Fault Simulation for Supply Current Testing of Bridging Faults in CMOS Circuits

by

Boey Yean Lim

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:

Scott F. Midkiff, Chairman

Joseph G. Tront

Dong S. Ha

September, 1989

Blacksburg, Virginia

Fault Simulation for Supply Current Testing of Bridging Faults in CMOS Circuits

by

Boey Yean Lim

Scott F. Midkiff, Chairman

Electrical Engineering

(ABSTRACT)

The objective of this research is to develop and implement a method for fault simulation that considers bridging faults in CMOS circuits that are tested using supply current monitoring. The discussion is restricted to single fault detection in CMOS combinational circuits. A CMOS circuit is represented by a two-level hierarchy. At the higher level, the circuit is partitioned into modules based on the circuit layout. Each module is represented at the lower level by a switch-level graph. This representation has the advantage of structural accuracy at the lower level and efficient logic propagation at the higher level. Based on a module's switch-level graph, an exhaustive list of bridging faults corresponding to certain physical defects can be derived. Fault collapsing techniques are used to optimize the exhaustive fault list. There are two major processes in this bridging fault simulation program, logic simulation and fault sensitization at switch level. The simulation program uses preprocessing and bit-wise parallelism to minimize computation time. At the end of fault simulation, a fault coverage and fault matrices suitable for test grading and fault diagnosis are produced for each test set.

This research also identifies types of CMOS modules and uses them to analyze test generation for bridging faults. The completeness and minimality of switch-level test sets are considered for general series-parallel (GSP) modules.

Finally, several single-module circuits are simulated using gate-level, switch-level and random test sets, and their effectiveness is compared.

Acknowledgements

I would like to thank my advisor, Dr. Scott Midkiff, for the guidance and advice given throughout the development of this thesis. He helped me keep going when it seemed that the end is beyond reach. I also thank my other committee members, Dr. Dong Ha and Dr. Joseph Tront, for their helpful suggestions.

In addition, I would like to recognize my Blacksburg friends who helped me generously, especially, , , , , , and . I would also like to thank my colleagues at work for their supports, especially , , and .

Special thanks also to , my best roommate, for making this experience all the more enjoyable. has been a special source of encouragement, and I am very thankful for his confidence and faithful friendship.

Finally, this thesis is dedicated to my parents. I am most grateful to their unconditional love and support, even at times when it is hard for them to comprehend.

Table of Contents

List of Illustrations

List of Tables

Chapter 1. Introduction

Complementary metal oxide semiconductor (CMOS) technology is widely used for large scale integration (LSI) and very large scale integration (VLSI) integrated circuits (ICs). Important characteristics such as low static power consumption, excellent noise immunity and high density have contributed to its success. Modern computer-aided design and test (CAD/T) tools have also been instrumental in the advance of CMOS technology. Increasing circuit complexity and the shrinking design and fabrication cycle lead to greater demands for quality and reliability controls for CMOS IC's [1,2].

Manufacturing imperfections can compromise an IC's functional performance by creating open or short faults. Some process instabilities can be easily detected and corrected, while others require sophisticated test systems [3,4]. Certain defects are more likely to occur than others and the push for higher chip density has led to an increase in the frequency of defects that result in bridging faults. Studies indicate that bridging faults are responsible for at least half of all failures [5,6]. An understanding of the failure mechanisms that cause bridging faults is therefore essential for developing effective computer-aided test programs.

A fault model, an abstraction of physical defects in the circuit, is used in generating test vectors, and also in grading the test sets by fault simulation. Physically realistic fault models for CMOS circuits must be closely related to the fabrication technology and to the circuit layout. Researchers

are examining switch-level fault models to replace the conventional gate-level line stuck-at fault model [9,10,11]. This new approach is superior in preserving certain transistor characteristics without paying the price of analog circuit computations.

In the past, fault simulators provided the fault coverage of a given test set by determining the ratio of detected faults to all specified faults. In an expanded role, a fault simulator may also [7]:

- identify undetected faults to improve a test set,
- create a fault dictionary useful for diagnostic purposes,
- analyze circuit testability,
- perform logic simulation,
- evaluate random vectors for testing, and
- compare different testing schemes.

Fault simulation, despite its importance to quality and reliability, is often bypassed due to long execution time, lack of understanding, and high cost [8]. The issue of speed is particularly critical for large circuits. Fast simulation can be approached through concurrent, deductive, and parallel algorithms [7].

The objective of this research is to develop and implement a method for fault simulation that considers supply current testing of bridging faults in CMOS combinational circuits and supports modeling accuracy and execution speed. Supply current testing is a technique that observes the leakage current in a CMOS circuit in order to detect short faults for which conventional techniques are ineffective. Supply current testing is discussed in Chapter 3. Using the simulation program, the effectiveness of gate-level, switch-level, and random pattern test sets for testing bridging faults is studied. The completeness and minimality of test sets for bridging faults in CMOS circuits are also discussed. This research is restricted to single fault detection in combinational CMOS circuits and only intra-module bridging faults described in Chapter 3 are considered.

A hierarchical circuit representation is used where the transistor layout of each CMOS module is described at the switch-level while the inputs and outputs of individual modules are connected globally at a higher level. This representation allows logic propagation at a high level for speed,

and fault sensitization at a low level for accuracy. The switch-level model is modified from Bryant's graph-based model [11]. Fault sensitization is determined from the logic simulation results of a good circuit; both fault injection and fault propagation are eliminated. Preprocessing is employed to increase the processing efficiency and to enable bit-wise parallelism in the simulation program. Circuit partitioning in terms of modules and their processing order are specified by the user. The preprocessing step also includes the extraction of essential characteristics from the switch-level graph of each module type.

For the experiments, a PODEM-based test generation program, Mahjong [12], is used to supply gate-level test vectors for line stuck-at faults. At the switch-level, a minimal complete leakage test set (MCLTS) based on a graph-theoretic algorithm by Malaiya and Su [31] is derived when applicable. Chapter 4 of this thesis presents another graph-based switch-level test set and also derives a lower limit for the size of test sets that have 100 percent fault coverage. The set so derived is called the Minimal test set for a module when it has 100 percent fault coverage; otherwise, it is called the Near-Minimal test set when additional test vectors are needed for it to achieve 100 percent coverage. Random test patterns are also included in the experiments. Fault coverages and size of the different test sets are compared.

Following this introduction, Chapter 2 presents a survey of important circuit models using different levels of representation. Special attention is given to switch-level models. Chapter 3 looks at failure mechanisms for CMOS technology, the supply current testing technique, and the types of bridging faults addressed by this thesis. Bridging faults, as considered in this research, are defined. With the circuit model and testing technique specified, Chapter 4 presents the algorithm for bridging fault simulation. It also includes analysis of test generation for certain classes of CMOS circuits. Algorithm implementation is then discussed in Chapter 5. Chapter 6 describes the fault simulation experiments and results. Finally, Chapter 7 summarizes these results and suggests directions for future research.

Chapter 2. Representation of CMOS Circuits

The purpose of this chapter is to survey circuit models, which are represented at four levels of abstraction, and to highlight some of the important characteristics of CMOS circuits relevant to this research. The merits and drawbacks of each category are mentioned and in particular, Bryant's switch-level graph is discussed.

2.1 Levels of Abstraction

The objective of circuit modeling for computer-aided test programs is to describe a circuit in terms of simple primitives. Four levels of abstraction are widely used in circuit models: functional or behavioral, gate, switch, and circuit. These four abstraction levels are ranked in accordance to their departure from the physical structure of the circuit. It should be noted that two or more levels of abstraction may be used in a single process, and mixed-level simulators have been developed [13,14].

2.1.1 Functional or Behavioral-Level Representation

The highest level of abstraction is the functional or behavioral level. Some differences do exist between functional and behavioral models, but for the purposes of this discussion they are effectively the same. A representation at this level gives a general behavioral description of a circuit in terms of functional blocks such as counters, shifters, decoders, multiplexers and even complete chips [15]. The output behavior of each functional block is based on the inputs and internal state. The internal organization, however, is not specified. Complex designs can be represented by a few blocks and fast evaluation can be employed for analysis and simulation. This high speed simulation is ideal for early design and testability verification. A circuit at this level is often described using a hardware description language (HDL) [15,16].

A behavioral fault model represents a failure in a VLSI design as a deliberate perturbation injected into the fault-free behavioral description. The injected fault may be an incorrect variable value, incorrect behavioral or functional description, or a combination of these [15,17]. Such models may be derived from actual observed failures in complex VLSI designs [18]. Circuit technology is not considered to any significant extent. Moreover, internal fault modes and detailed timing cannot be represented.

