fault model.

Several researchers studied transistor stuck-on (SON) faults [20-23]. A transistor SON fault may cause an intermediate voltage between Vdd (logic 1) and Vss (logic 0) at the faulty gate output node. In general, SON faults cannot be detected by a logical test, which examines only logic values, alone. When a SON fault exists in a CMOS gate, the voltage at the gate output node depends on the relative values of resistances from the output node to Vdd and ground (Vss). Thus, the output may be interpreted as the correct logic value. Detecting SON faults may require monitoring the static supply currents of the circuit under test [22]. Under fault-free conditions, a static CMOS circuit consumes only very small currents except during switching. When SON faults exist in the circuit, a high conductance path from Vdd to ground can be created through the faulty transistor causing large current consumption. Since SON faults are not detected by a logic test alone, we do not consider the detection of SON faults in this thesis.

Since Wadsack introduced the SOP fault model, many test generation methods have been developed to detect such faults. Two methods, gate level algorithms and switch level algorithms have been proposed. Gate level algorithms attempt to use well established stuck-at test generation techniques to test SOP faults [2-8]. Switch level algorithms use switch level descriptions to represent the circuits and the faults [9-15].

Early approaches for testing CMOS circuits used stuck-at test sets to detect SOP faults. These approaches are based on the gate level representation of the circuit which consist of only primitive logic gates and/or CMOS complex cells. Chandramouli showed that all single SOP faults in these kinds of circuits can be detected by organizing the sequence of the test patterns of a stuck-at test set which covers all single stuck-at faults assuming zero gate delays [4]. This method is described in Section 2.6 in detail. El-Ziq proposed an algorithm to detect SOP faults [2]. The first part of the algorithm generates a stuck-at test set for the circuit. The stuck-at test set is applied to detect some SOP faults. The second part of the algorithm generates test patterns for the undetected SOP faults. He also proposed a method of organizing the test sequence of a stuck-at test set to detect SOP faults [3]. The method sorts the stuck-at faults in proper order and generates test patterns to detect the faults in the order. The method is essentially the same as that of Chandramouli's method [4] except considering of CMOS complex cells. Unlike the above methods, Jain and Agrawal proposed a procedure to generate SOP tests for general CMOS circuits including transmission gates [5]. The key idea is to convert a CMOS circuit with SOP faults into an equivalent gate level circuit with stuck-at faults. Then a conventional stuck-at test generation algorithm, the D-algorithm, with slight modification, is applied to find a test set. The advantage of this method is that it is simple to apply and it can use well established stuck-at test generation algorithms to test SOP faults. However, the size of the equivalent gate level circuit is usually far larger than that of the original circuit. Moreover, the equivalent circuit has memory elements to make the test generation procedure complex. All of the methods mentioned above assume that all gates in the circuit have zero delay. When the circuit has different gate delays and/or different timing skew on the circuit input lines, a pair of test patterns which is supposed to detect a SOP fault can be invalidated [27]. To address this problem, Reddy et al. proposed a procedure to generate robust tests which are not invalidated by gate delays and/or input timing skews [6]. The D-algorithm was used to derive the robust tests of circuits represented at the gate level. In general, the test generation of robust test patterns is complex and time consuming. Moreover, some SOP faults may not be detected due to a lack of robust test patterns.

All of the above algorithms are gate level algorithms. Gate level algorithms suffer low fault coverage for SOP faults due to inaccurate circuit modeling and inaccurate fault modeling. To address the problems, switch level algorithms have been proposed. In the following, we discuss switch level algorithms.

Chiang et al. proposed a test generation method using a graph model to represent circuits [9-10]. Transistor networks are modeled as a connection graph, where each transistor is represented by an edge with a logic variable as its label. When the edge label equals logic

1 (0), the corresponding transistor is on (off). SOP (SON) faults are represented by assigning edge labels of faulty transistors to logic 0 (1) permanently. Once the circuit and faults are represented in this way, test patterns are generated by traversing a path in the connection graph or by analyzing the path and/or the cutset expressio.ıs driven from the graph representations. Unlike the above method, several researchers proposed switch level test generation algorithms based on the D-algorithm [11-13]. The D-algorithm which was originally developed to detect line stuck-at faults is extended to switch level networks to generate tests detecting transistor faults. Chen et al. used the PODEM algorithm to generate tests for switch level networks [14]. The method has no restrictions on the type of circuits.

In the above algorithms, only single SOP faults are considered. Rajski attempted to detect multiple SOP faults in CMOS circuits [7-8]. The test set is based on the application of sequences of three adjacent input vectors called trios. The test generation method is based on path tracing which is similar to critical path tracing developed for stuck-at faults. Jha studied the detection of multiple transistor faults including SON faults and SOP faults [24-25]. He showed that a test set which detects single SOP faults in a CMOS complex gate also detects most of the multiple faults of the gate.

In summary, switch level test generation algorithms give higher fault coverage than gate level test generation algorithms. However, they are more complex and time consuming.

2.4 CMOS SOP Fault Testing

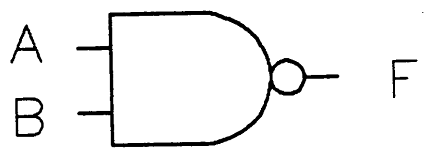
In the above section, previous studies on CMOS testing were briefly reviewed. In this section, the basic scheme for detecting SOP faults is discussed.

A SOP fault in a CMOS combinational circuit turns the faulty circuit into a sequential circuit [1]. A sequence of two test patterns, say T_1 and T_2, is required to detect a SOP fault [2-5]. T_1 is used for the initialization of the faulty gate output and T_2 is used for the detection of the fault. T_2 is an input pattern such that the gate output becomes floating from Vdd and ground under the presence of the SOP fault. T_1 is chosen to set the faulty gate output to logic value S, where $\bar{S}$ is the fault-free value of the faulty gate output under the application of T_2. When a T_2 pattern is applied to the faulty circuit, the faulty gate output is floating. Hence, the previous gate output value S is maintained due to the electric charges shared by the parasitic capacitance at the faulty gate output node. The SOP fault is detected provided that a sensitizing path exists from the faulty gate output to a primary output of the circuit.

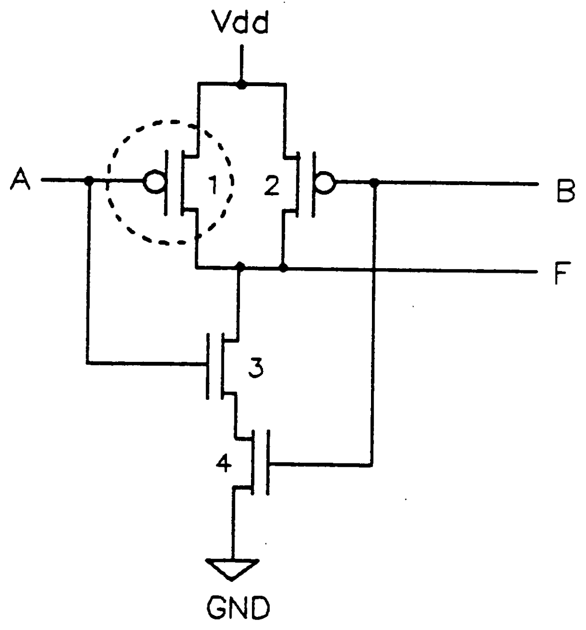
For example, consider the 2-input CMOS NAND gate shown in Figure 1. Suppose that the p-type transistor encircled by the broken line is stuck-open. Suppose that we apply the input pattern AB = 11 followed by 01 to the faulty circuit. When the second pattern AB = 01 is applied, the output F becomes floating and retains the previous value 0 generated by AB = 11. Since the faulty free output is 1 under application of the second pattern AB = 01, the fault is detected. In this case, T_1 is 11 and T_2 is 01. Table 1 shows the truth table of fault free and faulty 2-input NAND gates. In the table, F is the gate output under the fault free condition and F_i is the gate output under the transistor-i SOP fault. M denotes the floating output, i.e., the memory state of the faulty gate. Table 2 shows all the possible tests for the SOP faults identified in Table 1.

Two faults, say α and β, are called equivalent if and only if any test detecting fault α always detects fault β and vice versa. In the above example, the SOP fault at transistor-3 is equivalent to the SOP fault at transistor-4. In general, SOP faults occurring at the same type of transistors connected in series from the gate output node to Vdd or ground are equivalent [3]. Hence, n+1 distinct SOP faults exist for an n-input NAND (NOR) gate. Similarly, for an n-input AND (OR) gate implemented using an n-input NAND (NOR) gate and an inverter, it can be seen that the SOP fault at the p (n) type transistor of the inverter is equivalent to a SOP fault at any n (p) type transistor of the NAND (NOR) gate. Hence, n+2 distinct SOP faults exist for an n-input AND or OR gate.

As shown in the above, SOP faults in CMOS combinational circuits can be detected by the application of two test patterns, T_1 and T_2. However, the application of T_1 and T_2 does not



a) a 2-input NAND gate



b) CMOS implementation

Figure 1. A CMOS 2-input NAND gate

Table 1. Truth Table of a 2-Input NAND Gate with SOP Faults

A B	F	F_1	F_2	F_3	F_4
0 0	1	1	1	1	1
0 1	1	M	1	1	1
1 0	1	1	M	1	1
1 1	0	0	0	M	M

Table 2. Tests for a 2-input NAND Gate

faults	T_1	T_2	output fault-free/faulty
F_1	(11)	(01)	1/0
F_2	(11)	(10)	1/0
F_3	(00),(01),(10)	(11)	0/1
F_4	(00),(01),(10)	(11)	0/1

guarantee detection of the SOP fault when arbitrary gate delays are considered. The problem is that spurious values due to hazards occurring during the transition from T_1 and T_2 may de-initialize the faulty gate output node [27]. Hence, two pattern tests designed to detect a SOP fault assuming zero gate delays may not detect the fault under the consideration of gate delays. In CMOS circuits, it is known that charges shared by two transistors may also cause the invalidation of the test sequence [28]. As discussed in the previous section, robust test derivation algorithms try to find a pair of tests T_1 and T_2 which are not invalidated through arbitrary gate delays. In general, the methods are complex and time consuming. Moreover, there may be some SOP faults for which robust test patterns do not exist.

