FAULT SIMULATION FOR STUCK-OPEN FAULTS
IN CMOS COMBINATIONAL CIRCUITS

A Thesis Presented to
The Faculty of the College of Engineering and Technology
Ohio University

In Partial Fulfillment of the Requirements for the Degree

Master of Science
in
Electrical Engineering

By
Lang Su

March, 1993
Athens, Ohio
ACKNOWLEDGEMENTS

I would like to express my gratitude to my advisor Dr. Mokari for his constant advice and suggestions during my research for this thesis. I would also like to thank Dr. Curtis, Dr. Staryzk and Dr. Snyder for serving on my committee.

With much love, I wish to express my heartfelt thanks to my great mother Jiaping Mo and my father Huaqin Su who gave me valuable encouragement and moral support. I am heavily indebted to my brothers, Wei and Yang, for their variety of supports in my trouble time. Finally, special thanks to my wife Jian for her love and understanding throughout my graduate study. Without them, ......
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>CHAPTER ONE</th>
<th>INTRODUCTION</th>
<th>.......................... 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Stuck-at Fault Model</td>
<td>1</td>
</tr>
<tr>
<td>1.2</td>
<td>Stuck-open Fault Model</td>
<td>2</td>
</tr>
<tr>
<td>1.3</td>
<td>Brief Review of Stuck-open Fault Testing</td>
<td>7</td>
</tr>
<tr>
<td>1.4</td>
<td>Fault Simulation</td>
<td>8</td>
</tr>
<tr>
<td>1.5</td>
<td>The Objective of This Research</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CHAPTER TWO</th>
<th>CMOS CIRCUIT</th>
<th>.......................... 11</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>CMOS Technology</td>
<td>11</td>
</tr>
<tr>
<td>2.2</td>
<td>CMOS Failure Modes</td>
<td>14</td>
</tr>
<tr>
<td>2.3</td>
<td>FCMOS Circuits</td>
<td>18</td>
</tr>
<tr>
<td>2.3.1</td>
<td>Static FCMOS Circuits</td>
<td>18</td>
</tr>
<tr>
<td>2.3.2</td>
<td>Gate Level Circuits</td>
<td>19</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CHAPTER THREE</th>
<th>TRADITIONAL APPROACHES OF STUCK-OPEN FAULT DETECTION</th>
<th>.......................... 22</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Brief Review of Traditional Circuit Testing</td>
<td>22</td>
</tr>
<tr>
<td>3.2</td>
<td>Review of CMOS Stuck-Open Faults Testing in CMOS Circuits</td>
<td>23</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CHAPTER FOUR</th>
<th>CIRCUIT AND FAULT MODELING</th>
<th>.......................... 28</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Gate Level Circuit Model from Transistor Level Circuits</td>
<td>28</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Gate Level Modeling of CMOS Circuit From Transistor Level</td>
<td>29</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Simplified Model</td>
<td>33</td>
</tr>
<tr>
<td>4.2</td>
<td>Fault Equivalence Between SOP Fault and Stuck-At Faults</td>
<td>35</td>
</tr>
<tr>
<td>4.3</td>
<td>Detection of SOP Faults Using Stuck-At Faults</td>
<td>40</td>
</tr>
<tr>
<td>4.4</td>
<td>Stuck-Open Faults Collapsing</td>
<td>42</td>
</tr>
</tbody>
</table>
CHAPTER FIVE    STUCK-OPEN FAULT SIMULATION ................. 44

5.1 Test Patterns Generated By LFSRs ........................ 44
5.2 Simulation Methodology ..................................... 48
5.3 Parallel Fault Simulation ................................. 49
5.4 Stuck-Open Fault Simulator and Optimal Design of LFSR
For BIST Implementation ..................................... 55

CHAPTER SIX    EXAMPLES ........................................ 63

6.1 Simple Examples ............................................. 63
6.2 Bench Mark Circuits ......................................... 73

CHAPTER SEVEN   CONCLUSION .................................... 79

REFERENCES ..................................................... 81

APPENDIX A. Circuit Format ..................................... 84

APPENDIX B. A Primitive Polynomial For Every Degree Through
100 .......................................................... 85
CHAPTER ONE
INTRODUCTION

With the advent of very large scale integration (VLSI), very complex circuits can be implemented on a single chip. The advantages of VLSI circuits, such as low power requirement and high density, are obvious. CMOS (Complementary metal oxide semiconductor) technology has played a dominant role in this development. However, the rapid growth in circuit complexity resulting from development in VLSI technology has greatly increased the difficulty and cost of testing integrated circuits. The chips are subject to a great variety of physical failures. These failures can arise during the manufacturing process or can appear during the lifetime of the system. The problem is how to test the failures of VLSI chips to ensure that they function as they are supposed to. The fault models most commonly used for CMOS circuits are stuck-at, stuck-open, stuck-on and bridging [1].

1.1 Stuck-at Fault Model

Stuck-at fault model has been widely used in industry. In this model it is assumed that the fault causes a line in the circuit to behave as if it is permanently at logic 0 or logic 1. If the line is permanently at logic 0 it is said to be
stuck-at 0 (s-a-0), otherwise if it is permanently at logic 1 it is called to be stuck-at 1 (s-a-1).

<table>
<thead>
<tr>
<th>x1</th>
<th>x2</th>
<th>f</th>
<th>f1</th>
<th>f2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Consider the two-input CMOS NAND gate in Fig. 1.1. Let us first examine the short denoted by s1. This short forces the line fed by input x2 to behave in a s-a-0 fashion. Similarly, the short denoted by s2 forces the line fed by input x1 to behave in a s-a-1 fashion. In Table 1.1 the fault-free output is denoted as f whereas the outputs in the presence of shorts s1 and s2 are denoted as f1 and f2 respectively. From this table one can see that the vector (x1,x2) = 11 detects short s1 and the vector (x1,x2) = 01 detects short s2.

### 1.2 Stuck-Open Fault Model

One of the testing challenges of CMOS is the detecting of stuck-open (SOP) faults. When a transistor is rendered non-conducting by a fault it is said to be stuck-open. This can be the result of a broken conductor or can arise from a failure associated with loss of charge transfer capability in one or
Fig. 1.1 A Two-input CMOS NAND Gate With S1, S2 Short.
more of the transistors in a CMOS IC logic circuit [1-3]. It causes combinational circuits to exhibit sequential behavior [1-3]. This fault causes a high impedance state at the output node for at least one logic state and in such cases the node voltage is assumed to be that of the previous logic state. For example, consider the 2-input NAND gate shown in Fig. 1.2 which has an open circuit in the drain connection of n-channel transistor N2. Input condition x1x2 = 11 produces inactivation of N2 and the cell output floating. The load capacitance will hold the previous output value for some period of time which is a function of the amount of capacitance and the leakage currents in the circuit. The open circuit prevents active pull-down of the output node. The circuit failure has thus introduced memory into a combinational circuit. Yet the open circuit prevents active pull-down of the output node. For this reason, the CMOS stuck-open fault is sometimes referred to as a "memory" fault. So, for stuck-open faults, the order in which the test vectors are applied is important. Two test vectors are required for each targeted fault [1-3]. The first test vector "sets up" the circuit states, while the second test vector will detect the presence of the fault.

Consider the NAND cell in Fig. 1.2. Suppose conductor f is broken. This break prevents transistor P2 from conducting. Thus it results in a stuck-open fault in transistor P2. Suppose that the vectors shown in Table 1.1 are applied in the order shown. Even when f is present, the resultant output will
Fig. 1.2 2-input NAND Gate With Stuck-Open Faults.
still be the same as the fault-free output. This can be verified as follows. When 00 and 01 are applied, transistor P1 conducts, resulting in $Z = 1$. When the third vector 10 is applied, neither the pMOS network nor the nMOS network can conduct. Therefore the previous logic value is retained at the node. Finally, when 11 is applied, the nMOS network conducts and $Z$ becomes 0. Thus in order to detect this stuck open fault, a sequence of vectors is required. The reason that stuck-open fault in transistor P2 did not get detected is that the proper sequence of vectors was not fed to the circuit.

It usually requires a sequence of two vectors to detect a stuck-open fault. The first vector is called the initialization vector and the second vector is called test vector. The sequence of these two vectors is referred to as the two-pattern test. The two-pattern test for the stuck-open fault in transistor P2 is <11,10>. The vector 11 initializes the output nodes, when 10 is applied next, the output node remains at 0 and the fault is detected. Table 1.2 shows the

<table>
<thead>
<tr>
<th>Faults</th>
<th>f1 (a,b,c)</th>
<th>f2 (d,e,f)</th>
<th>f3, f4 (g,h,i,j,k)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization Vector (T1)</td>
<td>x1, x2</td>
<td>x1, x2</td>
<td>x1, x2</td>
</tr>
<tr>
<td></td>
<td>1 1</td>
<td>1 1</td>
<td>0 0</td>
</tr>
<tr>
<td>Test Vector (T2)</td>
<td>0 1</td>
<td>1 0</td>
<td>1 1</td>
</tr>
</tbody>
</table>
two-pattern tests for stuck-open faults in Fig. 1.2.

1.3 Brief Review of The Stuck-open Testing

Wadsack first developed the stuck-open (SOP) fault model in [1], since then many different schemes have been proposed for stuck-open testing. Typically, a method designed to find a test set for single stuck-at faults is extended to the stuck-open case [4]. However, the construction of a test set from stuck-at faults test pattern generators which have no ordering requirement, is quite complex, and is time-consuming, especially for VLSI circuits.