Since the internal structure is not known, some likely faults may be overlooked while unlikely faults may be injected. Therefore, the completeness and effectiveness of the fault list, and thus the test vectors, cannot be assured. Moreover, finding the cause of a known fault behavior, i.e., fault diagnosis, can be difficult since different defects can cause the same fault behavior. For CMOS fault simulation, the main drawback of functional and behavioral models is their lack of accuracy and their poor support of fault diagnosis.

2.1.2 Gate-Level Representation

Gate-level modeling is a well-understood concept originally developed for TTL technology and is widely used in many existing fault simulators [19]. The principle primitives are boolean logic gates that operate on binary signals. The set of primitives consists of functions such as AND, NAND, OR, NOR, NOT, and XOR. A circuit is modeled as a connected set of primitive gates. Faults are modeled by assuming that the connecting lines between logic gates are stuck at logic 1 or logic 0. Three well-known gate-level algorithms for test generation are the D-algorithm [20,21], PODEM [21], and FAN [21].

Again, because gate-level representations are independent of the transistor topology and the technology used, many CMOS failure modes cannot be represented, while irrelevant faults may be introduced [3,5]. Consider the function, $f = \{ (l_1 + l_2) \cdot l_3 + l_4 \cdot l_6 + l_5 \}'$, whose gate-level abstraction and transistor layout are depicted in Figure 1 and Figure 2, respectively. Gate-level stuck-at faults are difficult to relate to physical defects in the circuit layout. For example, there is no direct indication as to what kind of physical defects constitute stuck-at faults at lines 7, 8, and 9 in Figure 1, or if they correspond to any physical defects at all.

On the other hand, a physical defect such as a break in the diffusion layer or a missing transistor channel between point 1 and point 2 in Figure 2 may cause the output to have an indeterminate value depending on the severity of the defect and the test vector applied. A break in the conducting material between point 2 and point 3 may also result in an indeterminate output logic value. Another example is that an extra conducting layer bridging point 4 and point 5 could cause an indeterminate output logic value or create a conducting path between V_{dd} and V_{ss}. These defects have no line stuck-at equivalents at the gate-level.

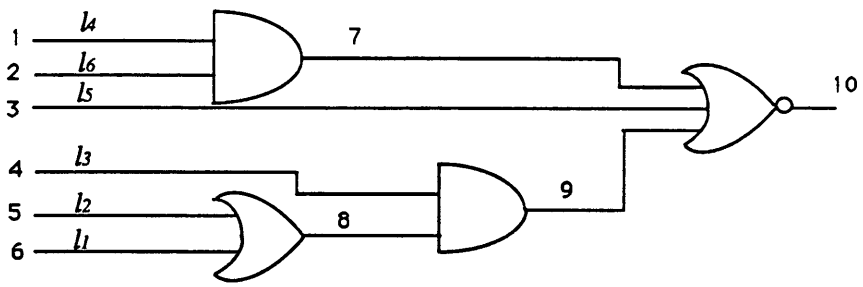

Figure 1. A gate-level representation $f = \{ (l_1 + l_2) \cdot l_3 + l_4 \cdot l_6 + l_5 \}'$.

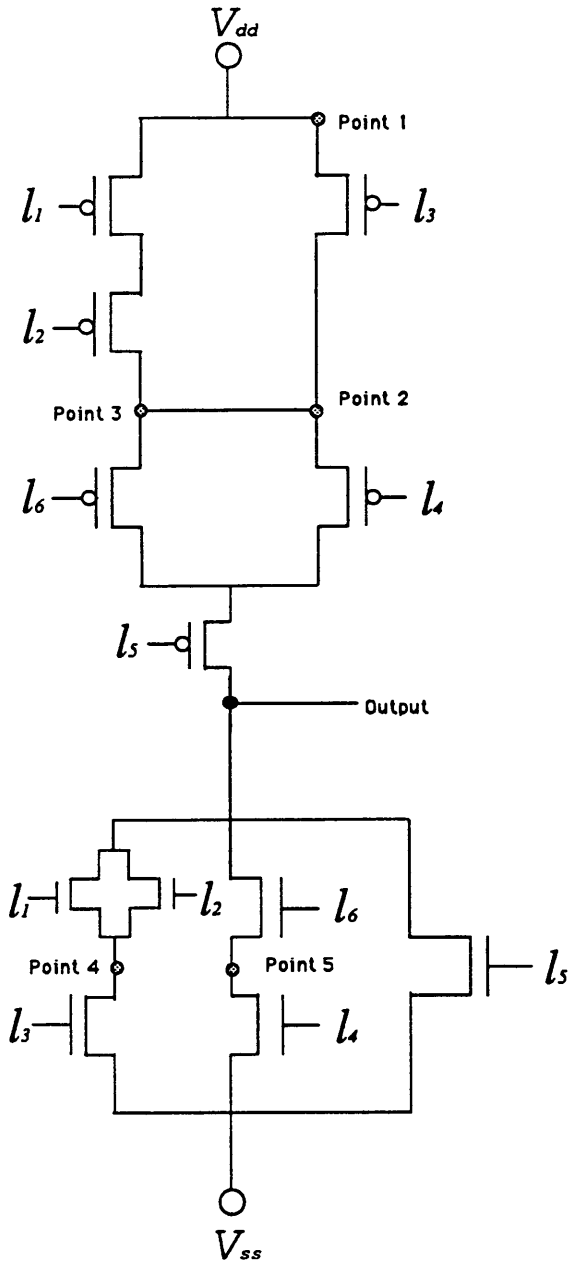

Figure 2. A transistor layout $f = \{ (l_1 + l_2) \cdot l_3 + l_4 \cdot l_6 + l_5 \}'$.

2.1.3 Switch-Level Representation

Switch-level models have been used to represent CMOS circuits more accurately than gate-level models [22,23,24]. These models provide a direct representation of the transistor structure, but at the same time greatly simplify the electrical behavior by treating each transistor as a simple switch, controlled by the voltage at its gate terminal. The source and drain terminals of a switch are called *nodes*; a CMOS circuit is then modeled as a set of nodes connected by switches. Important phenomena such as bidirectionality of signal flow, dynamic charge storage and sharing, and resistance ratios can be modelled at this level. Most switch-level models represent node voltages and capacitances by discrete logic levels, and different timing units may be used [22]. This close correspondence to the physical layout and technology permits a more realistic representation of CMOS circuits and defects. Switch-level fault modeling is discussed in detail in Chapter 4.

2.1.4 Circuit-Level Representation

The circuit level is the lowest level of representation. A set of connected elements such as transistors, resistors, and capacitors forms a circuit. Detailed electrical analysis can be performed. The ability to do analog analysis of current, voltage, and timing makes it the most accurate of all representations. A circuit simulator, such as SPICE [13,25], is commonly used for precise evaluation of circuits that are relatively small. However, the trade-off for this accuracy is slow speed. Therefore, a circuit-level model is not suitable for efficient fault simulation.

2.2 *Bryant's Switch-Level Model*

Switch-level models correspond more closely to CMOS circuit structures than gate-level models and can be processed with better speed than circuit-level models. Several switch-level models have been proposed for describing and simulating MOS circuits [3,10,26,27]. The pseudo-boolean representation in Hayes's Connector-Switch-Attenuator (CSA) model allows very accurate low-level circuit simulation [26]. Its computation complexity is a major drawback. The Node-Switch-Wire (NSW) model, developed by El-Ziq [3], is a highly simplified switch-level model where a transistor or a wire connects a pair of nodes. Every node is assumed to have a logical neighborhood of four other nodes: northern, eastern, western, and southern (NEWS). A NEWS tracing technique is developed to propagate sensitized faults within a CMOS module. In general, the graph-based switch-level model developed by Bryant [27] has wide application because of its accuracy and simplicity.

In Bryant's switch-level model, a circuit is represented by a switch-level graph, G = { T,N }, where T is a set of transistors and N is a set of nodes. A transistor has three terminals named gate, source, and drain. The source and drain terminals are conditionally connected by the state of the gate terminal. Every transistor is a bidirectional device and is given a *strength* that indicates its relative conductance. Two types of transistors are needed in a CMOS logic gate. An n-channel transistor is modeled by a positive switch, one that turns on when its gate terminal has a logic value of 1, and a p-channel transistor is modeled by a negative switch, one that conducts when the gate terminal is 0. A node is either an input node or a storage node. Examples of input nodes include the two supply sources, V_{dd} and V_{ss}, as well as any clock or external data inputs; these are strong signals unaffected by changes occurring within the circuit. The storage nodes behave like capacitors by retaining their states as a result of circuit operation. Each storage node has a state in the ternary set {0, 1, X}, where 1 and 0 are the usual high and low voltage levels, and X is an indeterminate voltage. A storage node is also assigned a *size* that indicates its capacitance relative to other charge-sharing nodes. For most CMOS circuits, neither the relative transistor strengths nor the

relative storage node sizes are significant, so one strength and one size can be used. Detailed circuit timing can also be avoided to improve simulation efficiency. The switch-level model used in this research is simplified from the model just described and is discussed in Chapter 4.