2.5 Fault Equivalence Between SOP Faults and Stuck-At Faults

In this section, we investigate the fault equivalence relation between a SOP fault and a stuck-at fault for primitive logic gates such as NAND, NOR, AND, OR and inverter. Consider an n-input NAND gate of a circuit. Suppose that the p-type transistor connected to input i of the NAND gate is stuck-open. Let us call this fault α . Suppose that the faulty gate output is properly initialized to logic 0. In order to detect α, gate input i should be 0 and the other inputs of the gate should be 1. Let us consider the s-a-1 fault on input i of the gate, say fault β. Obviously, the condition detecting α is the same as the one required to detect β. Hence, any test detecting α also detects β and vice versa. This means that α is equivalent to β provided that the faulty gate output is properly initialized. We say that α is potentially equivalent to β in this thesis. To detect an n-type transistor SOP fault of a NAND gate, all gate inputs should be 1. It is the same condition required to detect the s-a-0 fault on any input of the gate. Hence, an n-type transistor SOP fault of a NAND gate is potentially equivalent to an input line s-a-0 fault of the gate. Similarly, an n-type (p-type) transistor SOP fault of a NOR gate is potentially equivalent to the input s-a-0 (s-a-1) fault. For the case of an inverter, the p-type (n-type) transistor SOP fault is potentially equivalent to the input s-a-1 (s-a-0) fault. Since an AND (OR) gate is implemented using a NAND (NOR) gate and an inverter, there is a potentially equivalent stuck-at fault for any SOP fault of the gate. From the above discussion, we conclude that there exists a potentially equivalent stuck-at fault for any SOP fault of a CMOS combinational circuit consisting of only primitive logic gates.

Let us consider an n-input NAND gate again. SOP faults occurred on the n-type transistors connected in series to ground are equivalent. Hence, there are (n+1) distinct SOP faults, n p-type and one n-type transistor SOP faults, for the n-input NAND gate. The potentially equivalent stuck-at faults of the (n+1) distinct SOP faults are n s-a-1 faults on n inputs and the s-a-0 fault on any input of the gate. In this paper, we call these stuck-at faults as the primary faults of the NAND gate. Similarly, the primary faults of an n-input NOR gate are the n s-a-0 faults and one s-a-1 fault on the inputs. The primary faults of an inverter are the s-a-1 and the s-a-0 faults on its input. The primary faults of an AND (OR) gate are identical to those of a NAND (NOR) gate. Clearly, there is a one-to-one correspondence between primary faults and SOP faults of a gate after the removal of equivalent SOP faults. It is also known that a test set detecting all the primary faults of a gate also detects all the stuck-at faults of the gate [29].

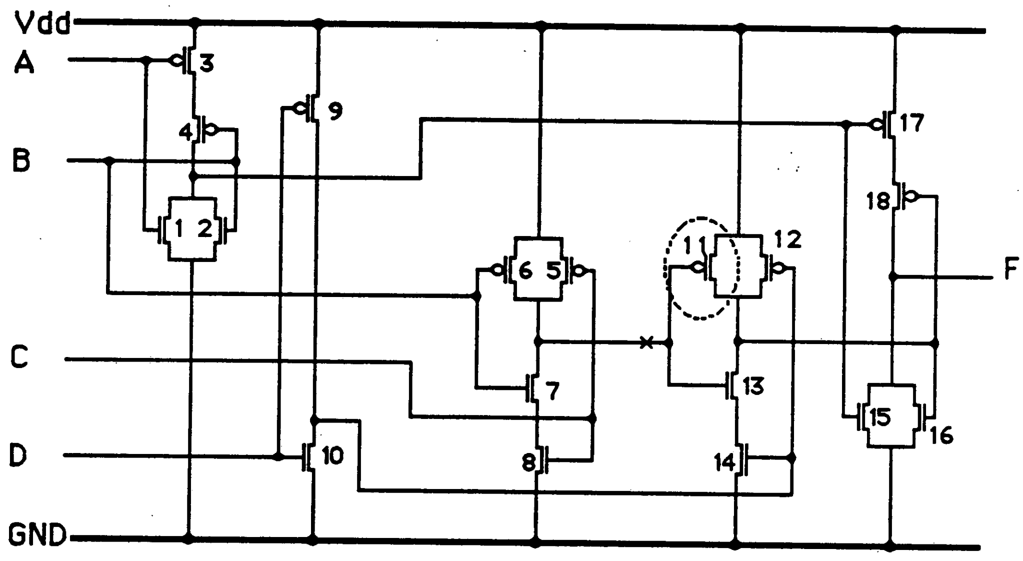
2.6 Detection of SOP Faults Using Stuck-At Faults

In Section 2.3, various test derivation algorithms for detecting CMOS SOP faults were discussed. As was shown, some of the early approaches attempted to use the stuck-at test set to test SOP faults [2-4]. In this section, previous studies on CMOS SOP fault testing using stuck-at test sets are described in detail since it is the essential part of this thesis.

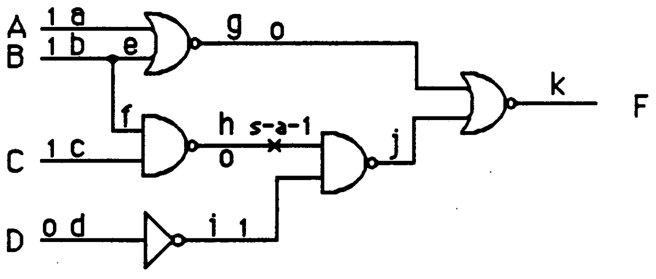
Consider a CMOS circuit consisting of only primitive logic gates. As shown in Section 2.5, there exists an equivalent stuck-at fault for any given SOP fault provided that the faulty gate output is properly initialized. This implies that at least one T_2 pattern for any SOP fault is included in the stuck-at test set which covers all stuck-at faults. Since the faulty gate output should be initialized to logic 0 or 1, the test detecting the gate output s-a-1 or s-a-0 fault is a T_1 pattern for the SOP fault. Hence, at least one T_1 pattern and one T_2 pattern for any SOP fault are included in the stuck-at test set. This implies that if the sequence of the test patterns of a stuck-at test set detecting all the stuck-at faults is properly organized, with possible repetitions of some patterns, they can detect all SOP faults under the assumption of zero gate delays.

For example, consider the circuit shown in Figure 2. Switch level and gate level descriptions of the circuit are shown in Figure 2 a) and Figure 2 b), respectively. A stuck-at test set detecting all stuck-at faults in the circuit is {1101, 0100, 1110, 1010, 0000}. The SOP fault encircled by the broken line is equivalent to the line 'h' s-a-1 fault provided that the gate output line 'j' is initialized to 0. The SOP fault is detected by the test pattern ABCD = 1110, which detects the line 'h' s-a-1 fault, after the application of an initialization pattern T_1 , for example ABCD = 1010. In fact, ABCD = 1010 is a test pattern which detects the line 'h' s-a-0 fault. However, a T_1 pattern need not be a test pattern detecting the line 'h' s-a-0 fault. In the example circuit, the test sequence (1110, 0100, 0000, 0100, 1101, 0100, 1110, 1010, 0000, 1010) detects all SOP faults of the circuit assuming zero gate delays.

Based on the above observation, El-ziq [3] and Chandramouli [4] suggested methods of organizing a sequence of test patterns for stuck-at faults to detect SOP faults. Both methods are based on the path sensitization scheme developed for detection of stuck-at faults. The methods organize the stuck-at test patterns in the order of detecting the s-a-0 fault followed by the s-a-1 fault, or vice versa, for every primary input and fan-out branch. Since the two methods are the same in the organization of the test patterns, we consider Chandramouli's method in this thesis. The method applies {(SZ),(SO)} or {(SZ),(SO),(SZ)} at every primary input which does not fan-out and at every fan-out branch, where (SZ) and (SO) denote the



a) Switch level description



b) Gate level description

Figure 2. A CMOS combinational circuit

s-a-0 test and the s-a-1 test of the input or the fan-out branch, respectively. {(SZ),(SO),(SZ)} is needed for only one input of a gate in which all the inputs are connected to primary inputs or fan-out branches. Otherwise, {(SZ),(SO)} is applied. The method is proved to be valid for NAND networks. However, we found that the method can not be directly applied to NOR networks. Suppose that an n-type transistor connected to input A of a 2-input NOR gate is stuck-open. B denotes the other input of the gate. For the SOP fault, the T_1 pattern is AB = 00, which is the same as the s-a-1 test for input A, and the T_2 pattern is AB = 10, which is the same as the s-a-0 test for input A. This implies that {(SO),(SZ)} instead of {(SZ),(SO)} should be applied to detect an n-type transistor SOP fault of a NOR gate. In general CMOS combinational circuits, {(SZ),(SO),(SZ)} should be applied to one input of each gate whose inputs are connected to primary inputs and/or fan-out branches. {(SZ),(SO)} should be applied to the other inputs of NAND or AND gates and {(SO),(SZ)} to NOR or OR gates. The above method assures the highest SOP fault coverage assuming zero gate delays. The method requires gate level stuck-at fault simulation to find (SZ)'s and (SO)'s for signal lines. In general, the test set sizes are large and the processing time is long due to fault simulations to be shown in Chapter IV.

Instead of above algorithmic approaches, one may be tempted to consider a simple heuristic of rearranging test patterns to maximize the output distance of two adjacent test patterns. The output distance of two patterns is the number of different bits in the fault-free output responses of the circuit under application of the two patterns. Once a test pattern is chosen, the next pattern is selected among the remaining test patterns to maximize the output distance of the two patterns. Since the heuristic requires only logic simulation, the processing time is far shorter than that of Chandramouli's method and that of our method to be proposed. It does not increase the size of a test set. However, our experimental results in Chapter IV show that this method suffers from low SOP fault coverage.

2.7 The Proposed Research

As discussed in the above section, stuck-at test sets can be used to detect SOP faults of a CMOS circuit. Assuming zero gate delays, a proper organization of the sequence of test patterns of a stuck-at test set, with possible repetitions of some patterns, can detect all SOP faults. However, when gate delays are considered, it does not guarantee the detection of all SOP faults. When arbitrary gate delays are considered, two important issues related to testing of CMOS circuits using stuck-at test sets are

1. the SOP fault coverage of stuck-at test sets, that is, how many SOP faults of the CMOS circuit under test are detected by a stuck-at test set, and
2. improvement of the SOP fault coverage of the stuck-at test sets.

The first item is important since many IC manufacturers rely on stuck-at test sets without organizing their test patterns to test CMOS circuits. In this thesis, we study the above two issues.