Recently, there has been an increasing interest in testing combinational circuit using exhaustive and pseudo-exhaustive method [5-8]. Most BIST schemes employ linear feedback shift registers (LFSRs) as the test pattern generators (TPGs) to apply exhaustive or random tests for circuits under test (CUTs). On the other hand, logical testing of stuck-open (SOP) faults in CMOS devices requires two-pattern tests. An application of consecutive input patterns is also effective for delay testing of CUTs. Although serial sequences of LFSRs might be considered sufficiently random, the consecutive parallel patterns have strong dependence. Therefore, analysis and synthesis of two-pattern capability of TPG circuits are current research subjects. Fault simulation is an important tool in this research area.
1.4 Fault Simulation

Fault simulation of logic circuits is an important part of the test-generation process. It is used for the purpose of generating dictionaries and for verifying the adequacy of tests (test pattern and test sequences) intended to detect and locate logic faults. Moreover, fault simulation is often necessary to determine the fault coverage of a given test, that is, to find all the faults detected by the test. Fault simulation is also employed for analyzing the operation of a circuit under various fault conditions in order to detect circuit behavior not considered by the designer. For very large circuits, fault simulation which requires a very sophisticated test station and testing based on software are time-consuming and costly.

1.5 The Objective of This Research

In this thesis, we present a SOP simulator, sopsim.chu, which is for stuck-open faults simulation. We have modified extensively the test pattern generator developed at Virginia Polytechnic & State University [9] to add stuck-open fault simulation capability to the software. We also added hardware based test pattern generation subroutines to the program. The key idea of testing and simulation of SOP faults in this method is to convert a CMOS circuit under test into an
equivalent gate level circuit and SOP faults into the stuck-at faults. Then simulator simulates the SOP fault by fault free simulation and fault simulation.

The test patterns can be generated by random pattern generator or different linear feedback shift registers (LFSRs). And the two-pattern capabilities of different designed LFSRs circuits can be evaluated. In the proposed method, the test vectors generated by LFSR are applied to the CUT. Starting with the initial state of LFSR (seed), the first pair of test vectors is applied to the circuit. The circuit is simulated by the proposed SOP fault simulator and the detected faults are marked. The LFSR generates another test vector pair and the newly detected faults are marked. This process is repeated until all detectable stuck-open faults are detected or desired fault coverage is obtained. Simulation is carried out using different designs for LFSRs and the circuit with the best two-pattern capability is selected to implement the final circuit. Experimental results show that the proposed method gives short test time while getting good fault coverage. We believe that this is the first attempt to study stuck-open fault detection by the two-pattern capabilities of LFSR circuits. The method proposed in this thesis can be expanded to different autonomous linear sequential circuits.

In chapter two, CMOS technology, CMOS failure modes and FCMOS circuits are presented. Chapter three presents the traditional means of detecting stuck-open faults. Chapter four
presents the circuits and fault modeling by proposed method. In chapter five, the proposed SOP fault simulator is presented. In chapter six, some simple circuits and benchmark circuits are simulated using proposed method, experimental results and observations are presented. Finally, chapter seven concludes this thesis.
CHAPTER TWO
CMOS CIRCUIT

In this chapter, we will discuss CMOS technology in very simplified fashion. We emphasize the switch behavior of CMOS model for clear understanding of functional fault mechanisms in a CMOS circuit. In this thesis, we limit our discussion in fully complemented MOS (FCMOS) circuits with their gate level model [10].

2.1 CMOS Technology

The CMOS circuits considered in this thesis are composed of P channel and N channel enhancement mode field effect transistors (FET) [11]. In considering the functional behavior of a circuit, the transistors can be considered as simple switches [11]. The symbols are shown in Fig. 2.1.

When a logic 1 is applied to the gate of an N channel transistor, a conducting path is created between the source and drain of the transistor. When a logic 0 is applied to the gate of an N channel transistor no conducting path is formed. For a P channel transistor, when logic 0 is applied to the gate, the conducting path is created. We summarize this switching behavior in Table 2.1.
Fig. 2.1 CMOS enhancement mode transistor symbols.

Fig. 2.2 Block Diagram of a CMOS Cell.
Table 2.1 Switch Behavior of CMOS Transistors

<table>
<thead>
<tr>
<th>Gate Input</th>
<th>Logic 0</th>
<th>Logic 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>N Channel FET</td>
<td>Not Conducting</td>
<td>Conducting</td>
</tr>
<tr>
<td>P Channel FET</td>
<td>Conducting</td>
<td>Not Conducting</td>
</tr>
</tbody>
</table>

Obviously, a conducting N channel transistor passed good logic 0s but poor logic 1s. While a conducting P channel transistor passes poor logic 0s but good logic 1s [11]. Because of this inverted logical behavior, a CMOS cell is constructed according to the block diagram in Fig. 2.2. The output of the cell is 0 if the logic values of the cell inputs create a path from the output to GND through conducting N channel transistors. Likewise, the output of the cell is 1 if the logic values of the cell inputs create a path from the output to Vdd through conducting P channel transistors. In normal operation, any set of input values should not create a path through both the load circuit (P channel transistors) and the driver circuit (N channel transistors).

Fig. 2.3 shows a CMOS inverter. When the input is at logic 0, the P channel transistor is conducting and the output goes to logic 1 (Vdd). When the input is at 1, the N channel transistor is conducting and the output goes to 0 (GND). Fig. 2.4 shows a CMOS 2-input NAND cell. Either input at 0 pulls the output to 1. The output is 0 only if both inputs are 1.
Fig. 2.5 shows a CMOS 2-input NOR cell. NAND, NOR, and NOT (inverter) gates are referred to as primitive gates. Any complex gate can be realized by these primitive gates.

The structure in Fig. 2.2 is termed fully complemented since for every input condition there is a conducting path through either the driver or the load circuit. We denote this kind of CMOS as FCMOS [10]. Several other variants of CMOS exist e.g. pseudo-NMOS, dynamic, clocked, cascade voltage switch logic, domino logic and pass transistor logic. In this thesis, we restrict our attention to fully complemented static cells. And the discussion of failure modes below is for this particular case.

2.2 CMOS Failure Modes

A CMOS circuit is formed by superimposing several layers of conducting material (metal and diffusion), insulating material and transistor forming material. The physical failures in a CMOS circuit which lead to permanent functional faults are broken conductors, faulty transistors and shorted conductors. Note that the latter can happen on the same layer i.e. within the same conducting material, as well as between layers due to failures in the insulating material between the layers. In [12], MOS failure modes involved the analysis of failures in 43 faulty 4-bit microprocessor chips showed that the short failures in chips are first place, and broken
Fig. 2.3 An CMOS Inverter.
Fig. 2.4 A Static CMOS 2-input NAND Gate

Fig. 2.5 A Static CMOS NOR Gate
failures are second. This indicates that shorted and broken conductors should be closely considered when considering failures in MOS circuits.

Conductor shorts in a CMOS circuit will generally result in an erroneous path from VDD to GND and thus result in a significant increase in the static current dissipation. These failures should thus be observably by monitoring this current. As we discussed in chapter one, broken conductors present more of a challenge.

If we were to try to derive test vectors for every possible physical failure in a VLSI chip, the problem would soon become unmanageable. In order to successfully deal with the problem, we present the physical failures in a chip at a higher level with the help of a fault model. Any one fault from the fault model may represent many physical failures. Thus the use of fault model speeds up the testing process. The fault models most commonly used for CMOS circuits are stuck-at, stuck-open, stuck-on and bridging. Another fault model which is becoming increasingly important is the delay fault model. We have discussed the stuck-at fault model and stuck-open model in 1.1 and 1.2 separately. In this thesis we concentrate on stuck-open faults.
2.3 FCMOS Circuits

CMOS circuits are generally categorized as either static or dynamic. In this thesis, we discuss the static FCMOS circuits.

2.3.1 Static FCMOS Circuits

A static CMOS circuit is made up of an interconnection of static CMOS gates, which consist of a network of pMOS transistors, called the pMOS network, and a network of nMOS transistors, called the nMOS network. The pMOS network is sometimes called the load network and the nMOS network is called the driver network. The structure is shown in Fig. 2.2. In positive logic a pMOS transistor conducts when its input is 0 and an nMOS transistor conducts when its input is 1.

This kind of circuit has the following two properties:

1. Corresponding to each nMOS transistor there exists a pMOS transistor which is fed by the same input, and vice versa.
2. For each input vector a conduction path is activated either in the pMOS network or the nMOS network, but not both.

Such CMOS gates are sometimes called fully complementary MOS (FCMOS) gate. If one of the networks is known to be series-parallel in nature then it is easy to derive the other network from it. This can be done by noting that a series (parallel) connection of transistors in the nMOS network corresponds to
a parallel (series) connection of transistors in the pMOS network. However, if one of the networks is known to be non-series-parallel in nature then it takes slightly more effort to derive the other network from it [10].

2.3.2 Gate Level Model

For testing a circuit, frequently its logic gate-level model is used. A gate-level model is simply the gate-level representation of the circuit [10].