2.3 CMOS Characteristics

Some characteristics of CMOS circuits are introduced in this section as background for later discussion. In this research, a CMOS module is a block of transistors that implement a boolean logic function and is connected to the power supply, V_{dd}, and ground, V_{ss}. There are two disjoint networks called the *P-network* and the *N-network* for each CMOS module. The N-network is a pull-down network of n-channel transistors connecting the output to ground, V_{ss}. The P-network is a pull-up network of p-channel transistors connecting the output to the power supply, V_{dd}. A simple CMOS module belongs to the set { NAND, NOT, NOR }, and a complex CMOS module is one that has no primitive boolean logic gate equivalent.

For a fully complementary CMOS gate, each p-channel transistor in the P-network and a dual n-channel transistor in the N-network are always in complementary modes, i.e., if one is turned on, the other is off. There is also a pair of dual transmission functions, $\hat{f}$ for the N-network, and f for the P-network. For a fault-free CMOS module, its output is always connected to either V_{dd} via the P-network, or V_{ss} via the N-network, but not both, at all times except during switching. The output may be temporarily linked to both V_{dd} and V_{ss} during switching. Since a non-conducting transistor has an extremely high resistance between the source and drain, a conduction path between the power supply and ground is prevented. Except during switching, the steady-state leakage current, I_{dd}, is extremely small. A ternary logic value is necessary for CMOS circuit models since shorts, transient or permanent, may lead to an uncertain output state that is neither high nor low.

2.4 Summary

Four important circuit representations were briefly reviewed for their performance in CMOS circuit representation and simulation. Switch-level models provide logical abstractions that are in close correspondence with the physical structure of CMOS circuits. The switch-level model developed by Bryant offers accuracy, simplicity, and generality. Finally, important CMOS circuit characteristics that have bearing on this research were pointed out.

Chapter 3. Failure Mechanisms and Testing of CMOS Circuits

Fault modes are usually a function of technology and circuit design. Recent studies of CMOS physical defects and their impact on circuit behavior have revealed new non-classical switch-level faults [5,28,29]. Especially prevalent are physical defects that cause unwanted conductance in a circuit. In this research, these are collectively called *bridging* faults. Like conventional gate-level fault simulators, existing switch-level fault simulators support testing based on logic observation. However, such logic-domain testing may be unsatisfactory for detecting bridging faults in CMOS circuits [10,25,30,31]. As an alternative testing approach, quiescent power supply current or I_{dd} current measurement can be used to effectively detect bridging faults [30,32,33].

3.1 Physical Defects

ICs are very susceptible to process instabilities and contamination during fabrication. These process errors may lead to malfunctions at the circuit level. Physical defects are broadly classified

as global or localized. Extensive tears in materials, mask misalignments, overlapped or discontinuous materials, and severe control errors are possible causes for global defects. Global defects are spread over multiple ICs, usually making them easy to detect. Localized defects are often caused by extra or missing material in a relatively small region within a single IC. Unlike global defects, the effects of local defects can be more subtle. Hence, the focus of fault detection is on these localized defects.

A fault is a deviation from a specified circuit behavior as a result of changes due to defects in the circuit. Several defects may cause the same effect, while several different effects may share a common cause. Without a clear understanding of defects at the low level, failure mechanisms are difficult to identify. Fortunately, not all localized defects are critical at the logic level. For instance, extra metal deposited in a neighborhood of non-conducting material does not cause a fault. Researchers have classified important localized defects by their characteristics [3,4,10]. Ferguson and Shen combine statistical data from fabrication lines with IC layout extraction in a systematic technique called Inductive Fault Analysis (IFA) [5,6]. The effects of gate oxide shorts on CMOS circuit reliability are reported in the work of Charles and Jerry [29]. Other investigations of non-classical faults in CMOS ICs are presented in [23,28,31].

The most significant failure modes are characterized as either opens or bridges in the circuit. A bridge results in unwanted conductance, whereas the absence of needed conductance marks the occurrence of an open fault. The transistor stuck-off fault prevents transistor conductance, while the transistor stuck-on fault causes the transistor to conduct regardless of the input at the gate terminal. In particular, missing polysilicon can cause a transistor to be stuck-on, and extra deposits of conducting material or inadequate deposit of insulating material at critical regions can cause other types of bridging faults. Gate-oxide contamination, pin holes, and p-well errors can also contribute to undesired bridging faults. Breaks due to extra insulating material or missing conducting material may cause unwanted open faults. Moreover, new transistors may be created as a result of extra polysilicon crossing the diffusion layer.

3.2 *Bridging Faults*

Much of the attention on CMOS fault analysis has focused on transistor stuck-open faults [8,28]. However, the results of the IFA studies of commercial CMOS circuits indicate that transistor stuck-off faults constitute a relatively small percentage of occurring faults [5,6]. The study further suggests that half of the occurring faults can be attributed to short faults in general [5]. Clearly, bridging faults should have a substantial impact on CMOS circuit test and are discussed below in more detail.

Since CMOS bridging faults are the primary focus for this research, all faults in the rest of the discussion refer to bridging faults which are classified into three categories. At the transistor level, any short between two distinct circuit nodes is called a *general short*. A special case of this general short is called a *transistor stuck-on* fault where the two nodes are the source and drain terminals of a transistor. An *input short* occurs when a transistor's gate terminal is shorted with any one of the circuit nodes. A special case for the input short is when a gate terminal is shorted to a power supply node, V_{dd} or V_{ss}, with the same effect as the classical line stuck-at faults. Faults are also classified according to their location. An *inter-module* fault is a bridging fault involving two different modules of a circuit, while an *intra-module* fault is limited to input gate terminals and circuit nodes within a single module. Layout proximity has a profound influence on the likelihood of fault occurrence. Therefore, a weighted fault list based on the likelihood of fault occurrence is realistic and advantageous for fault simulation. Figure 3 illustrates the three types of bridging faults considered in this work.

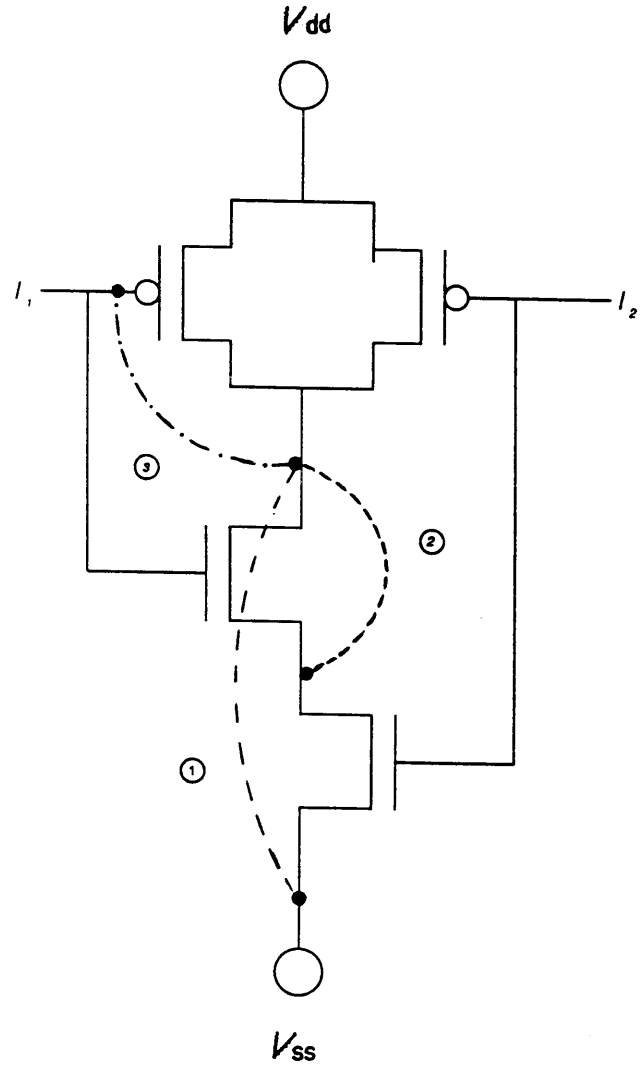

Figure 3. Types of bridging faults: (1) general short, (2) transistor stuck-on, (3) input short.