Although extensive research has been done on testing CMOS circuits, to our knowledge, there has been only one experimental report related to the SOP fault coverage of stuck-at test sets. Woodhall et al. reported empirical data on a purely combinational ASIC CMOS circuit implemented on many die [17]. They performed an experiment to measure the SOP fault escaping rate of a stuck-at test set. Among 4552 die examined, 44 die have one or more SOP faults. Out of 44 die with SOP faults, only 4 die escaped detection by the stuck-at test set which covers all stuck-at faults of the circuit. The observed SOP fault escaping rate was 0.121%. However, the experiment does not give the SOP fault coverage of the stuck-at test set, i.e., it does not tell how many SOP faults are covered by the stuck-at test set. Moreover, the results are based on one type of combinational circuit implemented on many die. In order to evaluate the SOP fault coverage of stuck-at test sets, we performed experiments on various CMOS combinational circuits. The SOP fault coverage of stuck-at test sets is compared with that of random pattern test sets.

The next issue of this thesis is improvement of the SOP fault coverage of stuck-at test sets. We propose a method of improving the SOP fault coverage through the organization of the test sequence of a given stuck-at test set. In general, a SOP fault under the application of a stuck-at test set may not be detected due to two reasons:

1. lack of the necessary test patterns in the stuck-at test set and/or
2. improper initialization of the faulty gate output.

Through experiments performed on various CMOS combinational circuits, we found that improper initialization is the major reason for SOP faults escaping detection. This suggests that proper rearrangement of the stuck-at test set may improve the SOP fault coverage substantially. The method proposed is to rearrange the test sequence of a given stuck-at test set to achieve proper initialization of the faulty gate output. The method attempts to obtain near minimal sizes of test sets while achieving high SOP fault coverage. The method is based on stuck-at fault simulation assuming zero gate delays. For simplicity, circuits consisting of only primitive logic gates are considered in this thesis. However, the method can be directly applied to CMOS combinational circuits consisting of complex gates.

III. Test Sequence Organization of Stuck-At Test Sets

3.1 Introduction

The two major objectives of this thesis related to the testing of SOP faults using stuck-at test sets are

1. the measurement of the SOP fault coverage of stuck-at test sets in CMOS combinational circuits and
2. the investigation of a method to improve the SOP fault coverage of stuck-at test sets.

Towards the objectives given above, we present the evaluation methods to measure the SOP fault coverage of various stuck-at test sets and the proposed method including its implementation in this chapter. Section 3.2 describes the analysis of undetected faults by the stuck-at test sets. Section 3.3 describes the proposed method and its implementation. Section 3.4

describes the implementation of Chandramouli's method. Finally, Section 3.5 describes the SOP fault simulator and its implementation.

3.2 Analysis of Undetected SOP Faults

As explained in Section 2.7, under application of a stuck-at test set, a SOP fault may not be detected due to two causes, the lack of necessary test patterns in the test set, or improper initialization of the faulty gate output. In this section, we quantify the contribution of each cause to understand the capability and the limit of using stuck-at test sets in detecting SOP faults.

Suppose that a test pattern t_i detects a SOP fault under proper initialization of the faulty gate output. The proper initialization is highly likely to be determined by the previous test pattern t_{i-1} and the transition from t_{i-1} to the current pattern t_i. Hence, we assume that the initialization of a gate is determined only by the previous pattern t_{i-1} and the current pattern t_i throughout this thesis.

Let us consider the possible highest SOP fault coverage that can be obtained by a stuck-at test set. Suppose that the stuck-at test set consists of n patterns enumerated by 1, 2, 3, ..., n. To achieve the possible highest SOP coverage, every pair of patterns (i,j), $i \neq j$, should be applied to the circuit at least once during the test. We call a test sequence containing every pair of (i,j), $i \neq j$, an Exhaustive Test Sequence (ETS).

Suppose that an ETS of a stuck-at test set fails to detect a SOP fault in the circuit under test. This is due to one of two cases:

1. a test pattern, say T_2, detecting the fault is not included in the stuck-at test set, or
2. there is a test pattern T_2 in the stuck-at test set, however a proper initialization pattern T_1 is not included in the test set. (It should be noted that such a pattern T_1 may not exist.)

In either case, the stuck-at test set lacks a necessary test pattern to detect the fault, i.e., the ETS can not detect the fault since the necessary pattern is not included in the test set. Clearly, the SOP fault coverage of a stuck-at test set (in which the test set is organized or not) is always less than or equal to that of an ETS of the stuck-at test set. Organization of a test set is an attempt to reduce the number of undetected faults due to improper initialization, i.e., T_1 and T_2 patterns of a fault are included in the test set, however they are not applied in the proper order to detect the fault.

We use the following notation to represent the escaping rate of SOP faults of a stuck-at test set.

E	Escaping rate of SOP faults (%)
ET	Escaping rate due to lack of test patterns (%)
EI	Escaping rate due to improper initialization (%)
nSOP	Number of SOP faults
nSAT	Number of SOP faults detected by the stuck-at test set
nETS	Number of SOP faults detected by an ETS

As a SOP fault is not detected due to either the lack of a test pattern or improper initialization, but clearly not by both,

$$E = ET + EI .$$

From the above discussion, the escaping rate due to lack of test patterns is

$$ET = \frac{nSOP - nETS}{nSOP} \times 100 \quad (\%).$$

Since

$$E = \frac{nSOP - nSAT}{nSOP} \times 100 \quad (\%),$$

the escaping rate due to improper initialization is

$$
\begin{aligned}
EI &= E - ET \\
&= \left(\frac{nSOP - nSAT}{nSOP} - \frac{nSOP - nETS}{nSOP}\right) \times 100 \quad (\%) \\
&= \frac{nETS - nSAT}{nSOP} \times 100 \quad (\%).
\end{aligned}
$$

Through the proper organization of the test patterns, EI for a stuck-at test set can be reduced to increase the SOP fault coverage of the stuck-at test set. However, ET cannot be reduced unless a different stuck-at test set is chosen. Hence, the SOP fault coverage of a stuck-at test set is bounded by (100-ET) %.

For the rest of this section, we discuss the minimal length ETS of a stuck-at test set. Suppose that a stuck-at test set consists of n patterns enumerated by 1, 2, ..., n. Let us consider the application of all the pairs of test patterns (i,j) such that $i<j$, in order. We ignore the application of the reverse order (j,i) for a moment. The number of pairs of (i,j), $i<j$, is ${}_nC_2$. As each pair has two patterns, i and j, the total number of patterns is

$$2 \times {}_nC_2 = \frac{2 \times n!}{(n-2)!2!} = n(n-1).$$

All n(n-1) patterns are shown in row 1 through row (n-1) of Table 3. Suppose that we apply the n(n-1) patterns in the order of left to right and top to bottom. As shown in the table, row i contains all (i,j) and (j,i), $j = i+1, i+2, ..., n$, pairs except (n,i) pairs. For example, the first three patterns of row 1 are (1, 2, 1). The first two patterns form the (1, 2) pair and the last two patterns form the (2, 1) pair. As the last pattern of each row is n and the first pattern of row i is i, all pairs (n,i) are also contained in the sequence except the pair (n,1). Row n is added in Table 3 to include the (n,1) pair. Hence, Table 3 gives a minimal length ETS of a stuck-at test set. From the above observation, we have the following theorem.

THEOREM: The minimal length of an ETS of a stuck-at test set with size n is n(n-1)+1.

Table 3. A Minimal Length Exhaustive Test Sequence

row	sequence	no. of patterns
1	1 2 1 3 1 4 ...... 1 n-1 1 n	2n-2
2	2 3 2 4 ...... 2 n-1 2 n	2n-4
3	3 4 ...... 3 n-1 3 n	2n-6
.	....	.
.	....	.
.	....	.
n-2	n-2 n-1 n-2 n	4
n-1	n-1 n	2
n	1	1

From the above theorem, the order of a minimal length ETS of a stuck-at test set is n^2. Hence it may not be practical to apply an ETS for a large stuck-at test set.

3.3 The Proposed Method

In this section, we describe our method to organize the sequence of test patterns of stuck-at test sets. Our approach is based on the detection of primary faults of individual gates rather than the faults on the primary inputs and fan-out branches which forms the basis for El-Ziq's [3] and Chandramouli's [4] methods. Suppose that a test pattern T_2 of a stuck-at test set detects a primary fault. In order to detect the corresponding SOP fault of the primary fault, the gate output should be initialized first. An initialization pattern of the test pattern is obtained in the following way. Let D ($\overline{D}$) denote that the fault free value of a gate under the application of a test pattern is 1 (0) and the faulty value is 0 (1). Suppose that a test pattern T_2 detects a primary fault of a gate. If the gate output is D ($\overline{D}$) under the application of T_2, then any pattern which produces logic 0 (1) at the gate output is an initialization pattern for T_2. Obviously, the test pattern detecting the s-a-1 (s-a-0) fault on the gate output can be one of the initialization patterns. In fact, the pattern detecting the s-a-1 (s-a-0) fault is used as an initialization pattern in El-Ziq's [3] and Chandramouli's [4] methods.

In the following, we present a procedure organizing the sequence of test patterns. The essence of the procedure is to apply the stuck-at test in the given sequence. Then a sequence of additional test patterns is obtained to cover any remaining faults. A gate level fault simulation assuming zero gate delays is used for this purpose. The procedure is divided into five steps. The first step is an initialization step. In the second step, the original test set is applied to the circuit in the given sequence and all faults detected by the test set are eliminated from further processing. In the third and fourth steps, the sequence of test patterns is organized to cover undetected faults in step 2. First, we construct a test table which contains all T_1 and T_2 patterns for each fault using fault simulation. Next, a sequence of test patterns is obtained by searching the test table. Finally, Step 5 reduces the test set size by eliminating unnecessary test patterns. For this purpose, we divide the test sequence into subsequences such that the first patterns of each subsequence do not detect any fault. Then, we apply the subsequences in the reverse order and eliminate all test patterns which do not detect new faults. The procedure is given below.

```
PROCEDURE TEST_SEQUENCE_ORGANIZER;

Step 1: { Initialization }
        Set up the fault list (FL) of all the primary faults.
        Set all the logic values to x (unknown).
Step 2: { Eliminate all faults detected by the original stuck-at test set (SAT). }
        Apply all the patterns in SAT in the given sequence.
        Eliminate all detected faults from FL.
        If FL is empty then go to Step 5.
Step 3: { Create the test table containing all $T_1$ and $T_2$ patterns of each fault. }
        FOR each test $t_i \in$ SAT DO
            FOR each fault $f_j \in$ FL DO
                if $t_i$ is a $T_1$ pattern of $f_j$,
                    then mark $t_i$ as $T_1$ of $f_j$.
                if $t_i$ is a $T_2$ pattern of $f_j$,
                    then mark $t_i$ as a $T_2$ of $f_j$.
            END FOR
        END FOR
        Eliminate all faults lacking a $T_1$ or a $T_2$ pattern.
      { These faults are undetectable for the given SAT. }
STEP 4: { Organize the sequence of test patterns to be added to the test set. }
```

```
        Pick the last pattern t from SAT.
        Set the current pattern $t_c$ <-- t.
        WHILE FL is not empty DO
            Find a fault $f_i \in$ FL such that $t_c$ is a $T_1$ of $f_i$.
            IF such an $f_i$ exists THEN
                Find a test $t_n$ which is a $T_2$ of $f_i$.
                Eliminate all the faults (including $f_i$) detected by ($t_c$, $t_n$) pair from FL.
            ELSE
                Select a new $f_j \in$ FL.
                Find a test $t_n$ which is a $T_1$ of $f_j$.
            END IF
            { $t_n$ is the next $t_c$. Set $t_n$ as $t_c$. }
            $t_c$ <-- $t_n$
        END WHILE
    Step 5: { This procedure compacts the above test set. }
        Apply the test patterns in the reverse order of subsequences.
        Eliminate all patterns which do not detect any new faults (i.e., eliminate
        those that detect only faults detected by the previous test patterns).

    END TEST_SEQUENCE_ORGANIZER.
```

Step 2 is needed to reduce the overall processing time. A substantial number of faults are detected in Step 2 and are eliminated from further processing. The above procedure guarantees to achieve the possible highest SOP fault coverage using a given stuck-at test set assuming zero gate delays.