Consider the static CMOS complex gate in Fig. 2.6. Its gate-level model is given in Fig. 2.7. This model is obtained by replacing a series connection of transistors on the nMOS network by an AND gate and a parallel connection by an OR gate. Since every CMOS gate is inverting, an inverter is placed at the output of the model. If a CMOS circuit consists of many CMOS gates, each CMOS gate is replaced by its gate-level model and these are interconnected as in the original circuit to obtain the gate-level model of the short circuit. We will be discussing the gate-level model more detail in Chapter four.
Fig. 2.6 A Static CMOS Complex Gate.
Fig. 2.7 The Gate-level Model of the Circuit in Fig. 2.6.
CHAPTER THREE

TRADITIONAL APPROACHES OF STUCK-OPEN FAULT DETECTION

Wadsack developed stuck-open (SOP) fault model for CMOS circuits in [1], since then, many techniques have been proposed on testing SOP faults in CMOS circuits. In this chapter, we give the background of the proposed research with emphasis on the detection of SOP faults using stuck-at test sets and built-in self test (BIST) approach. In section 3.1, we briefly present conventional circuit testing methods. Section 3.2 reviews previous researches on stuck-open testing in CMOS circuit and SOP fault testing in built-in self-test (BIST).

3.1 Brief Review of Traditional Circuit Testing

Digital circuits can be represented using primitive logic gates, such as NAND, AND, NOR, OR and NOT, and their interconnections. For most practical purposes, the on-line stuck-at fault model has been commonly used for testing the circuits. In the on-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 0 (s-a-0) fault on line i implies that the line i is stuck permanently at logic value 0. While the stuck-at 1 (s-a-1) fault on line i means that the line i is
stuck at logic 1 permanently.

Over the years, many test generation algorithms for detecting on-line stuck-at faults have been proposed. The widely used D-algorithm [13], PODEM [14] and FAN [15] are such algorithms. These algorithms and the on-line stuck-at fault model have been proved to be effective in testing conventional logic circuits for TTL and nMOS technology.

3.2 Review of Stuck-Open Fault Testing in CMOS Circuit

Because of inaccurate circuit modeling and fault modeling, the conventional circuit testing methods based on the line stuck-at fault model are no longer adequate to test CMOS circuits where stuck-open faults are considered [16-21].

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

Wadsack developed the stuck-open (SOP) fault model in [1], since then, many test generation methods have been developed to detect such faults. Two approaches, gate level algorithms and switch level algorithms have been proposed. The advantage of gate level algorithms is that the well established stuck-at test generation techniques can be used to
test SOP faults [16-19]. Switch level algorithms use switch level description to represent the circuits and the faults [20-21]. It is possible to derive tests from the switch-level model for some stuck-open faults in CMOS circuits which may not be detected if the test patterns are derived from the gate-level model [21]. Switch level test generation algorithms give better fault coverage than gate level test generation algorithms. However, they are more complex and time consuming.

Various previous approaches for testing CMOS circuits used stuck-at test to detect SOP faults. These approaches are based on the gate level description of the circuit which consist of only primitive logic gates and/or CMOS complex cells. In [18], Chandramouli showed that a test set can be generated to detect all single SOP faults in CMOS circuits. The sequence of the test patterns of a stuck-at test set is rearranged to cover all single stuck-at faults assuming zero gate delays. In [17], an algorithm in which CMOS complex cells are consider is proposed to detect stuck-open faults. The stuck-at test set is applied to detect some SOP faults. Then, the algorithm generates test patterns for the undetected SOP faults. Like Chandramouli's method [18], it also organizes the test sequence of a stuck-at test set to detect SOP faults.

Different from the above methods, in [4], Jain and Agrawal proposed a procedure to generate SOP tests for general CMOS circuits including transmission gates. The approach converts a CMOS circuit with stuck-open faults into an
equivalent gate level circuit with stuck-at faults. Then stuck-at test generation algorithm, D-algorithm which is modified a little, is applied to find a test set. The advantage of this method is that it is easy to apply and it can use well established stuck-at test algorithms to test SOP faults. However, the size of the equivalent circuit is usually far larger than that of the original circuit, and the equivalent circuit has memory elements to make the test generation procedure complex.

All 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 may be invalidated. To deal with this problem, Reddy et al [19] proposed a procedure to generate robust tests which are not invalidated by gate delays and/or input timing skews. 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. Because of inaccurate circuit modeling and inaccurate fault modeling, gate level algorithms suffer low fault coverage for SOP faults. To deal with the problems, switch level algorithms have been presented. Chiang first derived a test generation
method using connection graph model to represent circuits [20]. nMOS and pMOS 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. Stuck-open/on faults are represented by assigning edge labels of faulty transistors to logic 0 (1) permanently. Once the circuits and faults are represented in this way, test patterns are generated by reversing a path in the connection graph or by analyzing the path and/or the cutset expressions driven from the graph representations. Several researchers proposed switch level test generation algorithms based on the D-algorithm. 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 [22] used the PODEM algorithm to generate tests for switch level networks. The method has no restrictions on the type of circuits.

In the above algorithms, only single SOP faults are considered. Rajski [24] attempted to detect multiple SOP faults in CMOS circuits. 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.

The above approaches are all test set methods. The test
set approach has drawbacks in VLSI environment. For large circuits it is very time-consuming.

Recently, considerable attention has been paid to build-in self-test and design for testability [5-7]. Many BIST schemes employ LFSRs as the test pattern generators to apply exhaustive or random tests for circuits under test. However, stuck-open faults in CMOS circuits need two-pattern tests. Test length and time become very long for large circuits. In [6], $2^{2n-1}$ test patterns generated by 2n-stage of LFSR for an n-input CUT is used to apply exhaustive two-pattern test. However, the number of test patterns generated in a cycle of 2n-stage LFSR is too long for most real applications. In [5], $(n+1)2^n$ test length are applied for n-input circuit. It shortens the test length, but it still remains very long. In [8], analyses of two-pattern capabilities of autonomous test pattern generators are presented, the approach is more analytical than synthetical.

In this research, we have used LFSR circuits along with stuck-open fault simulation to design BIST circuits with short testing time.
CHAPTER FOUR
CIRCUIT AND FAULT MODELING

In this chapter, we describe the procedure to transform a CMOS switch level circuit into its equivalent gate level circuit. Once the gate level circuit models are established, we represent the fault equivalence between stuck-at faults and stuck-open faults, and how to use stuck-at faults detecting stuck-open faults. Our modeling is confined to FCMOS gates with the dual structure.

4.1 Gate Level Circuit Model From Transistor Level

The CMOS combinational circuit considered in this thesis is composed of interconnected FCMOS gates. But the method proposed here can be expanded to the general CMOS combinational circuits.

We first use the Jain-Agrawal method [4] which uses a modeling block called the B-block to represent the memory state caused by the stuck-open fault. Then, the Reddy-Agrawal-Jain method [10] is used to simplify the model, such that the block is not used, and only n network or p network of the circuits are used for modeling the FCMOS circuits.
4.1.1 Gate Level Modeling of CMOS Circuit From Transistor Level

We now use the primitive cells AND, OR, NOT gates, and the B-block which is used to represent the memory state caused by the stuck-open fault, to model CMOS circuits. A gate-level model is first derived for both the nMOS and pMOS networks of a CMOS gate in the circuit. A series (parallel) connection of transistors is replaced with an AND (OR) gate. The inputs to the gate-level model of the pMOS network are complemented. This is done to take care of the fact that a pMOS transistor conducts when its input is 0. Let the output of the gate-level model which models the pMOS (nMOS) network be $S_1$ ($S_0$). $S_1$ and $S_0$ are fed to the B-block. The truth table of the B-block is shown in Table 4.1.

<table>
<thead>
<tr>
<th>$S_1$</th>
<th>$S_0$</th>
<th>$Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>M</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

When $S_1=0$ and $S_0=1$, it means that there is a conduction path through the nMOS network but not the pMOS network. Therefore the output $Y$ of the B-block is 0.

Similarly, for $S_1 = 1$ and $S_0 = 0$, since only the pMOS network conducts, $Y$ is 1.

When both the networks conduct due to a fault, $S_1 = S_0$
In such a case it is assumed that the nMOS network dominates over the pMOS network.

Finally, \( S1 = S0 = 0 \) implies that both the networks are non-conducting. This situation can arise when a stuck-open fault is present. In such a case the output of the faulty gate remains in the floating state, and the logic value at the output depends on the previous logic value. This introduces memory and is denoted by \( M \).

Consider a FCMOS gate shown in Fig. 4.1. Whether a FET in \( P \) net (or \( N \) net) is in a conducting state or not is determined by its control variable. For example, transistor \( T1 \) in \( P \) net (\( N \) net) is in a conducting state if its control variable \( \bar{a}(a) = 1 \), otherwise \( T1 \) is in a non-conducting state.

We define that all control variables along a path from \( Vdd \) to output in \( P \) net and from output to \( Vss \) in \( N \) net is called the conducting condition of the path. The conducting condition of all paths is called the conducting condition of \( P \) net (\( N \) net).

For example, the conducting condition of \( P \) net in Fig. 4.1 is

\[
fp = \bar{a} \bar{b} \bar{e} + \bar{a} \bar{d} + \bar{c} \bar{b} \bar{d} + \bar{c} \bar{e}
\]

Similarly, the conducting condition of \( N \) net is,

\[
fn = abe + ac + cbd + de
\]

Obviously, the conducting of \( P \) net or \( N \) net is an AND and OR expression, and \( fp = \bar{fn} \).
Fig. 4.1 A FCMOS Gate.
Fig. 4.2 A Gate Equivalent Circuit of Fig. 4.1.
Using the conducting condition of $P$ net ($N$ net), the CMOS combinational circuit can be described as a gate equivalent circuit. A gate equivalent circuit of Fig. 4.1 is shown in Fig. 4.2, and the function of the block B is shown in Table 4.1.