3.3 *Supply Current Testing*

Detailed examinations of bridging faults and their behavior modes have been conducted [33,34]. When two circuit nodes which otherwise should have opposite logic values are shorted, an ohmic path connecting V_{dd} and V_{ss} is created, causing a large I_{dd} current to be drawn. The faulty circuit can also suffer from increased propagation delay as a result of the voltage divider caused by the short [25]. Depending on the resistance value of the short, the logical output can become unpredictable. A bridge with a very high resistance will have little influence over the normal logic output. If the resistance is very low, it may give rise to a classical stuck-at fault. For intermediate resistances, the logic output may be at a level that is neither a logic 1 nor a logic 0, and the interpretation of the signal by subsequent devices is unpredictable. This unpredictability renders conventional logic-domain testing ineffective for some bridging faults [25,30,33]. Therefore, I_{dd} current testing may be better suited for bridging fault detection. Some of the merits of current mode testing are listed below.

- A wider range of short resistances can be detected with current testing compared to the two-cycle propagation delay measurement [25].
- Current testing allows the early detection of non-logical faults that may lead to future failures [29,33].
- Excessive current is easily distinguished from normal leakage current, therefore providing high resolution [35].
- Existing parametric testing techniques can be adjusted to support current testing [30].
- Internal transistors are accessible without requiring that extra pins be added for testing.

The major drawback to this approach is the relatively long testing time and the need for a decision procedure for fault identification. A set-theoretic approach, suggested by Malaiya [31], can be employed to determine the parametric condition of a circuit under test. The testing time can

be reduced by limiting the size of test sets and by using a shorter fault list. Test grading through fault simulation and fault collapsing techniques are therefore essential to minimize I_{dd} test time.

3.4 Switch-Level Fault Simulators

Most physical failures in CMOS IC's are electrical shorts and opens that cannot be simulated by traditional fault simulators that use a gate-level circuit model and a line stuck-at fault model. The characteristics of MOS technology have led to new switch-level fault simulators with improved simulation accuracy. Most of the switch-level fault simulators support logic-domain test techniques and some are also able to support timing simulation.

Among the pioneers is FMOSSIM [36], a switch-level fault simulator for general MOS circuits. Although FMOSSIM is faster than circuit simulators by as much as two orders of magnitude, its complete switch-level representation causes it to be significantly slower than gate-level simulators. Other disadvantages include the inherent limitations of logic-domain test techniques for bridging faults and cumbersome fault injection.

CSASIM [37] is another switch-level simulator for general CMOS circuits based on the CSA model. Multiple logic values are used. The use of primitive elements such as resistors and capacitors makes it very accurate and capable of detecting delay faults. On the other hand, the detailed simulation is very time consuming. Like FMOSSIM, fault injection is required.

Both FMOSSIM and CSASIM lack simulation efficiency. To improve speed, concurrent hierarchical fault simulators have been designed. MOTIS [38] is a fast fault simulator. It supports multi-level circuit representations that range from switch-level to functional-level. A CMOS circuit is partitioned into several modules each of which is represented using a low level representation such as a gate or switch-level model. Both generic inter-module and intra-module faults can be modeled. High speed is achieved through fast logic and fault propagation at a higher level of abstraction. The main drawback of this simulator is also its limitation to logic-domain test techniques.

CHIEFS [14] is another concurrent hierarchical fault simulator. It supports direct simulation from the hierarchical description without having to convert to the level of primitives. The user has to create an extensive fault library for each circuit to be simulated. A separate simulator with timing capability or a circuit-level simulator such as SPICE is needed to provide accurate fault descriptions. This can be cumbersome and the completeness of a fault library is questionable, especially for large circuits. This simulator is also restricted to the logic-domain testing.

Special-purpose hardware has also been developed to improve both the accuracy and speed of fault simulation. The Yorktown Simulation Engine switch-level simulator, running on the parallel Yorktown Simulation Engine (YSE) [39], has impressive speed. However, it also depends on logic-domain test techniques.

The objective of this research is to develop an efficient fault simulation program that has the following characteristics:

- switch-level circuit representation,
- processing of realistic bridging faults,
- simple fault specification,
- supply current testing, and
- fast execution.

It should be noted that the fault simulator developed in this research has its own limitations. It considers only current mode test, bridging faults, and combinational circuits. Also, pass transistors are not considered.

3.5 Summary

Failure mechanisms for ICs are technology and layout dependent. Therefore, realistic fault models for CMOS combinational circuits should consider these two factors. The IFA study has indicated that the percentage of transistor stuck-off faults is relatively small compared to bridging faults. Conventional logic-domain testing may be inadequate for detecting bridging faults. I_{dd} current monitoring, however, is well suited for detecting bridging faults in most CMOS circuits and is a promising test technique. Existing switch-level simulators are either very precise and time consuming or cannot handle bridging faults appropriately. In the next chapter, a bridging fault simulation program is proposed.

Chapter 4. Analysis and Simulation of Bridging Faults

In this chapter, a two-level hierarchical model for CMOS circuits is presented. A circuit is partitioned into modules at the higher level, and individual modules are described by switch-level graphs at the lower level. This chapter focuses on the methods used for fault-free logic simulation, exhaustive fault list generation, and fault sensitization in a module at switch level. The last section of this chapter suggests a graphical approach to test generation for bridging faults and discusses minimum test sets for certain types of CMOS modules.

4.1 Circuit Model

CMOS combinational circuits conveniently conform to a two-level circuit representation. At the higher level, the circuit is partitioned into a connected network of gates or modules, each of which is the smallest set of transistors realizing a boolean function. Since the circuits are combinational, there is no feedback in the network. Logic values are directly assigned to all inputs of a

module. At the lower level, switch-level graphs accurately represent the transistor structure of individual modules.

4.1.1 Circuit Partitioning

Every module is a single-output, multiple-input logic block with a power supply, V_{dd}, and a ground, V_{ss}. A module's output may serve as an input to other modules; Module A is said to drive Module B if the output of the former is fed into the latter as an input. Obviously, the driver module has priority over the driven module during logic simulation. Both simple and complex CMOS logic blocks are treated as modules. Transmission gates which have neither the power supply nor the ground are not included in the discussion, but are important for future consideration. Figure 4 illustrates the partitioning of an 18-input CMOS circuit into three modules. Notice that Module 1 drives Module 2, and Module 2 drives Module 3. All primary inputs are indicated by positive numbers and all module outputs have negative numbers whose absolute value is the module number.

4.1.2 Switch-Level Representation

The switch-level model used to represent a module is a simplified version of Bryant's model [11,22]. Assuming combinational circuits and current monitoring as the detection technique for CMOS bridging faults, the following changes are made to simplify the model without considerable sacrifice to accuracy.

- Binary values { 0,1 } explicitly represent node voltages.
- Nodes have uniform capacitance or size.