In the following, we illustrate the proposed method by using the example circuit shown in Figure 1. The stuck-at test set {1101, 0100, 1110, 1010, 0000} detects all single stuck-at faults of the circuit.

Step 1: The fault list consisting of all the primary stuck-at faults is

FL = {$(a/0)$, $(a/1)$, $(c/0)$, $(c/1)$, $(d/0)$, $(d/1)$, $(e/0)$, $(f/1)$, $(g/0)$, $(g/1)$, $(h/0)$, $(h/1)$, $(i/1)$, $(j/0)$}.

(z/1) and (z/0) denote a s-a-1 fault and a s-a-0 fault on line z, respectively. Since the stuck-at faults represent corresponding SOP faults, the gate outputs should be initialized properly for the detection of the faults.

Step 2: The stuck-at test set {1101, 0100, 1110, 1010, 0000) is applied in the given order. Since all gate outputs are initialized to x, the first input, 1101, does not detect any fault. The second input, 0100, turns line d into $\overline{D}$ under the presence of (d/1) and the output F becomes D. Hence, 0100 is a T_2 pattern for (d/1). Since the gate output, line i, is properly initialized to 0 under application of 1101, the fault is detected. Similarly, (h/0) and (g/1) are detected by the sequence (1101, 0100). This step repeats until the test patterns are exhausted. Table 4 shows the faults detected by the stuck-at test set. In this example, 9 faults among 14 faults are detected by the stuck-at test set. The detected faults are eliminated from the fault list FL.

Step 3: The remaining faults in Step 2 are {(a/0), (c/1), (d/0), (e/0), (i/1)}. All T_1 and T_2 patterns for the faults are obtained through fault simulation. Table 5 is the test table for undetected faults. For example, the test 0100 is a T_2 pattern of (c/1) and 0000 is a T_1 pattern of (d/0).

Step 4: Once the test table is set up, the sequence of additional test patterns is organized to cover undetected faults using the table. Since 0000 is the last input of the stuck-at test set, the current pattern is $t_c = 0000$. From Table 5, $t_c = 0000$ is a T_1 pattern for (a/0), (d/0), (e/0) and (i/1). (a/0) is chosen to be processed. The test pattern 1010 is a T_2 pattern for (a/0), hence the next pattern is $t_n = 1010$. No other faults are detected by the (0000, 1010) pair. In the next iteration, t_c becomes 1010. The fault (d/0) is chosen and the next pattern is $t_n = 1101$. Since (1010, 1101) detects (i/1) as well as (d/0), both (d/0) and (i/1) are eliminated from FL. In the

Table 4. Faults Detected by The Stuck-At Test Set

test sequence	test pattern	detected faults	# of detected faults
1	1101	-	0
2	0100	(d/1), (g/1), (h/0)	3
3	1110	(c/0), (h/1), (j/0)	3
4	1010	(f/1)	1
5	0000	(a/1), (g/0)	2

Table 5. Test Table for Undetected Faults

fault / test	(a/0)		(c/1)		(d/0)		(e/0)		(i/1)	
	T_1	T_2	T_1	T_2	T_1	T_2	T_1	T_2	T_1	T_2
1101						√				√
0100				√	√			√	√	
1110			√		√					
1010		√			√				√	
0000	√				√		√		√	

third iteration, the remaining faults in FL are (c/1) and (e/0) and the current pattern t_c is 1101. Since no faults are initialized by $t_c = 1101$, fault (e/0) is selected. Two patterns (0000, 0100) are added to the sequence to detect the fault. In the final iteration, (1110, 0100) are added to detect (c/1). The test sequence organized to this step is shown in Table 6. The test sequence numbered 1 through 5 is the original test set and the remaining test sequence contains the added test patterns.

Step 5: This step eliminates unnecessary test patterns from the test sequence. The test sequence of Table 6 is divided into three subsequences, (1101, 0100, 1110, 1010, 0000, 1010, 1101), (0000, 0100) and (1110, 0100). Note that the first patterns of each subsequence, 1101, 0000 and 1110, do not detect any fault. The subsequences are applied to the circuit in the reverse order. The applied test sequence is (1110, 0100), (0000, 0100), (1101, 0100, 1110, 1010, 0000, 1010, 1101). All faults are detected by the first 10 test patterns. The last test pattern, 1101, does not detect any new fault and hence it is eliminated from the test sequence. The final test sequence is (1110, 0100, 0000, 0100, 1101, 0100, 1110, 1010, 0000, 1010) as shown in Table 7.

Comparison with El-Ziq's and Chandramouli's Methods

The major differences between the proposed method and El-Ziq's [3] and Chandramouli's [4] methods lie in selecting initial faults and in choosing initialization patterns. Initially, the number of faults considered for our method is larger than that for the two methods. However, since many faults are eliminated in Step 2 in our method, it has little effect in the overall processing time. As described in the early part of this section, the proposed method has more choices in selecting the initialization patterns. This leads to smaller test sizes as will be shown in Chapter IV. Moreover, the method does not require a path to be sensitized from the faulty gate output to a primary output for the initialization. The reason for this is that a necessary initialization for the proposed method is determined directly by ob-

Table 6. Test Sequence Organized Before Compaction

test sequence	test pattern	detected faults	# of detected faults
1	1101	-	0
2	0100	(d/1), (g/1), (h/0)	3
3	1110	(c/0), (h/1), (j/0)	3
4	1010	(f/1)	1
5	0000	(a/1), (g/0)	2
6	1010	(a/0)	1
7	1101	(d/0), (i/1)	2
8	0000	-	0
9	0100	(e/0)	1
10	1110	-	0
11	0100	(c/1)	1

Table 7. Compacted Test Sequence

test sequence	test pattern	detected faults	# of detected faults
1	1110	-	0
2	0100	(c/1), (g/1), (h/0)	3
3	0000	(a/1), (g/0)	2
4	0100	(e/0)	1
5	1101	(d/0), (i/1), (j/0)	3
6	0100	(d/1)	1
7	1110	(c/0), (h/1)	2
8	1010	(f/1)	1
9	0000	-	0
10	1010	(a/0)	1

serving the faulty gate output. However, El-Ziq's [3] and Chandramouli's [4] methods require the faulty line value to be propagated to a primary output. This requires substantial processing time.

3.4 Implementation of Chandramouli's Method

As described in Section 2.6, El-Ziq [3] and Chandramouli [4] suggested methods of test generation for SOP faults using stuck-at test sets. Since El-Ziq's method is essentially the same as Chandramouli's method, we implemented only Chandramouli's method to compare the performance of the proposed method with that of Chandraouri's method. In this section, we describe the implementation of Chandramouli's method. The method is to apply test patterns of a stuck-at test set in the order of {(SZ), (SO), (SZ)} or {(SZ), (SO)} for all primary inputs which do not fan-out and for all fan-out branches (in case of NAND, AND and inverter). For NOR and OR gates, the sequence {(SO), (SZ), (SO)} or {(SO), (SZ)} is applied as explained in Section 2.6. Since he does not provide any detailed procedure for implementing the method, we used similar techniques used in the proposed method for unspecified parts.

The procedure is divided into 4 steps. The first step is the initialization step. A check point list consisting of all primary inputs which do not fan-out and all fan-out branches is constructed instead of the fault list. Application of two or three test patterns, say (T_1, T_2) or (T_1, T_2, T_3), is required for each check point. In the second and third steps, the sequence of test patterns is organized. First, we construct the test table which contains all (SZ) and (SO) patterns for each check point using fault simulation. Next, a sequence of test patterns are obtained by searching the test table. Finally, we compact the obtained test sequence by applying the subsequences in the reverse order as in Step 5 of the proposed method. The procedure is given below.

```
PROCEDURE CHANDRAMOULI_METHOD;

Step 1: { Initialization }
        Set up the check point list (CPL).
        Set all the logic values to x (unknown).
Step 2: { Create the test table containing all (SZ) and (SO) patterns for each check point. }
        FOR each test $t_i \in$ SAT DO
            FOR each check point j $\in$ CPL DO
                if $t_i$ is (SZ) pattern of j, then mark $t_i$ as (SZ) of j.
                if $t_i$ is (SO) pattern of j, then mark $t_i$ as (SO) of j.
            END FOR
        END FOR
        Eliminate all check points lacking a (SZ) or (SO) pattern.
        { The SOP faults related to these check points are undetectable
        for the given stuck-at test set. }
Step 3: { Organize the sequence of test patterns using the test table. }
        Pick the first pattern t from SAT.
        Set the current pattern $t_c$ <-- t.
        WHILE CPL is not empty DO
            Find a check point i $\in$ CPL such that $t_c$ is a $T_1$ of i.
            IF such an i exist THEN
                Find a test $t_n$ which is a $T_2$ of i.
                Eliminate all check points (including i) covered by ($T_c$, $T_n$) from CPL.
                IF $T_3$ is needed for i THEN
                    $t_c$ <-- $t_n$.
                    Find a test $t_n$ which is a $T_3$ of i.
                    Eliminate all check points covered by ($T_c$, $T_n$) from CPL.
                END IF
            ELSE
                Select a new check point j $\in$ CPL.
                Find a test $t_n$ which is a $T_1$ of j.
            END IF
            { $t_n$ is the next $t_c$. Set $t_n$ as the current test pattern. }
            $t_c$ <-- $t_n$.
        END WHILE
    Step 4: { This procedure compacts the above test set. }
        Apply the test patterns in the reverse order of subsequences.
        Eliminate all patterns which do not cover any new check points (i.e., cover
        only the check points covered by the previous test patterns).

END CHANDRAMOULI_METHOD.
```

Step 4 in the above procedure is not included in Chandramouli's paper. However, we added this routine to allow fair comparison with the proposed method.