4.1.2 Simplified Model

Now, we simplify gate-level model for static CMOS circuits discussed in the previous section. The model does not need a memory block such as the B-block [10]. When $T_2$ test pattern is applied to the faulty gate, the gate output floats and maintains the previous output of the gate. However, if the gate output is properly initialized by a $T_1$ pattern before applying $T_2$, the logic value of the gate output under $T_2$ can be obtained. This knowledge enables us to eliminate the memory element. Furthermore, only the pMOS network or the nMOS network is modeled, but not both. In addition, a non-series-parallel network can also be modeled by this method.

We next present the method which lets us derive the equivalent gate-level model from the nMOS networks of the CMOS gates in the circuit. This method can be easily modified to be applicable to the pMOS networks. The method has the following two procedures:

Procedure 1:

Apply steps given below to the nMOS network of each gate
in the given CMOS circuit.
Step 1: Replace each series or parallel connection of nMOS transistors in the nMOS network with a single nMOS transistor.
Step 2: Repeat Step 1 until no further reduction is possible.
Step 3: Label all nodes of the reduced network by integers and the inputs of the nMOS transistors by letters.
Step 4: Find every loop-free conduction path from Vdd to the output node and express it as a product of the labels of the nMOS transistors lying on it.
Step 5: Derive the gate function Gf as a sum-of products expression in which the products are from Step 4.
Procedure 2:
   Apply the steps given below to the reduced networks in procedure 1.
Step 1: For every nMOS transistor in the reduced network, whose input is say Z, derive an equivalent circuit of the part of the nMOS network represented by the index set of Z as follows: replace a series (parallel) connection of transistors by AND (OR) gates and complement the inputs.
Step 2: By employing AND gates and an OR gate, combine the outputs of the circuits derived in Step 1 to realize the gate Gf.
Consider again the example shown in Fig. 4.1. We first derive the equivalent gate-level model from its nMOS network. The reduced network that can be obtained from this network is shown in Fig.4.3. The set of all loop-free conduction paths
from Vdd to f is \(\{1, 3, 4, 5, 4, 2, 3, 1, 2, 5\}\). Thus \(G_f = AC + DE + DBC + ABE\). The final gate-level model is shown in Fig. 4.4. This is the gate equivalent corresponding to \(N\) nets of the circuit in Fig. 4.1.

In this thesis, we select the n-type network to describe an equivalent gate level circuit. When an n-type network is used, an invertor is necessary due to the pull down operation of the n-type net.

Once an equivalent gate level circuit is constructed, the next problem is how to represent the SOP faults in the equivalent circuit.

4.2 Fault Equivalence Between SOP Fault and Stuck-at Fault

Definition: Two faults, say \(a\) and \(b\), are called equivalent if and only if any test detecting fault \(a\) always detects fault \(b\) and vice versa.

In this section, we investigate the fault equivalence between a SOP fault and a stuck-at fault for primitive logic gates such as NAND, NOR, AND, OR and invertor. 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 \(a\). Suppose that the faulty gate output is properly initialized to logic 0. In order to detect \(a\), 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
Fig. 4.3 The Reduced Network (n net).
Fig. 4.4 The Gate Equivalent of N net in Fig. 4.2.
the gate, say fault b. Obviously, the condition detecting a is
the same as the one required to detect b. Hence, the test
detecting a also detects b and vice versa. This means that a
is equivalent to b provided that the faulty gate output is
properly initialized. We say that a is potentially equivalent
to b 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 and 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 only primitive logic
gates.

Let us consider an n-input NAND gate again. Stuck-open
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. 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 invertor are one s-a-1 and one s-a-0 faults. 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 [25].

In summary, a potentially equivalent fault of an n-type transistor SOP fault of a COMS gate is the line stuck-at-0 (s-a-0) fault on the input line corresponding to the faulty transistor. Similarly, a potentially equivalent fault of a p-type transistor SOP fault is the stuck-at-1 (s-a-1) fault on the corresponding input line.

Consider the example of NAND gate given in Fig. 4.5, Table 4.2 shows the Fault equivalent between SOP fault and stuck-at faults of this circuit.
4.3 Detection of SOP Fault Using Stuck-At Faults

In section 3.2, various test derivation algorithms for detecting FCMOS SOP faults were discussed. As was shown, some of the early approaches attempted to use the stuck-at test set to test SOP faults. In this section, previous studies on CMOS SOP fault testing using stuck-at test sets are described.

Consider a CMOS circuit consisting of only primitive logic gates. As shown in last section, 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 T2 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 T1 pattern for the SOP fault. Hence, at least one T1 pattern and one T2 pattern for any SOP fault are included in the stuck-at test set. This implies if the test patterns detecting the stuck-at faults are properly initialized, they can detect SOP faults under the assumption of zero gate delays. The test pattern sequences which are generated by test pattern generator can not guarantee detecting all SOP fault. For example, the test sequence in Table 1.1 can not detect SOP faults f1 and f2 in Fig. 1.2. We will discuss this case in detail in next chapter.

In summary, if an input of a CMOS gate is s-a-0 (s-a-1),
Fig. 4.5 A 2-Input CMOS NAND Gate.
it is equivalent to the n-type (p-type) transistor SOP fault connected to the input line assuming that the faulty gate is properly initialized. This implies that if a test pattern $T_2$ detects a potentially equivalent stuck-at fault, it also detects the corresponding SOP fault provided that an initialization pattern $T_1$ is applied prior to the application of the test pattern. If the structure of a CMOS gate is dual and fully complementary, there is always a corresponding p-type transistor for an n-type transistor. Hence, SOP faults on the p-type network are represented by s-a-1 faults in the n-type equivalent network.

Table 4.2 Fault Equivalent Between SOP Fault and Stuck-at Fault of the Circuit in Fig. 4.5.

<table>
<thead>
<tr>
<th>SOP Fault</th>
<th>Potentially Equivalent Stuck-at Fault</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOP At P1</td>
<td>S-A-1 On Line 1</td>
</tr>
<tr>
<td>SOP At P2</td>
<td>S-A-1 On Line 2</td>
</tr>
<tr>
<td>SOP At N1</td>
<td>S-A-0 On Line 1</td>
</tr>
<tr>
<td>SOP At N2</td>
<td>S-A-0 On Line 2</td>
</tr>
</tbody>
</table>

4.4 SOP Fault Collapsing

Up to now, we have explained a mapping procedure of a SOP fault into a stuck-at fault. Here, we discuss about the reduction of the number of faults which are to be considered in order to test the circuit. Two stuck-at faults are equivalent if any test pattern detecting one fault also
detects the other fault. Similarly, two SOP faults are equivalent if any \((T_1, T_2)\) pair detecting one fault also detects the other fault. For two SOP faults in a CMOS circuit, if \(T_2\) patterns of the two gates are same, then it can be easily seen that \(T_1\) patterns are also the same. From the above discussion, we conclude that, two SOP faults in a CMOS gate are equivalent if and only if the potentially equivalent stuck-at faults are equivalent. Therefore, we just consider the number of collapsed faults in circuits as the number of faults to be detected.
In the previous chapters, we presented the methods which present how to convert a CMOS circuit under test into an equivalent gate level circuit and transform SOP faults into the equivalent stuck-at faults. In this chapter, we will present the SOP fault simulator which simulates the SOP faults. The test patterns can be generated by random pattern generator or different LFSR circuits. We can use the software to evaluate different LFSR design for their two-pattern capability. The method can be also expanded to different autonomous linear sequential circuits. Test patterns will be applied to the circuit under test in pairs, and the stuck-open faults in circuit will be simulated. The simulator is based on the parallel pattern single fault propagation [32]. We assume time-delay is zero in this thesis.

5.1 Test Patterns Generated By LFSRs

In our software, the test vectors can be generated by random pattern generator or different LFSR circuits. We now emphasized the LFSR circuits.

Because of relatively low overhead and short testing time [26-29], most built-in self testing (BIST) sachenms employ
Fig. 5.1 The General Form of an LFSR.
linear feedback shift registers (LFSRs) as the test pattern generator to apply exhaustive or random tests for circuit under test. This has been true for stuck-at faults in combinational circuits [26].

A linear feedback shift register (LFSR) is a linear sequential network for generating sequences of a given cycle length (period). The LFSR is composed of interconnections of unit-delays (D flip-flops) and modula-2 adders (Exclusive-OR gates) [26-29].

The LFSR has been used in many different applications. Examples of these applications are pseudo-random number generator, signature analyzer, in-circuit test pattern generator, and store address generator. In this paper, we use LFSRs as the test pattern generators.

Fig. 5.1 shows a general form of an n-stage LFSR whose internal structure can be defined by a characteristic polynomial \( f(x) \),

\[
    f(x) = 1 + h_1x + h_2x^2 + \ldots + h_{n-1}x^{n-1} + x^n
\]

If \( h_i \) is equal to one, then the corresponding \( x^i \) is connected; otherwise, it is not connected.