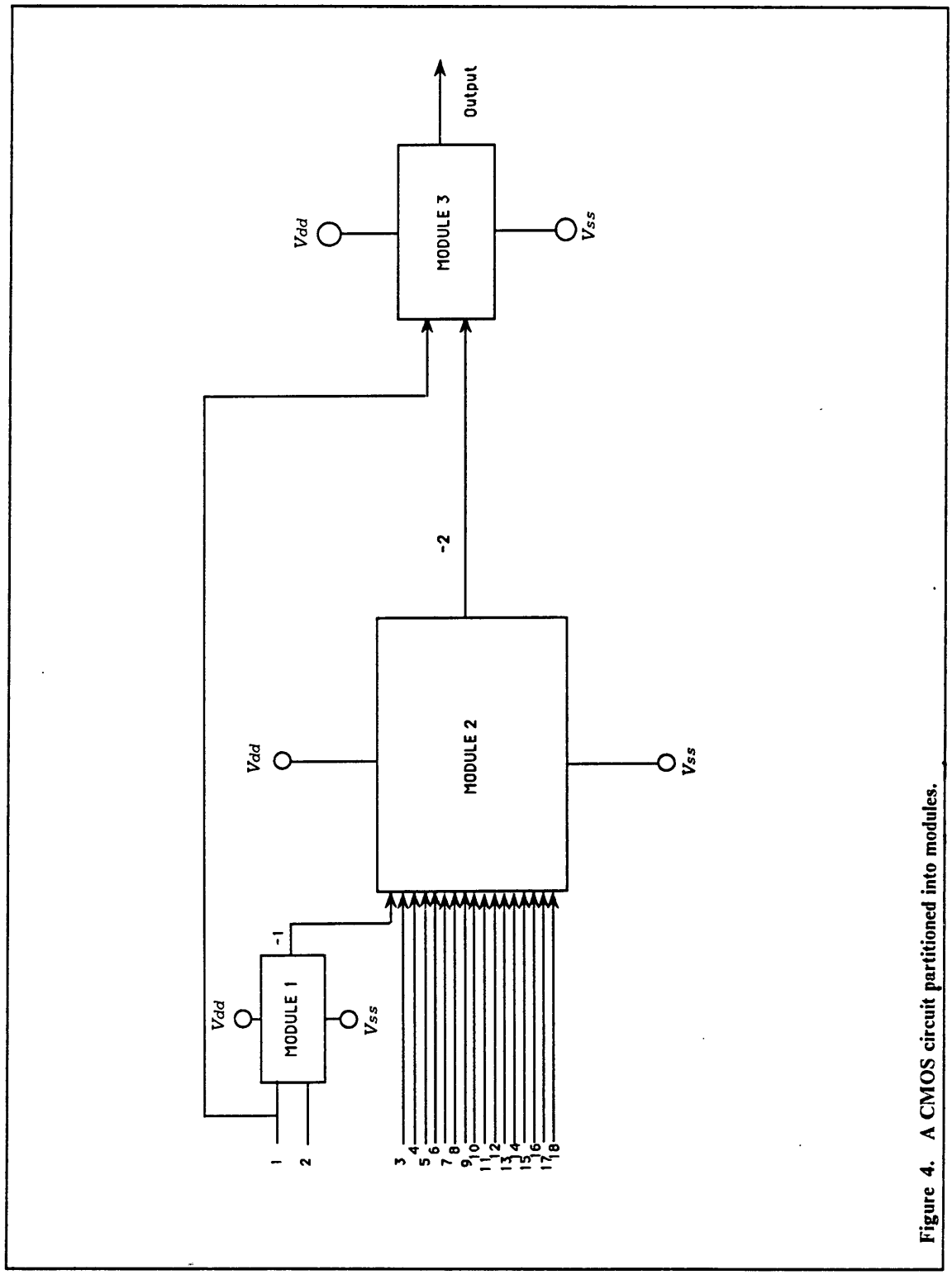

Figure 4. A CMOS circuit partitioned into modules.

- Transistors have uniform conductance or strength.

A labeled undirected graph G, similar to Bryant's channel graph, is used for describing an arbitrary switch-level module. It consists of three distinct sets, $G = \{ N, E, L \}$, which are defined as follows.

N A set of internal storage nodes that represents the source and the drain terminals of transistors in the module. Except for a group of parallel transistors, each series transistor connects a unique pair of nodes in N. For t transistors in parallel, only one pair of nodes, the common pair, is connected instead of t unique pairs. The drain node of a common pair represents the drain terminals of all the transistors in the parallel group. Likewise, the source node of the common pair represents the source terminals of the parallel transistors.

E A set of edges, where each edge represents a transistor or a group of parallel transistors whose drain and source nodes are in N. A common pair of nodes in N defines a common edge in E. That is, one common edge represents the t parallel transistors.

L A set of control labels, where each control label corresponds to an input of the module. Every edge in E has at least one control label. A common edge that represents multiple transistors in parallel will have multiple labels. In a switch-level graph, a control label with a negative sign means the inverse of the corresponding input is used.

Switch-level implementations of two of the modules in Figure 4 are illustrated in Figure 5 and Figure 6. In Figure 5, two parallel p-channel transistors are shown having a pair of common nodes (1,2), a common edge (1-2) and two control labels { 1,2 }. Notice that in the switch-level graph, an edge that represents a p-channel transistor or a group of parallel p-channel transistors is marked by a black filled-in circle, while an edge for an n-channel transistor or transistors in parallel has a plain circle.

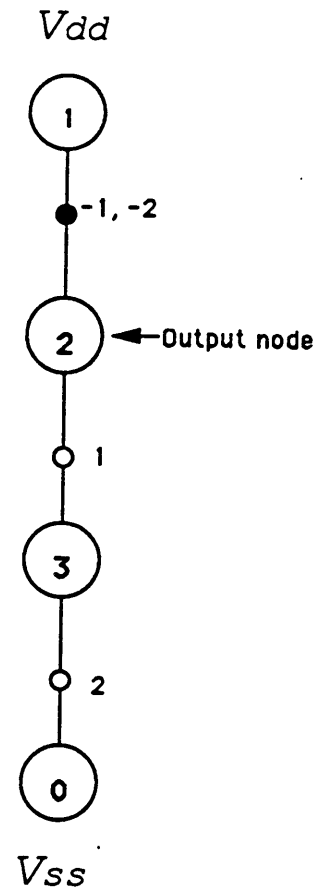

Figure 5. Switch-level graph of Module 1: $f = (l_1 \bullet l_2)'$.

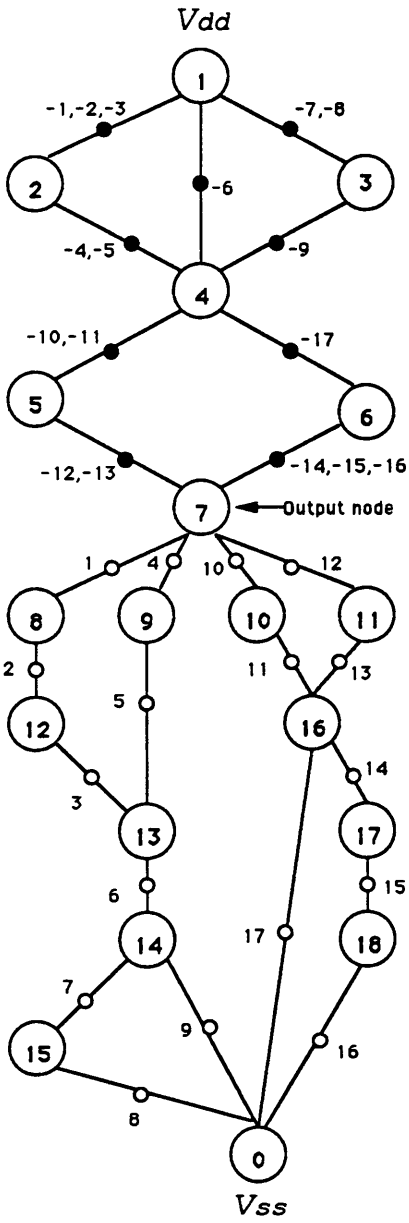

Figure 6. Switch-level graph of Module 2: $f = \{ [(l_1 \bullet l_2 \bullet l_3 + l_4 \bullet l_5) \bullet l_6 \bullet (l_7 \bullet l_8 + l_9)] + [(l_{10} \bullet l_{11} + l_{12} \bullet l_{13}) \bullet (l_{17} + l_{14} \bullet l_{15} \bullet l_{16})] \}'$.

In general, every series transistor in the layout structure has a unique edge with only one control label and a unique pair of nodes in the equivalent switch-level graph. There is a one-to-one correspondence between the physical layout and the switch-level graph for series transistors. However, a group of parallel transistors is represented by a single edge with multiple control labels and the effective input is the logical OR of all inputs assigned for that edge. This switch-level abstraction of parallel transistors is adopted to reduce the size of the fault list and to cut down on simulation iterations.

Circuit partitioning defines set L for every module at the higher level, but sets N and E are defined only at the switch level. By referring to the switch-level graph and the boolean function of the individual module, the sizes of N and E can be determined and are found to be linearly proportional to the size of set L. The size of N for a simple complementary CMOS module with a simple function, such as INVERTER, NAND, and NOR gates, is $\ell + 2$ where ℓ is the size of L. A simple module that has t parallel transistors in its N-network must have t series transistors in its P-network and vice versa. Unlike the parallel transistors that are treated as one common transistor, each series transistor can be uniquely associated with a node in N and a label in L. Therefore, the total number of nodes in N is $\ell + 2$, where the second term accounts for the two nodes, V_{dd} and V_{ss}. The 2-input NAND gate, $f = \{l_1 \cdot l_2\}'$, depicted in Figure 5 has two series transistors in the N-network and hence the size of N is $2+2 = 4$.

The same principle can be applied to complex modules, since their boolean functions are composed of several simple functions. The size of N equals the total of the node counts from all the simple functions with the power supply and ground nodes counted once. For a fan-out free complementary CMOS module, simple or complex, the number of nodes in N is:

$$n = \ell + 2$$

For example, the boolean function of Module 2 in Figure 6 can be broken down into nine distinctive terms, each of which is a NAND, NOR or INVERTER gate:

$$f = \{ [(l_1 \cdot l_2 \cdot l_3)' \cdot (l_4 \cdot l_5)' + (l_6)' + (l_7 \cdot l_8)' \cdot (l_9)'] \cdot [(l_{10} \cdot l_{11})' \cdot (l_{12} \cdot l_{13})' + (l_{17})' \cdot (l_{14} \cdot l_{15} \cdot l_{16})'] \}$$

By adding up the node count from each term, $N = (3 + 2 + 1 + 2 + 1 + 2 + 2 + 1 + 3) + 2 = 17 + 2 = 19$. For a fan-out complex module, the size of N is:

$$n = 2 + \sum_{i=1}^{\ell} m_i$$

In the above equation, m_i is the number of fan-outs for the *ith* control label.