3.5 The SOP Fault Simulator

Fault simulation is essential for measuring the fault coverage of test sets. To measure the SOP fault coverage of stuck-at test sets and the performance of the proposed method, we implemented a SOP fault simulator. In this section, we present the SOP fault simulator. The simulator is written in FORTRAN 77 and runs on an IBM 3090 computer. Since the simulator is developed specifically for our purpose, the simulator deals with CMOS circuits consisting of only primitive logic gates.

As explained in Section 2.3, there are two different approaches for modeling the SOP fault, gate level modeling and switch level modeling. In this thesis, we have chosen the gate level approach. In gate level modeling, a CMOS gate is represented by an equivalent gate level circuit using additional gates and a memory element to represent the sequential behavior of the faulty circuit. A SOP fault is represented by its equivalent stuck-at fault [5]. Unlike this method, we use a rather direct approach to represent SOP faults. Instead of converting the faulty gate into its equivalent circuit with the equivalent stuck-at fault, we describe the faulty behavior of the circuit directly in the form of a truth table such as Table 1 in Section 2.4. When a test pattern is applied to the circuit, the simulator matches the pattern with a T_2 pattern detecting the fault. If the pattern is a T_2 pattern for the SOP fault, the faulty gate output becomes M. Then the simulator maintains the previous logic value of the gate output as the current value. Otherwise, the faulty gate behaves the same as the fault free gate. For this purpose, the simulator preserves the previous logic values for all gates.

Another problem in SOP fault simulation stems from the fact that the behavior of a faulty circuit can be affected by hazards occurring in the circuit. As described in Section 2.4, two test patterns designed to detect a SOP fault assuming zero gate delays can be invalidated by hazards occurring due to unequal delays through different signal paths. In the simulator, we employ the transport delay model to consider the effects of hazards on detecting SOP faults. The transport delay is the time taken for an input change to reach the output of the gate, independent of the direction of the signal change [29]. The approach used for the fault simulation is that the simulator detects a SOP fault only if the faulty gate output is properly initialized considering gate delays upon application of a T_2 pattern. The delay values are specified by the user depending on the type and the number of inputs of each gate.

In the following, we present the algorithm used for the fault simulation. In the simulator, several features are considered to enhance the speedup of the fault simulation. The basic feature of the simulator is the concurrent fault simulation scheme. Suppose that a transistor in a CMOS combinational circuit is stuck-open. When a T_2 pattern is applied to the circuit, the faulty gate output becomes M. Then the previous logic value is maintained at the faulty gate output node. Clearly, the effect of the fault on the circuit occurs only at the faulty gate and the gates in the path from the faulty gate output to primary outputs. The other portions of the circuit behave the same as the fault free circuit. In the serial fault simulation scheme, which simulates one fault at a time, a large amount of simulations are repeated. These repeated simulations of the faulty circuits can be avoided by simulating the fault free circuit and the faulty circuits concurrently. In the simulator, the fault free behavior of the circuit is first evaluated and all the faulty functions are evaluated based on the fault free response of the circuit.

In the following, we present the fault simulation procedure. The essence of the procedure is to simulate the circuit assuming zero gate delays and to identify all detectable faults using the current test pattern. We define that a SOP fault is detectable by the current test pattern provided that the faulty gate output becomes M under the application of the current test pattern. Only these faults are applied to further processing that considers gate delays. The procedure is divided into 5 steps. The first step is an initialization step. In this step, all equivalents faults are collapsed using simple fault collapsing techniques described in Section 2.4. In the second step, the current test pattern is applied to the circuit and the circuit is simulated assuming zero gate delays. All detectable faults by the current test pattern are identified and tagged for further processing. In the third and fourth steps, all detected faults are identified and eliminated from the fault list. The third step simulates the tagged faults until the circuit reaches a stable state which is pre-calculated at the initialization step. Once the circuit becomes stable, the fourth step checks the initialization of the faulty gate output. If a faulty gate output is properly initialized, i.e., the faulty gate output is different from the fault free output, it is propagated toward the primary outputs of the circuit. If there exists a sensitizing path from the faulty gate output to a primary output, the fault is detected. All detected faults are eliminated from the fault list. Finally, the fifth step sets the next test pattern as the current test pattern and repeats the simulation. The above procedure is repeated until a predefined fault coverage is obtained or the test patterns are exhausted. The procedure is described below.

```
PROCEDURE CMOS_FAULT_SIMULATION;

Step 1: { Initialization }
          Set up fault list (FL).
          Set up unit_time and max_fault_coverage.
          Set all the logic values into x (don't care).
          { Set the first test pattern as the current test pattern. }
          $t_c$ <-- $t_1$.
Step 2: { This procedure simulates the circuit assuming zero gate delays
          and tags all the detectable faults. }
          Apply $t_c$ to the circuit.
          Perform fault free simulation and store fault free output $Y_n$ of the circuit.
          FOR every fault $f_i \in$ FL DO
                Evaluate faulty gate output $y_f$
                If $y_f$ == M, then tag $f_i$.
          END FOR
          If no detectable faults exist, then go to step 5.
Step 3: { This procedure simulates the circuit considering gate delays until the circuit
          becomes stable and updates faulty gate outputs.}
          { Set current time t into 0. }
          t <-- 0.
          WHILE $t \leq t_{stable}$ DO
                Perform fault free simulation.
                FOR every tagged fault $f_i$ DO
                      Evaluate the faulty gate output $y_f$.
                      IF $y_f$ == M THEN
                            $y_f$ <-- previous output of the gate.
                      ELSE
                $y_f$ <-- fault free output of the gate.
            END IF
        END FOR
        t = t + unit_time.
    END WHILE
Step 4: { This procedure identifies all the detected faults and eliminates
    the detected faults from FL. }
    FOR every tagged fault $f_i$ DO
        Check the faulty gate output $y_f$ and the fault free output $y_n$.
        IF $y_f \neq y_n$ THEN
            Compute primary output $Y_f$ of the circuit.
            If $Y_f \neq Y_n$, then set $f_i$ detected and eliminate $f_i$ from FL.
        END IF
    END FOR
Step 5: { This procedure checks the stop condition. }
    Clear all tags.
    Update the fault_coverage.
    If fault_coverage $\geq$ max_fault_coverage, then stop.
    If no test input exist, then stop.
    Go to step 2.

END CMOS_FAULT_SIMULATION.
```

IV. SOP Fault Coverage of Stuck-At Test Sets

4.1 Objectives of the Experiments

As described earlier, the objectives of this thesis are the evaluation of the SOP fault coverage of stuck-at test sets and improvement of the SOP fault coverage of stuck-at test sets. In Chapter III, we proposed a method for improving the SOP fault coverage of stuck-at test sets. In order to measure the SOP fault coverage of stuck-at test sets and the performance of the proposed method, we conducted experiments using various CMOS combinational circuits. In this chapter, experimental results and related analysis are reported.

The experiments can be classified into 3 categories according to the objectives of the experiments:

1. evaluation of the SOP fault coverage of original stuck-at test sets and comparison with that of random pattern test sets,
2. measurement of the performance of the proposed method and comparison with that of other competing methods, and
3. measurement of the SOP fault escaping rate of stuck-at test sets.

The experimental results of each category are presented in Section 4.2 through Section 4.4.

A 4-bit carry look-ahead adder, a 4-bit arithmetic and logic unit (74181 ALU), and 5 benchmark circuits presented by Brglez and Fujiwara [30] are used for the experiments. All the circuits consist of only primitive logic gates and the numbers of gates range from 6 to 1669. Stuck-at test sets of the circuits are derived using HILO [31], a commercial stuck-at test pattern generator. The SOP fault coverage of a given test set is measured using the SOP fault simulator described in Section 3.5. The transport delay model is employed in the simulator to consider the effect of de-initialization of the faulty gate output as described in Section 3.5. The delay values for the gates used in the simulator are shown in Table 8. The unit (1) delay is assigned to each inverter. Delays of 2 to 6 are assigned to other types of gates depending on the number of inputs of the gate.

4.2 SOP Fault Coverage of Original Stuck-At Test Sets

This section reports the SOP fault coverage of original stuck-at test sets and that of random pattern test sets. The first part of the experiment is to measure the SOP fault coverage of original stuck-at test sets and increased size stuck-at test sets. Four different size test sets are applied to each circuit under test. The size of a test set is increased by repeating the original test sets by two, five and ten times. The sequence of test patterns is rearranged randomly. The second part of the experiment uses random pattern test sets. The random patterns are generated using a pseudo-random number generator.

The experimental results are given in Table 9. Each fault coverage given in the table is the average of 10 experiments. For the stuck-at test sets, the sequence of the test patterns is rearranged randomly for each experiment. For the random pattern test sets, the test patterns are generated randomly with a different initial seed for each experiment.

The column headings of the table are described below.

Table 8. Gate Delay Assignment

gate	number of inputs				
	1	2	3	4	≥ 5
Inverter	1	-	-	-	-
NAND/NOR	-	2	3	4	5
AND/OR	-	3	4	5	6

name	name of the circuit
ng	number of gates
nSOP	number of SOP faults after collapsing equivalent faults
nt	number of test patterns
SA	stuck-at fault coverage of the stuck-at test set (%)
SOP_s	SOP fault coverage of the stuck-at test set (%)
SOP_r	SOP fault coverage of the random pattern test set (%)

In Table 9, four entries are given for each circuit under the column headings "SOP_s" and "SOP_r". The top entry is the SOP fault coverage of the original stuck-at test set. The remaining entries are the SOP fault coverage of the test sets increased in size by two, five and ten times, respectively.

From Table 9, the average SOP fault coverage of the original stuck-at test sets for the seven circuits is 81.7%. When the sizes of stuck-at test sets are increased by two, five and ten times, the average fault coverage is increased to 87.8%, 93.1% and 95.2%, respectively. When the size of a test set is small, the fault coverage increases rapidly as the size of the stuck-at test set increases. However, the fault coverage is saturated for test sets increased in size by five to ten times.

The experimental results show that the SOP fault coverage of a stuck-at test set is higher than that of the same size random pattern test set. However, the difference becomes smaller as the test set size increases. This is mainly due to the fact that a stuck-at test set contains many T_1 and T_2 patterns of SOP faults, while a random test set does not. As the number of test patterns increases, the fault coverage of a stuck-at test set saturates earlier than that of a random pattern test set. Hence, the difference becomes small as the test set size increases.