Now, we briefly describe the design of LFSRs [26-29]. As described in [12], assume \((n, w)\) be a n-input circuit, in which \( w \) is the maximum weight (maximum number of inputs that an output depends on), where \( w \leq n \). In order to test the \((n, w)\) CUT an n-stage LFSR of period \( T = 2^k \cdot 1 \) is necessary, such that any \( w \) combinationals of the LFSR contain all \( 2^w \) distinct test
patterns. The \((n,k)\) LFSR must have the smallest integer \(k\) \((w \leq k \leq n)\) given by:

\[
  w \leq \left\lfloor \frac{k}{n-k+1} \right\rfloor + \left\lceil \frac{k}{n-k+1} \right\rceil
\]

where

\([x]\) denotes the smallest integer greater than or equal to \(x\),

and

\([x]\) denotes the largest integer smaller than or equal to \(x\).

The characteristic polynomial of an \((n,k)\) LFSR is given by

\[
f(x) = g(x)p(x)
\]

where \(p(x)\) is any primitive polynomial of degree \(k\) and \(g(x)\) is any monic polynomial of degree \(n-k\). Also, the \((n,k)\) LFSR requires that the initial state \(S_0(x)\) of the LFSR, or seed, be divisible by \(g(x)\). Thus, for simplicity, \(S_0(x)\) could be taken as \(g(x)\).

We can select different primitive polynomial \(p(x)\) and different monic polynomial \(g(x)\) to design different structures of LFSRs. The test patterns generated by these LFSRs will be applied to the circuit under test to study stuck-open fault detection capabilities of LFSRs.
5.2 Simulation Methodology

In the simulator, table-driven technique is used for the description of the logic circuit to be simulated. The circuit is stored in tables (circuit files) in the host computer. The circuit files are created in ISCAS85 format [30]. The tables are accessed by the simulation program as necessary. The simulation progress is circuit-independent, a new set of tables is required for each distinct circuit to be simulated.

The first step in simulation is initialization, as discussed in previous chapters, the simulator transforms the given circuit into the gate level circuit, stuck-open faults into stuck-at faults and set up a fault list.

As described in last section, test patterns will generated by LFSR circuit. We can design different LFSR circuits and determine the number of test patterns (clock cycles) needed for certain fault coverage.

The test patterns are applied to the circuit, and the circuit is simulated. The first pair, which composes the first two patterns generated by LFSR, is applied to the circuit. If the second pattern of the first pair detects a primary fault \( f_i \). The fault is marked as "T2 is found". Then if the fault \( f_i \) can be initialized by the first pattern of the first pair test pattern using fault free responses of the circuit is mark as "T1 is found". A primary fault which is marked as "T1 is found" and "T2 is found" is eliminated from the fault list.
The second pair of test pattern will be taken as the second pattern and the third pattern, and the fault simulation is repeated for this pair. The simulation terminates if either the fault list is empty or test patterns (the clock cycle) are exhausted.

The simulators are employed to simulate logic circuits which are assumed in a zero-delay model, and race and hazard conditions are ignored. The simulator is based on the parallel pattern single fault propagation technique.

5.3 Parallel Fault Simulation

A computer has several bit-oriented instructions, including logical instructions such as the AND, OR, Exclusive OR, and NOT operations. During execution of instructions from this group the several bits of a single computer word are manipulated identically and independently. The AND, OR, and Exclusive OR instructions each operate on a pair of operands; the \( i \text{th} \) bits of each operand are combined to give the \( i \text{th} \) bit of the result. For example, the OR operation of 4-bit words is processed as follows:

\[
[1010].\text{OR.}[1100]=[1110]
\]

Each \( i \text{th} \) bit of two words is simultaneously ORed, and the resultant value is stored in the \( i \text{th} \) bit of the third word. Several advantages accrue from adopting a bit-oriented view of the contents of computer words instead of the one-symbol-per-
word view. One advantage is that memory can be used with great efficiency if many items per word, instead of just one, are stored. Another advantage is that parallel computation is possible because there are instructions that operate on all the bits of a word concurrently.

Fault simulation based on this parallel process of bit-oriented operations is called parallel fault simulation. In parallel fault simulation, if a word or a string has \( n \) bits, then \( n \) different problems can be processed in parallel. If we want to simulate a circuit for \( m \) different faults, then \( \lceil m/n \rceil \) passes must be made with \( n \) faults being simulated during each passed, when \( \lfloor x \rfloor \) is the smallest integer equal to or greater than \( x \).

In the fault simulation, a logical effect of a fault is injected onto the computation of the faulty element. This procedure is called fault injection. Two masks, \( \text{mask}(S) \) and \( \text{fvalue}(S) \), associated with each signal line \( S \) are used to inject faults. Each fault injected on \( S \) corresponds to a unique bit position \( i \) in \( \text{mask}(S) \) and \( \text{fvalue}(S) \). This bit position must be different from the bit positions on the corresponding masks of any other remaining \( n-1 \) faults being simulated in parallel with the fault on \( S \). These masks are defined as follows: \( \text{mask}(S)_i = 1 \) if a fault exists on the signal line \( S \) when the \( i^{th} \) bit of the computer word is simulated, \( \text{mask}(S)_i = 0 \) otherwise; \( \text{fvalue}(S)_i = 1 \) if the fault on \( S \) is stuck-at-1, \( \text{fvalue}(S)_i = 0 \) if the fault on \( S \) is stuck-at-0, where the
subscript $i$ denotes the bit position.

These two masks are used to inject faults into each bit of the word $S$, and the masked word $S'$ is formed as follows (Fig. 5.2):

$$S' = S \cdot \text{mask}(S) + \text{mask}(S).fvalue(S)$$

where $\cdot$, $+$, and $-$ mean bit-oriented logical operations AND, OR, and NOT, respectively.

The masks used for fault injection are generated during the preprocessing of the circuit for fault simulation. Except for the overhead required to generate the masks and carry out fault injection, $n$ faults can be processed almost as rapidly as one fault.

Consider a two-input NAND gate and the four faults associated with the gate Fig. 5.3. Faults are specified in Table 5.1, and the values of the masks associated with signal lines $A$, $B$, and $C$ are defined as shown in Table 5.2.

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Fault</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Fault Free</td>
</tr>
<tr>
<td>2</td>
<td>$S$-$A$-$0$ ON $A$</td>
</tr>
<tr>
<td>3</td>
<td>$S$-$A$-$1$ ON $B$</td>
</tr>
<tr>
<td>4</td>
<td>$S$-$A$-$0$ ON $C$</td>
</tr>
<tr>
<td>5</td>
<td>$S$-$A$-$1$ ON $C$</td>
</tr>
</tbody>
</table>
Table 5.2

<table>
<thead>
<tr>
<th>Line</th>
<th>Mark</th>
<th>Fvalue</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>[01000]</td>
<td>[00000]</td>
</tr>
<tr>
<td>B</td>
<td>[00100]</td>
<td>[00100]</td>
</tr>
<tr>
<td>C</td>
<td>[00011]</td>
<td>[00001]</td>
</tr>
</tbody>
</table>

For example, mask(B) = [00100] since the bit position of the fault associated with line B is bit 3, and fvalue(B) = [00100] because the fault is stuck-at-1 and is processed in bit 3. Similarly, since two faults associated with line C (C s-a-0 and C s-a-1) must be injected into bits 4 and 5 of word C, mask(C) = [00011] and fvalue(C)₄ = 0 and fvalue(C)₅ = 1.

After the masks for fault injection have been generated, parallel fault simulation is carried out. Suppose we want to simulate an input pattern with A = 1 and B = 0. First the words A and B are initialized to A = [11111] and B = [00000]. The simulation is carried out from primary inputs toward primary outputs. To evaluate properly the logic value of each element, all elements of the circuits of the circuit should be ordered or levelized. The details of levelization will be described later. In Fig 5.4, A and B are masked to form A' and B':

\[
A' = A \text{ mask}(A) + \text{mask}(A) \text{ fvalue}(A) \\
= [11111] [01000] + [01000] [00000] \\
= [11111] [10111] + [00000] \\
= [10111] \\
B' = B \text{ mask}(B) + \text{mask}(B) \text{ fvalue}(B)
\]
Fig. 5.2 Model of Fault Injection.

Fig. 5.3 Parallel Fault Simulation.
Next the NAND gate is simulated:

\[ C = \overline{A' \cdot B'} \]
\[ = \overline{[10111] \cdot [00100]} \]
\[ = [00100] \]
\[ = [11011] \]

Then \( C' \) is masked:

\[ C' = C \text{ mask}(C) + \text{mask}(C) \cdot \text{fvalue}(C) \]
\[ = \overline{[11011] \cdot [00011]} + [00011] \cdot [00001] \]
\[ = [11000] + [00001] \]
\[ = [11001] \]

On the output \( C' \), bits 3 and 4 are different from bit 1 of the fault-free value, and hence the faults corresponding to the bits 3 and 4, that is \( B \text{ s-a-1} \) and \( C \text{ s-a-0} \), are detectable.

If the change of the logic value of a line \( s \) is observable at a line \( t \), the line \( s \) is said to be detectable at line \( t \). We define an event as a change in value of a single line. The output of a logic element will change value only when one of more or its inputs have changed in the preceding time interval. Hence, an element need be simulated only when an event occurs at one of its inputs. When an event occurs, event element to which this line fans out has the possibility of a new event. We only need to simulate those potentially
active elements.

5.4 Stuck-Open Fault Simulator and Optimal Design of LFSR For BIST Implementation

Fault simulation is essential for measuring the fault coverage of the given test sets. Fault simulation is also employed for analyzing the circuits behavior. We implemented a SOP fault simulator which is to simulate stuck-open faults in CMOS circuits in BIST environment. In this section, we present the SOP fault simulator. The simulator is written in C and runs on an Sun workstation.