Since every label for a fan-out free complementary CMOS module controls two dual edges in E, without the simplification of parallel transistors at the switch-level, a module with ℓ labels will have 2ℓ distinct edges. With parallel transistors represented by a single edge, the size of E for a simple module is reduced to $\ell + 1$, one edge for each series transistor and one for all the parallel transistors. For example, the 2-input NAND gate in Figure 5 has three distinct edges, two for the series transistors and one for the parallel transistors. Again, by breaking a complex function into a collection of simple functions and summing their contributions, the size of E can be determined. For a fan-out free fully complementary module with ℓ_i inputs for its *ith* simple function, where $\sum_i \ell_i = \ell$, there are $(\ell_i + 1)$ edges corresponding to that simple function. Therefore, the size of E is $e = \sum_i^s (\ell_i + 1) = \ell + s$, where s is the number of simple functions contained in the complex function. For ℓ inputs, the upper limit for e is dependent on the maximum number of simple functions that can be found in a complex function, which is the maximum number of 2-input simple functions. Therefore, the size of E for a fan-out free fully complementary function is:

$$e = \ell + s \leq \ell + \left[\frac{\ell}{2} \right]$$

where $[i]$ is the smallest integer greater than or equal to the number i. The complex switch-level graph in Figure 6, for example, has $e = [(3+1) + (2+1) + (1+1) + (2+1) + (1+1)] + [(2+1) + (2+1) + (1+1) + (3+1)] = 17 + 9 = 26$. For fan-out complementary CMOS modules, the number of distinct edges in E is counted in the same manner as in the case of fan-out free modules and every control label in a simple function is counted whether or not it also appears in another simple function, i.e.,

$$e = \sum_{i=1}^{\ell}(m_i) + s < 2\sum_{i=1}^{\ell} m_i$$

It can be concluded that for a fan-out free fully complementary module, the size of E and the size of N are $O(\ell)$. In the case of a fan-out fully complementary module, N and E are $O(\ell_m)$ where ℓ_m is the sum of all inputs including their fan-out counts. In general, the size of N and E are linearly proportional to the size of L for a fully complementary module.

4.2 *Logic Simulation*

In the following discussion of fault-free logic simulation of a CMOS module, the switch-level graph is labeled such that every node in N and every label in L are identified by numbers using the numbering system discussed in Chapter 5. The response of a module to an input vector during logic implication is captured in a *response* graph. Every unique input vector defines a unique response graph, while the switch-level graph remains unchanged.

In a response graph, all nodes in N of the switch-level graph are present but only *active edges* are defined. An edge is active when the corresponding transistor or at least one of the corresponding parallel transistors is turned on. A label with a logic value "1" will turn on an n-channel transistor and turn off a p-channel transistor that it controls, while a logic value "0" will have the opposite effect. Figure 7 and Figure 8 show response graphs of Module 1 and Module 2, whose switch-level graphs are shown in Figure 5 and Figure 6, respectively.

An active node is one that is linked to one of the supply nodes, V_{dd} and V_{ss}, through a low resistance path. Both the V_{dd} and V_{ss} nodes are active nodes by default. An active edge effectively causes its two end nodes to share a common state; the active node will force its state to the other node. Through this kind of mutual node interaction across active edges, paths are created in the

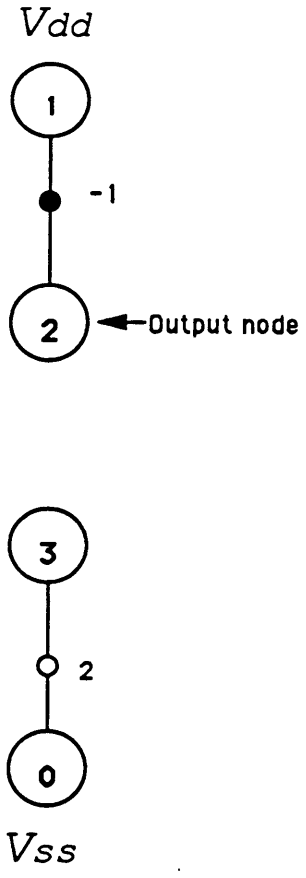

Figure 7. A response graph of Module 1: input vector = (0 1).

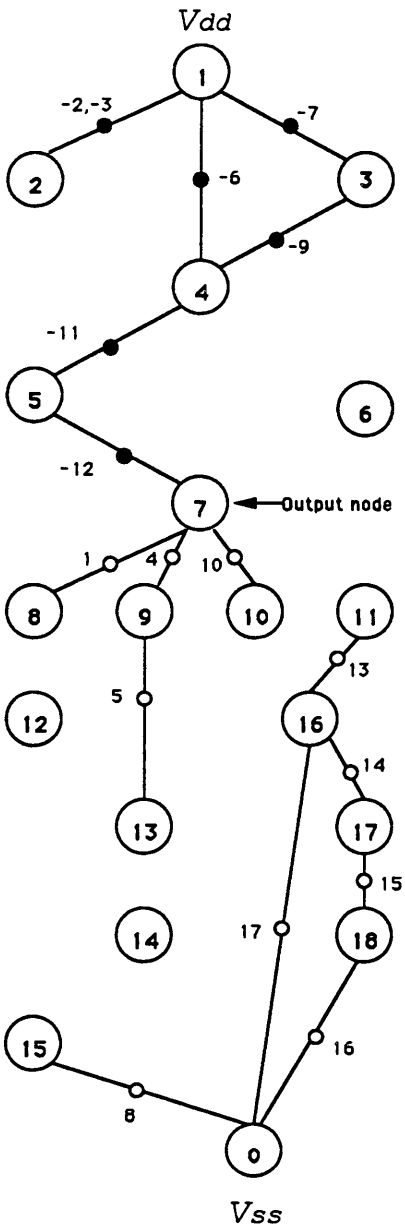

Figure 8. A response graph of Module 2: input vector = (1 0 0 1 1 0 0 1 0 1 0 0 1 1 1 1 1).

response graph. The effective state of a node is influenced by other nodes that are linked to it through established paths. To simplify the discussion, all paths with a common end node are treated as an entity called a *tree*, which consists of one or more nodes. The V_{dd} tree, therefore, includes all nodes connected to V_{dd} including V_{dd} itself. Likewise, the V_{ss} tree includes nodes of all paths terminating at V_{ss} in addition to V_{ss}. Since this is a fault-free CMOS module, there is always one low resistance path connecting the output node to either the V_{dd} or the V_{ss} node and therefore, the output node always belongs to one of the two trees. From Figure 8, the V_{dd} tree of Module 2 contains nodes { 1,2,3,4,5,7,8,9,10,13 }, and the nodes of the V_{ss} tree are { 0,11,15,16,17,18 }. There may also be paths that are not included in either the V_{dd} tree or the V_{ss} tree. Their associated nodes are called *floating* nodes. In the example of Figure 8, { 6,12,14 } are floating nodes. Every node in the V_{dd} tree has a logic value of "1"; every node in the V_{ss} tree has a logic value "0" and a floating node has an indeterminate logic value. The two trees, V_{dd} and V_{ss}, are mutually exclusive for a fault-free module. For a faulty module, however, a bridging fault may cause the two trees to merge.

4.3 Fault Sensitization

Two important processes in most fault simulation algorithms are logic implication and fault sensitization. Logic implication assigns input values to every module. A fault is sensitized by a test vector when its effect is detectable at an observable output. Fault sensitization identifies faults in a module that are sensitized by a particular set of inputs. Fault injection, the process of adding predefined faults into a circuit model, is widely used in fault simulation. The results of both good and faulty circuits are compared to determine if injected faults are detected by a particular input vector [20]. This section describes a different approach to fault sensitization for the bridging fault simulation algorithm developed in this research.

The fault simulation algorithm assumes that supply current testing is used to detect faults. An exhaustive list of bridging faults, as defined in Chapter 3, can be generated by the simulation pro-

gram. This automation increases the fault list accuracy and allows larger and more irregular circuits to be analyzed. For each test vector, only one logic simulation of the fault-free module is needed to determine all of the bridging faults that are sensitized by the input vector. Fault injection and logic simulation of faulty modules are therefore eliminated. This results in a considerable saving in computation time, especially for large circuits.