In summary, we can make the following observations. First, the SOP fault coverage of a stuck-at test set increases as the test set size increases. However, the fault coverage is saturated when the test size is increased by five to ten times depending on the circuit. Second, stuck-at test sets achieve higher SOP fault coverage than random pattern test sets.

Table 9. SOP Fault Coverage of Stuck-At Test Sets and Random Pattern Test Sets

name	ng	nSOP	nt	SA(%)	SOP_s(%)	SOP_r(%)
C17	6	18	6 12 30 60	100.0	70.0 86.7 97.8 99.4	47.8 68.9 90.0 98.3
Adder	15	70	12 24 60 120	100.0	65.1 75.6 88.0 94.9	41.7 54.0 68.3 77.9
74181	91	304	29 58 145 290	100.0	82.8 89.5 96.3 98.2	78.9 88.2 95.8 98.6
C880	383	1206	57 114 285 570	100.0	89.8 93.6 97.1 98.3	74.5 82.1 88.2 91.3
C1355	546	1604	87 174 435 870	99.4	91.9 92.4 92.7 92.9	76.5 83.2 88.1 91.3
C1908	880	2117	122 244 610 1220	99.3	85.9 87.5 89.2 90.2	67.3 72.5 79.6 84.5
C3540	1669	4752	179 358 895 1790	95.8	86.4 89.1 91.4 92.7	75.7 81.7 87.8 91.1
avg	513	1439	70 140 351 702	99.2	81.7 87.8 93.1 95.2	66.1 75.8 85.4 90.4

4.3 SOP Fault Coverage of the Proposed Method

This section reports the SOP fault coverage of the stuck-at test sets organized using the proposed method. The results are compared with those of two competing methods, Chandramouli's method and the heuristic presented in Section 2.6. Since El-Ziq's method [3] is essentially the same as Chandramouli's method [4], results are only compared with Chandramouli's method. Implementation of Chandramouli's method is described in Section 3.4. It should be noted that a similar compaction technique to reduce the test set size is also employed for Chandramouli's method. The heuristic is to organize the sequence of test patterns so that the Hamming distance between two consecutive fault-free outputs is maximized. It should be noted that the sequence of an organized test set, obtained using any one of the three methods, depends on the sequence of the original stuck-at test set.

The experimental results for the three methods are given in Table 10. The SOP fault coverage of the original stuck-at test sets is also given for comparison. Each SOP fault coverage given in the table is the average of 10 simulations in which the sequence of the original stuck-at test set is chosen randomly.

The column headings of the table are described below.

name	name of the circuit
ng	number of gates
nSOP	number of SOP faults after collapsing equivalent faults
SA	stuck-at fault coverage of the stuck-at test set (%)
nt	number of test patterns
SOP	SOP fault coverage of the test set (%)

Two entries are given under column heading "stuck-at" in the table. The top entry is the number of test patterns and the SOP fault coverage of the original stuck-at test set which is

Table 10. SOP Fault Coverage of Organized Test Sets

name	ng	nSOP	Stuck-At		Proposed		Chandramouli		Heuristic	
			nt	SOP(%)	nt	SOP(%)	nt	SOP(%)	nt	SOP(%)
C17	6	18	6 11	70.0 80.6	11	100.0	10	100.0	6	73.3
Adder	15	70	12 34	65.1 80.4	34	100.0	33	100.0	12	69.2
74181	91	304	29 74	82.8 91.2	74	97.2	88	97.7	29	88.3
C880	383	1206	57 134	89.8 94.6	134	97.6	207	98.2	57	90.3
C1355	546	1604	87 304	91.9 92.6	304	92.7	355	92.6	87	91.8
C1908	880	2117	122 358	85.9 88.2	358	89.7	425	89.6	122	85.3
C3540	1669	4752	179 506	86.4 90.1	506	92.7	832	90.1	179	86.4
avg	513	1439	70 203	81.7 88.2	203	95.7	279	95.5	70	83.5

also given in Table 9. For the second entry, the size of the test set is increased to be equal to that for the proposed method. The increased size test set is obtained by randomly selecting the necessary number of test patterns from the original stuck-at test set.

From Table 10, the average SOP fault coverage of the proposed method is 95.7% and that of Chandramouli's method is 95.5%. The average SOP fault coverage of the heuristic is very low (83.5%) and comparable to that of the original stuck-at test sets. The average number of test patterns for the proposed method is 203 and that for Chandramouli's method is 279. The average size of the test sets for the proposed method is 27.2% less than that for Chandramouli's method.

The processing time to organize the sequence of test patterns for the three methods is given in Table 11. The processing time is the average of CPU times from ten runs on an IBM 3090. The column headings of Table 11 are described below.

name	name of the circuit
ng	number of gates
nt	number of test patterns
time	average CPU processing time (seconds)

The average CPU times for the proposed method and Chandramouli's method are 107.6 seconds and 357.53 seconds, respectively. The processing time for the heuristic is negligible. The average processing time for the proposed method is less than one-third of the time for Chandramouli's method. The main reason for the short processing time is that the proposed method does not require sensitizing a path from the faulty gate output to primary outputs for the initialization patterns, while Chandramouli's method does as explained in Section 3.3.

In summary, the proposed method gives small test set sizes and needs substantially less processing time than Chandramouli's method, while achieving high SOP fault coverage. The increase in the SOP fault coverage is negligible for the heuristic.

Table 11. Processing Time of Test Organizing Methods

name	ng	Proposed		Chandramouli		Heuristic	
		nt	time	nt	time	nt	time
C17	6	11	0.03	10	0.03	6	0.03
Adder	15	34	0.07	33	0.07	12	0.04
74181	91	74	0.63	88	1.30	29	0.09
C880	383	134	8.16	207	39.59	57	0.41
C1355	546	304	21.96	355	164.38	87	0.76
C1908	880	358	70.69	425	345.59	122	1.27
C3540	1669	506	651.68	832	1951.80	179	3.14
avg	513	203	107.60	279	357.53	70	0.82

4.4 SOP Fault Escaping Rate

This section reports the SOP fault escaping rate of the original stuck-at test sets and that of the proposed method. We applied an exhaustive test sequence (ETS) of an original stuck-at test set to each circuit and measured its SOP fault coverage. The ETS of a stuck-at test set is obtained as shown in Table 3 in Section 3.2. The escaping rates due to the lack of test patterns and improper initialization are computed using the equations given in Section 3.2. In this experiment, we used only the first 6 circuits since the processing time for C3540 is prohibitive. The experimental results are given in Table 12. Each escaping rate given in the table is the average of 10 simulations.

The column headings of the table are described below.

name	name of the circuit
nSOP	number of SOP faults after collapsing equivalent faults
SOP	SOP fault coverage of the test set (%)
E	total SOP fault escaping rate (%)
ET	SOP fault escaping rate due to lack of test patterns (%)
EI	SOP fault escaping rate due to improper initialization (%)

From Table 12, the average SOP fault escaping rate of the original stuck-at test sets is 19.1%. Among the escaped faults, only 2.3% of the faults are undetected due to the lack of necessary test patterns. The rest of the faults (16.8%) are undetected due to improper initialization. This implies that the SOP fault coverage of an original stuck-at test set can be improved substantially (up to 16.8%) by organizing the test sequence in a proper order. This is achieved by the proposed method. The average SOP fault escaping rate of the proposed method is 3.6%. Among the escaped faults, only 1.5% of SOP faults are undetected due to improper initialization. This means that the SOP fault coverage of the proposed method is

Table 12. SOP Fault Escaping Rate

name	nSOP	Stuck-At				Proposed				ETS			
		SOP	E	ET	EI	SOP	E	ET	EI	SOP	E	ET	EI
C17	18	70.0	30.0	0.0	30.0	100.0	0.0	0.0	0.0	100.0	0.0	0.0	0.0
Adder	70	65.1	34.9	0.0	34.9	100.0	0.0	0.0	0.0	100.0	0.0	0.0	0.0
74181	304	82.8	17.2	1.3	15.9	97.2	2.8	1.3	1.5	98.7	1.3	1.3	0.0
C880	1206	89.8	10.2	0.0	10.2	97.6	2.4	0.0	2.4	100.0	0.0	0.0	0.0
C1355	1604	91.9	8.1	6.4	1.7	92.7	7.3	6.4	0.9	93.6	6.4	6.4	0.0
C1908	2117	85.9	14.1	6.0	8.1	89.7	10.3	6.0	4.3	94.0	6.0	6.0	0.0
avg	887	80.9	19.1	2.3	16.8	96.2	3.8	2.3	1.5	97.7	2.3	2.3	0.0

slightly (1.5%) below the possible highest SOP fault coverage of the stuck-at test sets. This would be also true for Chandramouli's method.

4.5 Remarks

As pointed out earlier, the SOP fault coverage depends on the ordering of the original stuck-at test set. In our experiment, we performed 10 simulations for each circuit, in which the original test set is rearranged in an arbitrary order. We noticed that the effect of the order in the original stuck-at test set is insignificant for large circuits. In the case of C880, which is the most sensitive among the last four large circuits in Table 9, the minimum and the maximum fault coverage among 10 simulations are 88.64% and 90.88% for the original stuck-at test set. When the test sets are organized by the proposed method, the minimum fault coverage is 97.93% and the maximum fault coverage is 98.43%.

The SOP fault coverage may depend on the test patterns of the original stuck-at test set. Even though we did not perform experiments with different test sets, we believe that the SOP fault coverage is not sensitive to the original stuck-at test sets for large circuits.

V. Conclusions

The traditional line stuck-at fault model does not represent SOP faults properly in CMOS circuits. In general, test generation methods detecting CMOS SOP faults are complex and time consuming due to the sequential behavior of faulty circuits. In practice, most testing still relies on stuck-at test sets to test CMOS combinational circuits without organizing the test set at the risk of some SOP faults not being detected.

In this thesis, we conducted experiments to evaluate the SOP fault coverage of various stuck-at test sets for CMOS combinational circuits. The SOP fault coverage is compared with that of random pattern test sets. We proposed a method that improves the SOP fault coverage of the stuck-at test sets. The basic idea of the method is to organize a given test set in a specific sequence to cover undetected faults. The proposed method leads to small test sets and short processing time, while maintaining high SOP fault coverage. The performance of the proposed method is compared with that of two other methods.

The experimental results based on seven circuits show that the average SOP fault coverage of the original stuck-at test sets is 81.7%. The average SOP fault coverage of random pattern test sets is 66.1%. Hence, stuck-at test sets are effective for detecting SOP faults when compared with random pattern test sets.