To measure the stuck-open fault coverage and evaluate the two-pattern capabilities of LFSRs, we have modified the stuck-open test pattern generator developed at Virginia Polytechnic & State University [7] to add stuck-open fault simulation capability to the software. We also added hardware based test pattern generation subroutines to the program. The simulator is based on the parallel single fault propagation.

The simulator is based on the parallel single fault propagation. The essential idea of the simulator is to simulate the circuit in the forward levelized order and to prune off unnecessary gates in the early stages. In this way, the simulator performs fault simulations only for the gates which are affected by the injected faults.

As explained before, there are two different approaches
for modeling the SOP fault, gate 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. A SOP fault is represented by its equivalent stuck-at fault. When a test pair is applied to the circuit, the simulator matches the pattern with a T2 pattern detecting the fault. If the pattern is a T2 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 simulation preserves the previous logic values of all gates. And then T1 is applied to the circuit to initiate the circuit. If a fault is detected by T2 and initiated by T1, then the fault is detected.

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 before, 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 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 simulation is the parallel simulation scheme. Suppose that a transistor in a CMOS combinational
Fig. 5.4 The Flow Chart of the Simulator.
circuit is stuck-open. When a T2 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. Using the single (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 circuit concurrently. The flow chart of the simulator is shown in Fig.5.4.

We now 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 values 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 equivalent faults are collapsed using simple fault collapsing techniques described before. In the second step, the test pattern is generated by LFSRs. In the third step, test pattern pairs are 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 fourth step, all detected faults are identified and eliminated from the fault list, the initialization of the faulty gate output is checked. 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 next test pattern pair is set as the current test pattern and the fault simulation is repeated. The above procedure is repeated until test patterns (clock cycles) are exhausted. The procedure is described below.

PROCEDURE SOP_FAULT_SIMULATION:

Step 1: (initialization)

Transform the given circuit into the gate level circuit.
Set up fault list (FL)
Set all the logic values into x (don't care).
{Set the first test pattern as the current test pattern.}
Set up initial time.
Step 2: (This procedure generates the test patterns by LFSR circuits)
Generate the test patterns by LFSRs.
Set up the initial seed of the LFSR.

Step 3: (This procedure simulates the circuit assuming zero gate delays and tags all the detectable faults.)
If all faults are detected, GOTO step 4.
If the clock cycle is zero, GOTO step 4.
If test patterns > max_test_pattern, then stop.
Perform fault simulation for the circuit.
FOR every fault $f_i$ in FL DO
  If the test pattern $T_2$ detects $f_i$,
    mark "fi is detected by T2".
END FOR
FOR every fault $f_i$ marked as "fi is detected by T2"
  If the test pattern $T_1$ initiates $f_i$,
    mark "fi is initiated by T1".
  Tag the detected faults.
  Eliminate $f_i$ if both $T_1$ and $T_2$ are found.
END FOR
Set up simulation time.

Step 4: (This procedure identifies all the detected faults)
Update the fault_coverage.
Set up running time.
Stop.
In the following, we present the evaluation procedure of LFSRs. The procedure is divided into 4 steps. The first step is to set up LFSR circuit parameter. In this step, parameters of the LFSRs are set up. In the second step, the test patterns which will be applied to the \((n,w)\)CUT is generated by \((n,k)\)LFSR. In the third step, test pattern pairs are applied to the circuit and the circuit is simulated assuming zero gate delays. The procedure is repeated until all the faults are detected or the clock cycles are exhausted. In the fourth step, the LFSRs are evaluated. The LFSR which has the best two-pattern capability is selected for realizing the final circuit. The procedure is described below.

PROCEDURE LFSR-OPTIMIZATION:

Step 1: (set up the parameters of LFSR circuit)
Determine the maximum output weight, \(w\).
Determine the degree of primitive polynomial \(p(x)\), \(k\).
Determine the degree of monic polynomial \(g(x)\), \(n-k\).

Step 2: (create LFSR circuit)
Create LFSR Circuit \(f(x) = p(x)g(x)\).
Set up the initial seed of the LFSR.
Set up the clock cycles.
Set up initial time.

Step 3: (This procedure simulates the circuit assuming zero gate delays)
If all faults are detected, GOTO step 4.
If the clock cycle is exhausted, GOTO step 4.
If test patterns > max_test_pattern, then stop.
Perform stuck-open fault simulation for the circuit.
Record simulation time.

Step 4: {This procedure checks the stop condition.}
If the desired fault_coverage using a specified number of clock cycles is reached, stop.
Select new p(x), g(x), or clock cycles, GOTO step 2.

In the next chapter, we will use the proposed simulator to run some simple and benchmark circuits, and present experimental results.
CHAPTER SIX
EXAMPLES

The proposed simulator, sopsim.ohu, is implemented for the SOP fault simulation of FCMOS circuits which consist of only primitive logic gates with zero gate delays. The simulator generates test patterns by random pattern generator or different LFSR designs. In this chapter, we present some examples which use sopsim.ohu to simulate some simple circuits and 1985 ISCAS benchmark circuits.

6.1 Simple Examples

Example 1:

First, we consider a very simple circuit, a CMOS NAND gate, given in Fig. 6.1 which is a FCMOS NAND gate. Following is the procedure to simulate this circuit:

(1) Create the circuit under test file (CUT) file.
   (See Appendix 1)
   
   1 1gat inpt 1 0
   2 2gat inpt 1 0
   3 3gat nand 0 2
   1 2

(2) From equation 5.1, k = 2 for this circuit.

(3) Select primitive polynomial \( p(x) = x^2 + x + 1 \)
Fig. 6.1 A 2-input CMOS NAND Gate.
(4) Select monic polynomial \( g(x) = 1 \)

(5) Implement the LFSR circuit \( f(x) = p(x)g(x) = x^2 + x + 1 \)

\[ \text{Take } S_0(x) = g(x) \]

(6) The period of this LFSR is \( T = 2^k - 1 = 3 \),

We take clock cycle as 4. There are 3 pairs of test patterns are generated.

Fig. 6.2 shows the LFSR circuit applied to the NAND gate.

File 6.1 is the log file of the simulation result.

Because the test patterns \((11, 01)\) which can detect stuck-open fault at pi never apply to the circuit, \( f_1 \) can not be detected.

If a 3-stage LFSR is used (Fig. 6.3), all stuck-open faults in NAND can be detected. Because of the dependencies of parallel patterns generated by LFSRs, two-pattern testing capabilities of LFSRs strongly depends on the choice of the LFSR design.

Example 2:

Consider the ISCAS85 benchmark circuit c17.isc which includes 6 NAND gates (Fig. 6.3). The simulation procedure is the same as described in example 1. We use different LFSR designs in this example. The results are presented in Table 6.1. Form the Table 6.1, we noticed that the combination of \( p(x) = x^4 + x^3 + 1 \) and \( g(x) = x + 1 \) results in the best fault coverage. This means that this LFSR has the best two-pattern capability. The circuit file and log file are presented in File 6.2 and File 6.3.
Fig. 6.2 A LFSR Circuit Applied to The NAND Gate.
Fig. 6.3 A 3-stage LFSR Circuit Applied to The NAND Gate.
**Name of circuit: Fig. 6.3**

**Primary inputs:**
1  2

**Primary outputs:**
3

-The circuit has 2 inputs.

-Creat the test pattern generator (LFSR):

---What is the degree of primitive polynomial p(x): 2

---Please input a primitive polynomial p(x) of degree 2
2 1 0

---Please input a monic polynomial g(x) of degree: 0
0

---How many clock cycles (test pattern applied): 4

-Test pattern pairs, fault free responses,
* and number of SOP faults detected by each test pair:

  Initial seed : 10 1
  test pair 1: 01 1 0 SOP faults detected
  test pair 2: 11 0 1 SOP faults detected
  test pair 3: 10 1 1 SOP faults detected

-End of SOP faults simulation.

****** SUMMARY OF SOP FAULTS SIMULATION RESULTS ******

1. Circuit structure
   Number of gates : 3
   Number of primary inputs : 2
   Number of primary outputs : 1
   Depth of the circuit : 2

2. Simulator input parameters
   Simulation mode : LFSR

3. Simulation results
   Number of test pairs applied : 3
   Fault coverage : 66.667 %
   Number of collapsed faults : 3
   Number of detected faults : 2
   Number of undetected faults : 1

4. CPU time
   Initialization : 0.117 secs
   Fault simulation : 0.067 secs
   Total : 0.183 secs

* List of undetected faults:
  1. Input line 1 s-a-1 of gate 3

File 6.1 Log File of the Simulation Results of Fig. 6.3.
Fig. 6.4 Gate Level Circuit of c17.isc.
* c17.isc circuit file
*--------------------------------------------------
* 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
*
1 1gat inpt 1 0
2 2gat inpt 1 0
3 3gat inpt 2 0
8 8fan from 3gat
9 9fan from 3gat
6 6gat inpt 1 0
7 7gat inpt 1 0
10 10gat nand 1 2
11 11gat nand 2 2
14 14fan from 11gat
15 15fan from 11gat
16 16gat nand 2 2
2 14
20 20fan from 16gat
21 21fan from 16gat
19 19gat nand 1 2
15 7
22 22gat nand 0 2
10 20
23 23gat nand 0 2
21 19

File 6.2 The Circuit File of c17.isc.
* Name of circuit:  c17.isc

* Primary inputs :
  1  2  3  6  7

* Primary outputs:
  22  23

*The circuit has 5 input.