4.3.1 Sensitization

For supply current testing, a bridging fault is considered to be sensitized whenever its two end nodes have opposite logic values, i.e., when one of the nodes belongs to the V_{dd} tree and the other to the V_{ss} tree. This characteristic provides a simple scheme whereby the V_{dd} and V_{ss} trees are used to derive all bridging faults that are tested by a given test vector. Any edge with one of its end nodes in the V_{dd} tree and the other in the V_{ss} tree constitutes a transistor stuck-on fault that can be detected by the input vector. When the end nodes of an edge that represents parallel transistors span the two supply trees, then transistor stuck-on faults at one or more of the parallel transistors are sensitized. The rest of the pair-wise combinations of nodes from the two trees represent sensitized general short faults. An input short fault is a bridging fault between a node from the trees and a control label. For a control label with a logic value "1", a bridge with any node from the V_{ss} tree constitutes a sensitized input short fault. Similarly, a bridge between a control label with value "0" and a node from the V_{dd} tree is a sensitized input short fault.

4.3.2 Generating An Exhaustive Fault List

The exhaustive list of potential bridging faults for a fully complementary CMOS module can be determined from its switch-level graph, G. If we let

n = number of nodes in G, i.e., the size of N

ℓ = number of module inputs, i.e., the size of L

p = total number of possible bridging faults in G, then

$$p = \frac{(n-1)n}{2} + n\ell - 1$$

The first term accounts for all possible node-to-node faults, including general shorts and transistor stuck-on faults; the second term accounts for all possible input short faults. Since the general short fault between node V_{dd} and node V_{ss}, (1,0), is sensitized regardless of the input vector, it is not included in the exhaustive list. The subtraction of 1 at the end of the equation reflects this exclusion. Going back to the switch-level graph of Module 1 in Figure 6, the following is observed.

$$n = 19$$

$$\ell = 17$$

$$p = \frac{(18x19)}{2} + (19x17) - 1 = 493$$

For fault sensitization, the number of faults tested in a module by a particular test vector is generalized from the response graph as follows. Let

n_1 = number of nodes in V_{dd} tree

n_0 = number of nodes in V_{ss} tree

ℓ_0 = number of gate inputs with logic value "0"

ℓ_1 = number of gate inputs with logic value "1"

t_g = number of tested general shorts and transistor stuck-on faults

t_i = number of tested input short faults

t = total number of tested bridging faults, then

$t_g = n_1 n_0 - 1$

$t_i = n_1\ell_0 + n_0\ell_1$

$t = t_g + t_i = n_1 n_0 - 1 + n_1\ell_0 + n_0\ell_1$

Again, referring to Figure 6 with (10011001010011111) as the test input, the number of tested faults is accounted as follows.

$n = 19$

$n_1 = 10$

$n_0 = 6$

$\ell_0 = 7$

$\ell_1 = 10$

$t_g = 10x6 - 1 = 59$

$t_i = 11x7 + 6x10 = 137$

$t = 59 + 137 = 196$

Notice that $n_0 + n_1$ is not equal to n since there are three floating nodes in the response graph.

As mentioned in Section 4.1.2, for a fan-out free CMOS module with ℓ inputs, the number of nodes in N is $n = \ell + 2$. In the case of a fan-out CMOS module with ℓ inputs,

$$n = 2 + \sum_{i=1}^{\ell} m_i.$$

Since

$$p = \frac{(n-1)n}{2} + n\ell - 1$$

and

$$n \geq (\ell + 2)$$

then

$$p \geq \frac{(\ell + 1)(\ell + 2)}{2} + (\ell + 2)\ell - 1$$

Therefore,

$$p \geq \ell^2$$

Clearly, the total number of possible bridging faults increases rapidly as the number of module inputs or transistors get larger. However, not all faults are detectable or distinguishable and some are more likely to occur than others. The IFA technique [5,6] enables a defect generator to provide a ranked fault list that is based on fabrication statistics and circuit layout topology. For large circuits, a ranked fault list can significantly reduce the number of faults to be tested without a severe loss of accuracy.

4.4 Test Generation Analysis

Test generation for bridging faults consists of deriving a set of test vectors that will sensitize all detectable faults. To guarantee that every possible bridging fault in a module is sensitized, all node-to-node and gate-to-node pairs must be considered. This exhaustive testing may not be feasible for modules that have a large number of transistors. Fortunately, there are techniques for minimizing the number of faults that need to be explicitly tested. Apart from using a ranked list, fault collapsing can also be used to minimize a fault list and to optimize a test set. This approach is applied to test generation for bridging faults in this section. Based on the test generation discussion, a minimal or near-minimal test set for a switch-level graph is analyzed.

4.4.1 Fault Collapsing

In a switch-level graph G, let T_α be the set of all test vectors that sensitize fault α, and T_β be the set of all test vectors that sensitize fault β. The two faults, α and β are said to be equivalent if $T_\alpha = T_\beta$, i.e., the two faults are indistinguishable by any test vector. A test vector from either T_α or T_β is then able to sensitize any fault in the equivalent set. As a result, it is sufficient to consider only one fault from each set of equivalent faults in test generation. This concept is referred to as equivalence fault collapsing [20]. The representation of a group of transistors in parallel by a single edge in a switch-level graph is an application of equivalence fault collapsing. The number of essential faults to be considered can be further reduced by a process called dominance fault collapsing [20]. Fault α dominates fault β, if and only if any test vector that sensitizes β will at the same time sensitize α, i.e., $T_\beta \subset T_\alpha$. Thus, α is not an essential fault to consider in test generation. Unlike equivalent faults, α and β in this case are distinguishable. By utilizing these collapsing techniques, the complexity of test generation for bridging faults can be greatly reduced. For fault diagnosis, however, dominance fault collapsing can not be applied and a larger test set may be necessary to satisfy both detectability and uniqueness of faults. These two fault collapsing techniques are useful in the generation of a collapsed fault list and a minimum number of test vectors that completely test all faults.

4.4.2 General Series-Parallel Graphs

Most switch-level graphs for CMOS modules are composed of distinct series-parallel arrangements of transistors. They belong to the class of general series-parallel (GSP) graphs [41]. The series-parallel design can be used to advantage in switch-level test generation. A *cell* is defined as a set of storage nodes and edges that can be identified according to the topological arrangement in a switch-level graph. Each cell has two terminating nodes with some or no nodes between the two

terminating nodes. The two fundamental types of cells in GSP graphs are the *parallel cell* and the *series cell.* Each parallel cell represents two or more transistors of the same type, n-channel or p-channel, arranged in parallel. A series cell represents one or more transistors of the same type arranged in series. The switch-level representation of parallel and series transistors was discussed in Section 4.2.1. A group of parallel transistors is represented by a single edge with multiple control labels and every series transistor is represented by an edge with a single control label. A complex cell consists of a parallel and series transistors. Both simple and complex cells may be nested within another complex cell. A GSP graph is then composed of a sequence of simple and/or complex cells.

A CMOS GSP graph has at least one cell in the P-network and one cell in the N-network. For a fully complementary CMOS module, every cell in the P-network has a dual cell in the N-network, and vice versa. That is, if the N-network has a parallel cell, then the P-network will have its dual cell, a series cell. All simple CMOS modules have only two cells which are dual simple cells. For instance, the switch-level graph of a 2-input NAND gate in Figure 9 has a parallel cell represented by a single edge in the P-network and a series cell represented by two consecutive edges in the N-network. Figure 10 illustrates the decomposition of a complex module into three complex cells. Notice that between the terminating nodes (7,0), the single complex cell in the N-network has four complex cells, { (7,8,9,12,13), (7,10,11,16), (14,15,0), (16,17,18,0) }, and one simple cell (13,14) embedded in it. The application of cell decomposition in test generation for GSP graphs is discussed in the following section.