The experimental results show that the average SOP fault coverage of the proposed method is 95.7% and that of Chandramouli's method is 95.5%. The average size of test sets for the proposed method is 27.8% less than that for Chandramouli's method. Moreover, the average processing time of the proposed method is less than one-third of Chandramouli's method.

Finally, it is worth noting that the experimental results reported in this thesis are the fruit of more than 100 CPU hours on an IBM 3090 computer.

Bibliography

1. R. L. Wadsack, "Fault Modeling and Logic Simulation of CMOS and MOS Integrated Circuits", Bell Sys. Tech. J., Vol. 57, No. 5, pp. 1449-1474, May-June 1978.

2. Y. M. El-Ziq, "Automatic Test Generation for Stuck-Open Faults in CMOS VLSI," Proc. 18th Design Automation Conference, Nashville, TN, pp. 347-354, June 1981.

3. Y. M. El-Ziq and R.J. Cloutier, "Functional-Level Test Generation for Stuck-Open Faults in CMOS VLSI," Proc. 1981 International Test Conference, Philadelphia, PA, pp. 536-546, Oct. 1981.

4. R. J. Chandramouli, "On Testing Stuck-Open Faults," Proc. 13th International Symposium on Fault-Tolerant Computing, Milan, Italy, pp. 258-265, June 1983.

5. S. K. Jain and V. D. Agrawal, "Test Generation for MOS Circuits Using D-Algorithm," Proc. 20th Design Automation Conference, Miami Beach, FL, pp. 64-70, June 1983.

6. S. M. Reddy, M. K. Reddy and V. D. Agrawal, "Robust Tests for Stuck-Open Faults in CMOS Combinational Logic Circuits," Proc. 14th International Symposium on Fault-Tolerant Computing, Orlando, FL, pp. 44-49, June 1984.

7. J. Rajski and H. Cox, "Stuck-Open Fault Testing in Large CMOS Networks by Dynamic Path Tracing," Proc. International Conference on Computer Design, Rye Brook, NY, pp. 252-255, OCt. 1986.

8. H. Cox and J. Rajski, "Stuck-Open and Transition Fault Testing in CMOS Complex Gates," Proc. 1988 International Test Conference, Washington D.C., pp. 688-694, Sept. 1988.

9. K. W. Chiang and Z. G. Vranesic, "Test Generation for MOS Complex Gate Networks," Proc. 12th International Symposium on Fault-Tolerant Computing, Santa Monica, CA, pp. 149-157, June 1982.

10. K. W. Chiang and Z. G. Vranesic, "On Fault Detection in CMOS Logic Networks," 20th Design Automation Conference, Miami Beach, FL, pp. 50-56, June 1983.

11. P. Agrawal, "Test Generation at Switch Level," Proc. International Conference on Computer Aided Design, Santa Clara, CA, pp. 128-130, Nov. 1984.

12. S. H. Robinson and J. P. Shen, "Towards a Switch Level Test Pattern Generation Program," International Conference on Computer-Aided Design, Santa Clara, CA, pp. 39-41, Nov. 1985.

13. M. K. Reddy, S. M. Reddy and P. Agrawal, "Transistor Level Test Generation for MOS Circuits," Proc. 22nd Design Automation Conference, Las Vegas, NV, pp. 825-828, June 1985.

14. H. H. Chen, R. G. Mathews and J. A. Newkirk, "An Algorithm to Generate Tests for MOS Circuits at the Switch Level," Proc. 1985 International Test Conference, Philadelphia, PA, pp. 304-312, Nov. 1985.

15. R. I. Damper and N. Burgess, "MOS Test Pattern Generation Using Path Algebra," IEEE Trans. on Computers, Vol. C-36, No. 9, pp. 1123-1128, Sept. 1987.

16. J. P. Roth, "Diagnosis of Automata Failures: A Calculus and a Method," IBM J. Res. Develop., Vol. 10, pp. 278-291, July 1966.

17. B. W. Woodhall, B. D. Newman and A. G. Sammuli, "Empirical Results on Undetected CMOS Stuck-Open Failures," Proc. 1987 International Test Conference, Washington D.C., pp. 166-170, Sept. 1987.

18. P. Goel,"An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," IEEE Trans. on Computers, Vol. C-30, No. 3, pp. 215-222, March 1981.

19. H. Fujiwara and T. Shimino, "On the Acceleration of Test Generation Algorithms," IEEE Trans. on Computers, Vol. C-32, No. 12, pp. 1137-1144, Dec. 1983.

20. J. M. Acken, "Testing for Bridging Faults (Shorts) in CMOS Circuits," 20th Design Automation Conference, Miami, FL, pp. 717-718, June 1983.

21. D. Baschiera and B. Courtois, "Testing CMOS: a Challenge," VLSI Design, vol. 10, pp. 58-62, Oct. 1984.

22. Y. K. Malaiya, Y. H. Su, "A New Fault Model and Testing Technique for CMOS Devices," Proc. 1982 International Test Conference, Philadelphia, PA, pp. 25-34, 1982.

23. D. Baschiera and B. Courtois, "Advances in Fault Modelling and Test Pattern Generation for CMOS," International Conference on Computer Design, Rye Brook, NY, pp. 82-85, Oct. 1986.

24. N. K. Jha, "Testing of Multiple Faults in Domino-CMOS Logic Circuits",IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Vol. 7, No. 1, pp. 109-116, Jan. 1988.

25. N. K. Jha, "Multiple Stuck-Open Detection in CMOS Logic Circuits", IEEE Trans. on Computers, Vol. C-37, No. 4, pp. 426-432, April 1988.

26. G. Gupta and N. K. Jha, "A Universal Test Set for CMOS Circuits", IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Vol. 7, No. 8, pp. 590-579, May 1988.

27. S. M. Reddy, M. K. Reddy and J. G. Kuhl, "On Testable Design for CMOS Logic Circuits," Prec. 1983 International Test Conference, Philadelphia, PA, pp. 435-445, Oct. 1983.

28. Z. Barzilai, J. L. Carter, V. S. Iyengar, I. Nair, B. K. Rosen and J. Rutledge, "Efficient Fault Simulation of CMOS Circuits with Accurate Models," Proc. 1986 International Test Conference, Philadelphia, PA, pp. 520-529, Sept. 1986.

29. M. A. Breuer and A. D. Friedman, "Diagnosis & Reliable Design of Digital Systems," Computer Science Press, Inc., 1976.

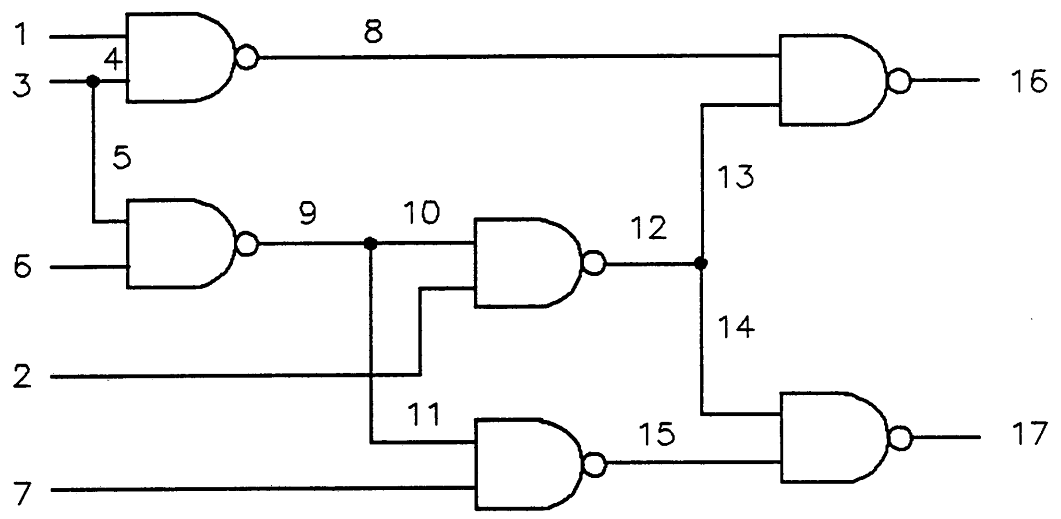
30. F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinational Bench Mark Circuits and a Target Translator in FORTRAN," Special Session on ATPG and Fault Simulation, 1985 International Symposium on Circuits and Systems, Kyoto, Japan, June 1985.

31. HILO-3 User Manual, GenRad Inc., June 1985.

Appendix A. An Example Run of the Fault Simulation Program CMOSIM

The simulator, CMOSIM, is implemented for the SOP fault simulation of CMOS circuits which consist of only primitive logic gates with fixed gate delays. The simulator receives 3 input files which are the circuit input file, the test input file and the delay input file. It outputs the single SOP fault coverage of a given test set. In this appendix, we show an example run of CMOSIM using circuit C17. The gate level circuit description of C17 and input and output files of CMOSIM using this circuit are given next.

A.1 The Gate level description of C17



A.2 Circuit Input File (C17 CCT)

The format of circuit descriptions in CMOSIM is same as that of the benchmark circuits for Automatic Test Generation (ATPG) distributed at the International Symposium On Circuits and Systems (ISCAS), 1985. In fact, C17 is one of the benchmark circuits. The circuit is re-presented at the gate level and in the flat (non-hierarchical) form. The signals are topologically sorted so that no signal is referred before its definition. Along with the topological information, the description includes the definition of a collapsed set of stuck-at faults.

```
*c17 iscas example
*-------------------------------------------------
*
*
*  total number of lines in the netlist .............    17
*        lines from primary input  gates .......     5
*        lines from primary output gates .......     2
*        lines from interior gate outputs ......     4
*        lines from **     3 ** fanout stems ...     6
*
*        avg_fanin  =  2.00,     max_fanin  =  2
*        avg_fanout =  2.00,     max_fanout =  2
*
*
*  simplistically reduced equivalent fault set size =     22
*
*
    1     1gat inpt    1   0        >sa1
    2     2gat inpt    1   0        >sa1
    3     3gat inpt    2   0 >sa0 >sa1
    8     8fan from     3gat        >sa1
    9     9fan from     3gat        >sa1
    6     6gat inpt    1   0        >sa1
    7     7gat inpt    1   0        >sa1
   10    10gat nand    1   2        >sa1
     1     8
   11    11gat nand    2   2 >sa0 >sa1
     9     6
   14    14fan from    11gat        >sa1
   15    15fan from    11gat        >sa1
   16    16gat nand    2   2 >sa0 >sa1
     2    14
   20    20fan from    16gat        >sa1
   21    21fan from    16gat        >sa1
   19    19gat nand    1   2        >sa1
    15     7
   22    22gat nand    0   2 >sa0 >sa1
    10    20
   23    23gat nand    0   2 >sa0 >sa1
    21    19
end
```

In the above description, the line which begins with '*' is a comment line and it is ignored during processing. Each line represents a line specification or input lists of a gate. Seven columns are given for each line in the circuit format unless the line specifies input lists of the gate. The first column is the line number. The second column is the name of the line which is used for the connections. The third column represents the type of line or gate which can be "inpt" or "from" or one of logic functions such as "and", "nand", "or", "nor" or "not". The "inpt" is a primary input of the circuit. The "from" is a fanout branch which is connected to a source line which is specified in the next column after "from". The third column and the forth column of each line except the line with "from" represent the number of fanout branches and the number of fan-in lines of the line, respectively. If the type of the line is a logic function, the next row to the current line specifies the line numbers of input lists of the gate. The ">SA0" (">SA1") on the last two columns of the row indicate that a stuck-at-0 (stuck-at-1) fault on this line is included in the fault list, a blank indicates it is not included. These two columns are ignored in CMOSIM. The SOP fault list of the circuit is automatically created internally by CMOSIM. Hence no fault specification is required. Finally the circuit input file should end with "end".