*Create the test pattern generator (LFSR):

--What is the degree of primitive polynomial p(x):4

--Please input a primitive polynomial p(x) of degree 4
  4 3 0

--Please input a monic polynomial g(x) of degree:1
  1  0

--How many clock cycles (test pattern applied):16

*Test pattern pairs, fault free responses,
*and number of SOP faults detected by each test pair:

Initial seed : 11000 00
test pair 1: 01100 11  3 SOP faults detected
test pair 2: 00110 00  2 SOP faults detected
test pair 3: 00011 01  3 SOP faults detected
test pair 4: 11011 11  1 SOP faults detected
test pair 5: 10111 10  3 SOP faults detected
test pair 6: 10011 01  1 SOP faults detected
test pair 7: 10010 00  1 SOP faults detected
test pair 8: 01001 11  0 SOP faults detected
test pair 9: 11110 10  1 SOP faults detected
test pair 10: 01111 00  1 SOP faults detected
test pair 11: 11101 11  0 SOP faults detected
test pair 12: 10100 10  1 SOP faults detected
test pair 13: 01010 11  0 SOP faults detected
test pair 14: 00101 01  0 SOP faults detected
test pair 15: 11000 11  0 SOP faults detected

End of SOP faults simulation.

****** SUMMARY OF SOP FAULTS SIMULATION RESULTS ******

1. Circuit structure
   Number of gates : 11
   Number of primary inputs : 5
   Number of primary outputs : 2
   Depth of the circuit : 4

File 6.3 Log File For the Circuit c17.isc.
2. Simulator input parameters
   Simulation mode : LFSR

3. Simulation results
   Number of test pairs applied : 15
   Fault coverage : 94.444 %
   Number of collapsed faults : 18
   Number of detected faults : 17
   Number of undetected faults : 1

4. CPU time
   Initialization : 0.167 secs
   Fault simulation : 0.033 secs
   Total : 0.200 secs

* List of undetected faults:
  1. Input line 1 s-a-1 of gate 22

File 6.3 Continued.
Table 6.1. Simulation Results of c17.isc. \((n = 5, w=4, k = 4)\)

(with 16 clock cycles, 15 test pairs)

<table>
<thead>
<tr>
<th>(g(x))</th>
<th>(p(x) = x^4 + x + 1)</th>
<th>(p(x) = x^4 + x^3 + 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fault Coverage (%)</td>
<td>CPU Time (secs)</td>
</tr>
<tr>
<td>(g(x) = x)</td>
<td>88.889</td>
<td>0.213</td>
</tr>
<tr>
<td>(g(x) = x + 1)</td>
<td>88.889</td>
<td>0.183</td>
</tr>
</tbody>
</table>
6.2 Benchmark Circuits

In this section, we have applied our method to study several ISCAS85 benchmark circuits. The simulation results which are the results of different LFSRs and/or different clock cycles are given in Tables 6.2, 6.3, 6.4, and 6.5. In Table 6.2, eight different LFSRs for ISCAS85/c880.isc are simulated, the combination of \( p(x) \) and \( g(x) \) which has the best fault coverage will be selected. We have applied 10,000 clock cycles (9,999 pattern pairs) to this circuit. Note that if we apply this clock with 10 MHZ clock rate, it will take 0.1 msec. to apply all the tests. Also the number of test patterns applied is a small fraction of the pseudo-exhaustive testing which needs \( 2^{59} \) for this circuit.

In Table 6.3, 6.4, and 6.5 different clock cycles are applied for LFSR circuits for ISCAS85/c432.isc, c1908.isc, and c1355.isc. The LFSR circuit which reaches desired fault-coverage using a specified number of clock cycles may be selected for final BIST realization.

A summary of experimental results are given in Table 6.6. The results are the LFSRs with best fault coverage in each of the tables (Table 6.1 to Table 6.5). The fault coverage is compared with the fault coverage which uses the test patterns generated by soprano. It should be pointed out that soprano is a software test pattern generator to minimize the number of test pairs. One will need a sophisticated test station to
apply the pattern to the CUT. The LFSR design is a built-in self-test environment and the test patterns will be generated internally and an expensive test station is not needed. From Table 6.6, the average stuck-open fault coverage using soprano is 98.80%, while the average of stuck-open fault coverage using designed LFSRs is 96.02%. There are only 2.78% difference between them. It is noticed that with 10000 clock cycles, the fault coverage is at least 92.85% for these circuits. We have observed that, the more LFSR circuits we try, the better results we achieve, and the more clock cycles we apply, the better fault coverage targets we get.
Table 6.2. Simulation Results of ISCAS85/c880.isc

(ISCAS85/c880.isc consists of 383 gates. There are 1112 collapsed stuck-open faults in circuit. $n = 60$, $w = 45$, $k = 59$)

(with 10,000 clock cycles, 9,999 test pairs)

<table>
<thead>
<tr>
<th>$p(x)$</th>
<th>$g(x) = x + 1$</th>
<th>$g(x) = x$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fault Coverage</td>
<td>CPU Time (secs)</td>
</tr>
<tr>
<td>$x^{59}+x^{22}+x^{21}$ $+x+1$</td>
<td>92.373%</td>
<td>40.133</td>
</tr>
<tr>
<td>$x^{59}+x^{58}+x^{37}$ $+1$</td>
<td>97.033%</td>
<td>44.267</td>
</tr>
<tr>
<td>$x^{59}+x^{6}+x^{5}+x^{4}$ $+x^{3}+x+1$</td>
<td>93.157%</td>
<td>46.217</td>
</tr>
<tr>
<td>$x^{59}+x^{58}+x^{56}+x^{55}+x^{54}$ $+x^{53}+1$</td>
<td>94.195%</td>
<td>39.281</td>
</tr>
</tbody>
</table>

Table 6.3. Simulation Results of ISCAS85/c432.isc

(ISCAS85/c432.isc consists of 160 gates. There are 514 collapsed stuck-open faults in circuit. $n = 36$, $w = 36$, $k = 36$)

(with $g(x) = 1$)

<table>
<thead>
<tr>
<th>Clock Cycles</th>
<th>$p(x) = x^{36} + x^{11} + 1$</th>
<th>$p(x) = x^{26} + x^{25} + 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fault Coverage</td>
<td>CPU Time (secs)</td>
</tr>
<tr>
<td>1000</td>
<td>90.856%</td>
<td>6.933</td>
</tr>
<tr>
<td>2000</td>
<td>96.887%</td>
<td>9.017</td>
</tr>
<tr>
<td>4000</td>
<td>98.249%</td>
<td>12.950</td>
</tr>
<tr>
<td>6000</td>
<td>98.638%</td>
<td>14.767</td>
</tr>
<tr>
<td>10000</td>
<td>98.783%</td>
<td>38.357</td>
</tr>
</tbody>
</table>
Table 6.4. Simulation Results of ISCAS85/c1908.isc
(ISCAS85/c1908.isc consists of 880 gates. There are 2378 collapsed stuck-open faults in circuit. \( n = 33, \ W = 33, \ k = 33 \))

(with \( g(x) = 1 \))

<table>
<thead>
<tr>
<th>Clock Cycles</th>
<th>( p(x) = x^{33} + x^{13} + 1 )</th>
<th>( p(x) = x^{33} + x^{20} + 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fault Coverage</td>
<td>CPU Time (secs)</td>
</tr>
<tr>
<td>2000</td>
<td>88.141%</td>
<td>30.050</td>
</tr>
<tr>
<td>5000</td>
<td>90.917%</td>
<td>65.633</td>
</tr>
<tr>
<td>10000</td>
<td>92.851%</td>
<td>100.417</td>
</tr>
</tbody>
</table>

Table 6.5. Simulation Results of ISCAS85/c1355.isc
(ISCAS85/c1355.isc consists of 546 gates. There are 1610 collapsed stuck-open faults in circuit. \( n = 41, \ W = 41, \ k = 41 \))

(with \( g(x) = 1 \))

<table>
<thead>
<tr>
<th>Clock Cycles</th>
<th>( p(x) = x^{41} + x^{3} + 1 )</th>
<th>( p(x) = x^{41} + x^{38} + 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fault Coverage</td>
<td>CPU Time (secs)</td>
</tr>
<tr>
<td>2000</td>
<td>91.118%</td>
<td>33.100</td>
</tr>
<tr>
<td>4000</td>
<td>94.596%</td>
<td>47.933</td>
</tr>
<tr>
<td>10000</td>
<td>96.121%</td>
<td>58.826</td>
</tr>
</tbody>
</table>
Table 6.6 Experimental Results

<table>
<thead>
<tr>
<th>Circuit Names</th>
<th>n</th>
<th>Soprano (Software)</th>
<th>LFSRs (Hardware)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>No. of Test Patterns</td>
<td>Fault Coverage</td>
</tr>
<tr>
<td>c17.isc</td>
<td>5</td>
<td>13</td>
<td>100%</td>
</tr>
<tr>
<td>c432.isc</td>
<td>36</td>
<td>144</td>
<td>96.12%</td>
</tr>
<tr>
<td>c880.isc</td>
<td>60</td>
<td>202</td>
<td>100%</td>
</tr>
<tr>
<td>c1355.isc</td>
<td>41</td>
<td>337</td>
<td>98.38%</td>
</tr>
<tr>
<td>c1908.isc</td>
<td>33</td>
<td>401</td>
<td>99.52%</td>
</tr>
<tr>
<td>Average</td>
<td>35</td>
<td>220</td>
<td>98.80%</td>
</tr>
</tbody>
</table>
CHAPTER SEVEN

CONCLUSION

Because of the sequential behavior of faulty circuits caused by stuck-open faults, test generation methods detecting CMOS SOP faults are complex and time consuming. The traditional line stuck-at fault model does not represent SOP faults properly in CMOS circuits. In practice, most testing still relies on stuck-at test sets to test CMOS combinational circuits.

In thesis, we presented a simulator which simulates the stuck-open faults in faulty circuits. The test pattern generator program developed at Virginia Polytechnic & State University is modified to develop stuck-open fault simulator in BIST design study environment. We also presented a method to evaluate the two-pattern testing capabilities of LFSRs circuits. The experimental results showed that good stuck-open fault coverage can be reached with short test time. SOP fault coverage is compared with that of pattern test sets generated by soprano.

The experimental results based on five bench mark circuits show that the average SOP fault coverage of test sets by LFSRs is 96.02%. The average SOP fault coverage of pattern test sets by soprano is 98.80%. It should be pointed out that soprano is a software test pattern generator to minimize the
number of test pairs. One will need a sophisticated test station to apply the pattern to the CUT. The LFSR design is a built-in self-test environment and the test patterns will be generated internally and an expensive test station is not needed.

The simulator is restricted to FCMOS circuits with zero delay. It can be expanded to different MOS circuits. The method can be also extended to different autonomous test pattern generator circuits.
References


25. Lee, R. J. and Ha, D. S., "Test Pattern Generation for


APPENDIX A. Circuit Format

The format of circuit descriptions in SOPSIM.OHU is the same as ISCAS85 benchmark circuits. For our purpose, no fault specification is described. The SOP fault list of the circuit is automatically created by the software.

File 6.2 in chapter 6 is one of the benchmark circuit. We describe the format using this circuit. The circuit is represented at the gate level and in flat form. 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. Five 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 function 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.
APPENDIX B. A Primitive Polynomial for Every Degree Through 100

For quick reference, we list primitive polynomials which have the fewest number of terms for degree through 100 [31]. Only the exponents are shown in the table. For example, the entry 4 3 2 1 0 represents the polynomial

\[ x^4 + x^3 + x^2 + x + 1 \]

<table>
<thead>
<tr>
<th>k</th>
<th>p(x)</th>
<th>k</th>
<th>p(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 0</td>
<td>32</td>
<td>32 28 27 1 0</td>
</tr>
<tr>
<td>2</td>
<td>2 1 0</td>
<td>33</td>
<td>33 13 0</td>
</tr>
<tr>
<td>3</td>
<td>3 1 0</td>
<td>34</td>
<td>34 15 14 1 0</td>
</tr>
<tr>
<td>4</td>
<td>4 1 0</td>
<td>35</td>
<td>35 2 0</td>
</tr>
<tr>
<td>5</td>
<td>5 2 0</td>
<td>36</td>
<td>36 11 0</td>
</tr>
<tr>
<td>6</td>
<td>6 1 0</td>
<td>37</td>
<td>37 12 10 2 0</td>
</tr>
<tr>
<td>7</td>
<td>7 1 0</td>
<td>38</td>
<td>38 6 5 1 0</td>
</tr>
<tr>
<td>8</td>
<td>8 6 5 1 0</td>
<td>39</td>
<td>39 4 0</td>
</tr>
<tr>
<td>9</td>
<td>9 4 0</td>
<td>40</td>
<td>40 21 19 2 0</td>
</tr>
<tr>
<td>10</td>
<td>10 3 0</td>
<td>41</td>
<td>41 3 0</td>
</tr>
<tr>
<td>11</td>
<td>11 2 0</td>
<td>42</td>
<td>42 23 22 1 0</td>
</tr>
<tr>
<td>12</td>
<td>12 7 4 3 0</td>
<td>43</td>
<td>43 6 5 1 0</td>
</tr>
<tr>
<td>13</td>
<td>13 4 3 1 0</td>
<td>44</td>
<td>44 27 26 1 0</td>
</tr>
<tr>
<td>14</td>
<td>14 12 11 1 0</td>
<td>45</td>
<td>45 4 3 1 0</td>
</tr>
<tr>
<td>15</td>
<td>12 1 0</td>
<td>46</td>
<td>46 21 20 1 0</td>
</tr>
<tr>
<td>16</td>
<td>16 5 3 2 0</td>
<td>47</td>
<td>47 5 0</td>
</tr>
<tr>
<td>17</td>
<td>17 3 0</td>
<td>48</td>
<td>48 28 27 1 0</td>
</tr>
<tr>
<td>18</td>
<td>18 7 0</td>
<td>49</td>
<td>49 9 0</td>
</tr>
<tr>
<td>19</td>
<td>19 6 5 1 0</td>
<td>50</td>
<td>50 27 26 1 0</td>
</tr>
<tr>
<td>20</td>
<td>20 3 0</td>
<td>51</td>
<td>51 16 15 1 0</td>
</tr>
<tr>
<td>21</td>
<td>21 2 0</td>
<td>52</td>
<td>52 3 0</td>
</tr>
<tr>
<td>22</td>
<td>22 1 0</td>
<td>53</td>
<td>53 16 15 1 0</td>
</tr>
<tr>
<td>23</td>
<td>23 5 0</td>
<td>54</td>
<td>54 37 36 1 0</td>
</tr>
<tr>
<td>24</td>
<td>24 4 3 1 0</td>
<td>55</td>
<td>55 24 0</td>
</tr>
<tr>
<td>25</td>
<td>25 3 0</td>
<td>56</td>
<td>56 22 21 1 0</td>
</tr>
<tr>
<td>26</td>
<td>26 8 7 1 0</td>
<td>57</td>
<td>57 7 0</td>
</tr>
<tr>
<td>27</td>
<td>27 8 7 1 0</td>
<td>58</td>
<td>58 19 0</td>
</tr>
<tr>
<td>28</td>
<td>28 3 0</td>
<td>59</td>
<td>59 22 21 1 0</td>
</tr>
<tr>
<td>29</td>
<td>29 2 0</td>
<td>60</td>
<td>60 1 0</td>
</tr>
<tr>
<td>30</td>
<td>30 16 15 1 0</td>
<td>61</td>
<td>61 16 15 1 0</td>
</tr>
<tr>
<td>31</td>
<td>31 3 0</td>
<td>62</td>
<td>62 57 56 1 0</td>
</tr>
</tbody>
</table>


<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>63</td>
<td>1</td>
<td>0</td>
<td>82</td>
<td>82</td>
<td>38</td>
<td>35</td>
</tr>
<tr>
<td>64</td>
<td>64</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>83</td>
<td>83</td>
</tr>
<tr>
<td>65</td>
<td>65</td>
<td>18</td>
<td>0</td>
<td>84</td>
<td>84</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>66</td>
<td>66</td>
<td>10</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>85</td>
<td>85</td>
</tr>
<tr>
<td>67</td>
<td>67</td>
<td>10</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>86</td>
<td>86</td>
</tr>
<tr>
<td>68</td>
<td>68</td>
<td>9</td>
<td>0</td>
<td>87</td>
<td>87</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>69</td>
<td>69</td>
<td>29</td>
<td>27</td>
<td>2</td>
<td>0</td>
<td>88</td>
<td>88</td>
</tr>
<tr>
<td>70</td>
<td>16</td>
<td>15</td>
<td>1</td>
<td>0</td>
<td>89</td>
<td>89</td>
<td>38</td>
</tr>
<tr>
<td>71</td>
<td>71</td>
<td>6</td>
<td>0</td>
<td>90</td>
<td>90</td>
<td>19</td>
<td>18</td>
</tr>
<tr>
<td>72</td>
<td>72</td>
<td>53</td>
<td>47</td>
<td>6</td>
<td>0</td>
<td>91</td>
<td>91</td>
</tr>
<tr>
<td>73</td>
<td>73</td>
<td>25</td>
<td>0</td>
<td>92</td>
<td>92</td>
<td>13</td>
<td>12</td>
</tr>
<tr>
<td>74</td>
<td>74</td>
<td>16</td>
<td>15</td>
<td>1</td>
<td>0</td>
<td>93</td>
<td>93</td>
</tr>
<tr>
<td>75</td>
<td>75</td>
<td>11</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>94</td>
<td>94</td>
</tr>
<tr>
<td>76</td>
<td>76</td>
<td>36</td>
<td>35</td>
<td>1</td>
<td>0</td>
<td>95</td>
<td>95</td>
</tr>
<tr>
<td>77</td>
<td>77</td>
<td>31</td>
<td>30</td>
<td>1</td>
<td>0</td>
<td>96</td>
<td>96</td>
</tr>
<tr>
<td>78</td>
<td>20</td>
<td>19</td>
<td>1</td>
<td>0</td>
<td>97</td>
<td>97</td>
<td>6</td>
</tr>
<tr>
<td>79</td>
<td>79</td>
<td>9</td>
<td>0</td>
<td>98</td>
<td>98</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>80</td>
<td>80</td>
<td>38</td>
<td>37</td>
<td>1</td>
<td>0</td>
<td>99</td>
<td>99</td>
</tr>
<tr>
<td>81</td>
<td>81</td>
<td>4</td>
<td>0</td>
<td>100</td>
<td>100</td>
<td>37</td>
<td>0</td>
</tr>
</tbody>
</table>