A.3 Test Input File (C17 TEST)

```
**** c17 stuck-at test set (100%) ****
    5
10000
01100
01111
01010
10110
10101
eeeee
```

The test input file contains test input patterns to be applied to the circuit under test. The first line is a comment line and is ignored during processing. The second line is the number of primary inputs of the circuit under test. The following lines are test patterns. Each line represent a test pattern. The input bits are arranged so that ith bit represents the ith primary input appearing in the circuit input file. For example, the first bit corresponds to the line 1 and the fifth bit to the line 7 of C17 circuit given in A.1. If the number of inputs exceeds 60, the remaining inputs are written in next lines consecutively. The file ends with the string of e's whose number of characters is equal to the number of primary inputs.

A.4 Delay Input File (DELAY INPUT)

```
**** DELAY VALUE OF EACH GATE ***
  GATE #INPUT #DELAY
   not     1      1
   buf     1      0
   xor     2      2
   and     2      3
   and     3      4
   and     4      5
   and     5      6
   and     6      6
   and     7      6
   and     8      6
   and     9      6
   and    10      6
  nand     2      2
  nand     3      3
  nand     4      4
  nand     5      5
  nand     6      5
  nand     7      5
  nand     8      5
  nand     9      5
  nand    10      5
    or     2      3
    or     3      4
    or     4      5
    or     5      6
    or     6      6
    or     7      6
    or     8      6
    or     9      6
    or    10      6
   nor     2      2
   nor     3      3
   nor     4      4
   nor     5      5
   nor     6      5
   nor     7      5
   nor     8      5
   nor     9      5
   nor    10      5
   end     0      0
```

The delay input file defines the delay value of each gate. The first and the second lines are comments and are ignored during the processing. Three entries are given in each line. The first entry represents the name of the gate, the second one represents the number of inputs of the gate and the last one represents the number of unit gate delays. The maximum allowable delay value is 10. If the delay of a gate is not specified, zero delay is assumed. The file should end with "end" with "0" for the second and the third entries.

A.5 Simulation Results

```
************    CIRCUIT  STRUCTURE    ************

Name of circuit :   C17
# of lines                 =          17
# of gates                 =           6
#NAND=    6  #AND=     0  #NOR=     0  #OR=      0
#NOT=     0  #XOR=     0  #BUF=     0  #FANOUT=   6
# of primary inputs        =           5
List of primary inputs :
     1     2     3     6     7
# of primary outputs       =           2
List of primary outputs :
     22   23

************    FAULT  SIMULATION    ************

# of SOP faults            =          24
# of simulated faults      =          18
# of equivalent faults     =           6

used test set              =    stuck_at
# of test patterns         =           6

************    SOP FAULT COVERAGES    ************

#run    # of        # of       fault
        test      detected    coverage
      patterns     faults      ( % )

   1       6          13       72.222
   2       6          13       72.222
   3       6           9       50.000
   4       6          14       77.778
   5       6          14       77.778
   6       6          16       88.889
   7       6          13       72.222
   8       6          11       61.111
   9       6          11       61.111
  10       6          12       66.667

Average SOP fault coverage ( % )   =  69.9999237
```

The output file of CMOSIM gives the single SOP fault coverage of the test set along with the circuit characteristics and the number of faults considered. CMOSIM can run any number of simulations depending on the internal parameters of the program. The parameters for this simulation is given in A.6. The above file is the output of 10 simulations using the test input file given in A.3. For the first run, CMOSIM runs using the original test set given in the input data file. When more than one simulation is specified, CMOSIM rearranges the sequence of the test patterns in an arbitrary order using a random number generator and applies the test pattern into the circuit under test.

A.6 Parameters for CMOSIM

```
C The following parts defines internal parameters.
C If MODE = 0, fault simulation.
C         = 1, fault free simulation.
C If MINPUT = 0, uses test file.
C           = 1, use random patterns.
C MLOOP defines number of runs in fault simulation.
C If MAXLEN = 0, number of test patterns depends on MSIZE.
C          else, MAXLEN defines the number of test patterns.
C MSIZE = n multiples of original test set.
C COVMAX defines maximum fault coverage.
C If MLOG = 0, no log file created.
C         = 1, creates log file.
C ISEED defines initial random seed.
C
      MODE = 0
      MINPUT = 0
      MLOOP = 10
      MAXLEN = 0
      MSIZE = 5
      COVMAX = 100.0
      MLOG = 0
      ISEED = 1764817
```

Since CMOSIM is designed to run in batch mode, the operation mode is defined internally by setting parameters of the program. Hence the program should be compiled again after setting the parameters. The parameters for the operation mode are defined at the beginning of the program as shown above.

"MODE" defines the simulation mode. If MODE is set to 0, a good circuit simulation is performed. Otherwise, a fault simulation is performed.

"MINPUT" selects the input test pattern of the program. If MINPUT is set to 0, CMOSIM uses the given test input file for simulation. Otherwise, it generates random patterns internally and uses them for simulation.

"MLOOP" defines the number of runs in a fault simulation. For the first run with MODE = 1, the simulator simulates the given test set in the original sequence. For multiple runs, it rearranges the test patterns randomly each time and uses the rearranged test patterns for simulation.

"MAXLEN" defines the number of test patterns to be applied to the circuit under test for each run. If MAXLEN is 0, the number of test patterns is determined by the parameter "MSIZE".

"MSIZE" defines the number of test patterns to be applied to the circuit under test for each run when MAXLEN = 0. The number of test patterns is determined by the number of test patterns of the given test input file multiplied by MSIZE. For example, MSIZE = 10 means the number of test patterns of the test set is increased by 10 times.

"COVMAX" defines the maximum fault coverage of a fault simulation.

MLOG determines whether the program produces the log file or not. If "MLOG" is set to 1, CMOSIM generates the log file. For the C17, the output file name is "C17 LOG". This log file is generated only for the first run of a simulation.

Finally, "ISEED" defines the initial seed for the random number generator used in the program.

Appendix B. An Example Run of the Test Organization Program TSORT1

TSORT1 organizes the sequences of test patterns of a given test set according to our proposed method. This appendix gives a sample run of the test organization program "TSORT1" using C17. TSORT1 receives 2 input files which are the circuit input file and the test input file. It outputs the test set organized by the proposed algorithm. The circuit input file and the test input file are same as those of CMOSIM described in Appendix A. The output file is formatted such that it can be applied directly into "CMOSIM" to enhance the SOP fault coverage of the test set. B.1 contains the output file.

B.1 Result (C17 TEST1)

```
****** C17 ORGANIZED TEST SEQUENCE ******
     5
10101
10000
01010
10000
01100
01111
01010
10110
10101
01111
10110
eeeee

# of original test patterns          =      6
# of organized patterns              =     13
# of compacted test patterns         =     11
# of simulated faults                =     18
# of undetectable faults             =      0
# of potentially detectable faults =     18 ( 100.0 % )

Execution times (10 msecs) :
Initialization =         2
Process        =         1
Output         =         1
Total          =         4
```

The format of the output file of TSORT1 is the same as that of a test input file used in CMOSIM except for the information written in the last part of the output file. Since CMOSIM ignores this part in processing, the output file can be applied directly to CMOSIM except that the name of the test input file for CMOSIM should be redefined to indicate the organized test file.

The first three lines displaying test data represent the number of test patterns. "# of original test patterns" is the number of test patterns of the original stuck-at test set given in the test input file. "# of organized patterns" is the number of test patterns of the intermediate test set organized without compaction. "# of compacted test patterns" is the number of test patterns of the final output test set organized by TSORT1.

The following two lines indicate the number of simulated faults and the number of potentially detectable faults, respectively. The number of potentially detectable faults means the number of faults whose t_1 and t_2 patterns exist in the original test set.

The last five lines show the CPU run time taken by TSORT1. The unit of the time is 10 msecs. "Initialization" means the time taken for the initialization of the program. "Process" means the time taken to organize the test set. "Output" means the time taken to write the output results.

Appendix C. Circuit Format Translation Program, TRANS

Since the netlist of a circuit used in HILO is different from that of CMOSIM or TSORT1, the circuit format is converted using a circuit format translation program "TRANS". "TRANS" receives the circuit input file described in Appendix A.2 and outputs the converted circuit format for HILO. The input file name is "C17 CCT" and the output file name is "C17 HILO" for the C17 circuit. It also generates the random patterns for HILO test input patterns. The translated output file for C17 is given in C.1. For a detailed description of the file specification, refer to the HILO-3 User's Manual [31].

C.1 Translated Output File (C17 HILO)

```
cct c17(w16,w17,a[1:5])
buf         igat1(w1,a[1]);
buf         igat2(w2,a[2]);
buf         igat3(w3,a[3]);
buf         gat4(w4,w3);
buf         gat5(w5,w3);
buf         igat6(w6,a[4]);
buf         igat7(w7,a[5]);
nand(1,1)   gat8(w8,w1,w4);
nand(1,1)   gat9(w9,w5,w6);
buf         gat10(w10,w9);
buf         gat11(w11,w9);
nand(1,1)   gat12(w12,w2,w10);
buf         gat13(w13,w12);
buf         gat14(w14,w12);
nand(1,1)   gat15(w15,w11,w7);
nand(1,1)   gat16(w16,w8,w13);
nand(1,1)   gat17(w17,w14,w15);
input       a;
wire        w1,w2,w3,w4,w5,w6,w7,w8,w9
            w10,w11,w12,w13,w14,w15,w16,w17.
```