4.4.3 Minimal Test Sets

This discussion is restricted to the class of irredundant CMOS modules and assumes that at most a single fault is present. Within this class, there are fan-out free and fan-out modules [20]. In a fan-out free module, each input controls only one pair of complementary transistors. A

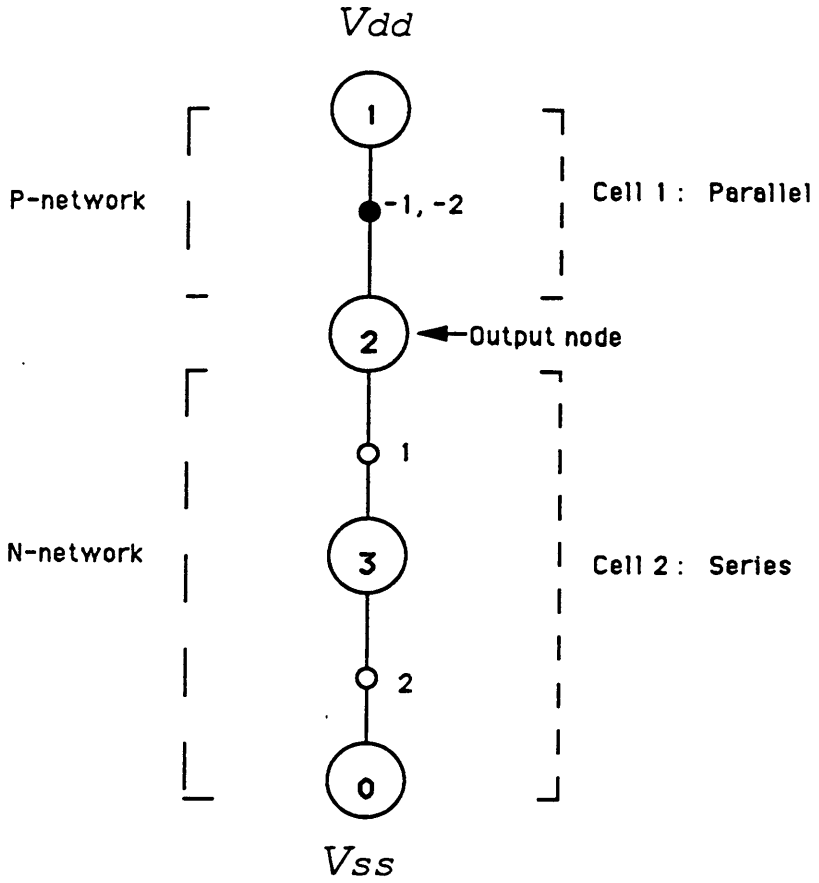

Figure 9. Decomposition of a simple switch-level graph into cells.

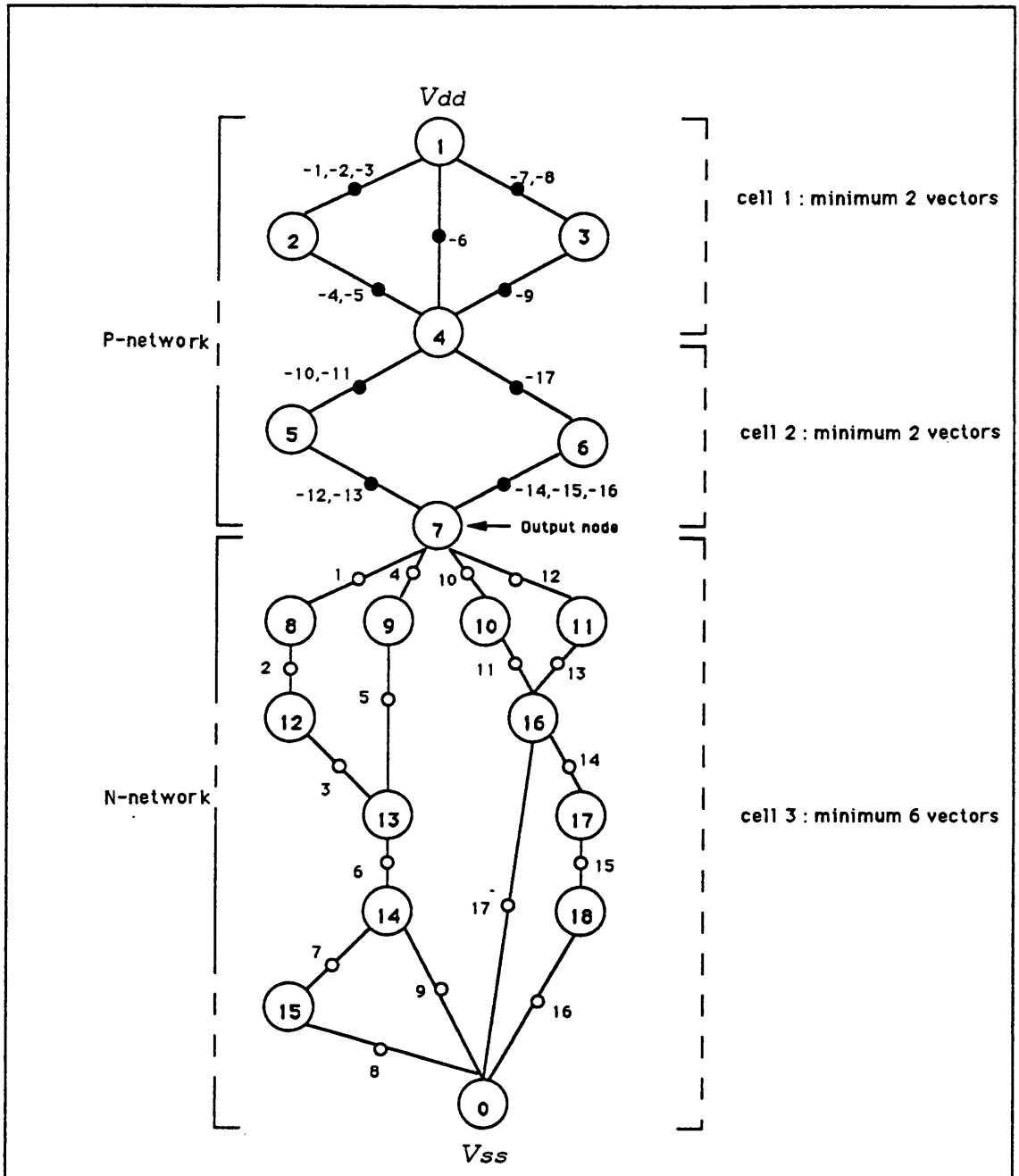

Figure 10. Decomposition of a complex switch-level graph into cells.

CMOS module has a fan-out when one of its inputs drives multiple pairs of complementary transistors in the same module. All fan-out free modules can be represented as GSP graphs, but not all GSP graphs are fan-out free. In general, CMOS modules fall into one of three categories:

1. fan-out free and GSP,
2. fan-out and GSP, and
3. fan-out and non-GSP.

The case of a fan-out free and non-GSP CMOS module is not possible since a non-GSP graph involves at least one edge that has a fan-out input.

A minimal test set is defined as the test set that covers all bridging faults in an exhaustive list and has the smallest size among test sets that have 100 percent coverage for the exhaustive list. Only fan-out free GSP CMOS modules are guaranteed to have minimal test sets that can be derived from switch-level graphs using the technique described below. Because of reconvergence, the derivation of a minimal test set for a fan-out GSP module is not as straightforward, nor is it always possible. Every fan-out input could imply that an extra test vector must be added to the minimal test set derived assuming no fan-outs for the switch-level graph. The graph method may give a near-minimal test set for a fan-out GSP module. The switch-level graph of a non-GSP module cannot be analyzed in terms of cells and hence the graphical approach to test generation does not apply to the fan-out non-GSP modules.

The following discussion considers the derivation of a minimal test set for a fan-out free GSP module. A parallel cell of t transistors indicates t equivalent transistor stuck-on faults that can be sensitized by a single test vector. Assuming this parallel cell is in the P-network, to sensitize any one of the transistor stuck-on faults, all t control labels must have their values equal to "1". If the set of t parallel transistors is in the N-network, then all t control labels will have to be "0" to sensitize any one of the transistor stuck-on faults.

In the case of a series cell with t transistors, to sensitize a transistor stuck-on fault, the transistor under consideration is turned off and the other t-1 transistors are turned on. For the P-network, this means the control label is "1" for the off transistor and "0" for the other transistors in series

with it. If the series cell is in the N-network, then the off transistor will have a control label of value "0" and the other transistors have "1". Therefore, a series cell with t transistors needs at least t unique test vectors to fully test all its transistor stuck-on faults.

Since a simple CMOS module has only two cells that are simple and are duals, at least $t + 1$ test vectors are needed to cover all transistor stuck-on faults; one for the parallel cell and t for the series cell of t transistors. When a test vector from this set is applied to the switch-level graph of a simple module, the single inactive edge divides the graph into the two supply trees without creating any floating nodes. This division allows all general and input short faults to be sensitized by the test set that sensitized all transistor stuck-on faults. A simple CMOS module, therefore, has a unique minimal test set of $t + 1$ test vectors. Since simple cells are fan-out free, $t = \ell$ and the unique minimal test set has $\ell + 1$ test vectors. The 2-input NAND gate, whose switch-level graph is shown in Figure 9, has a minimal test set of $(2+1)=3$ vectors and they are { (11), (10), (01) }.

In the case of a complex GSP CMOS module that is without fan-outs, the same generalization applies. Figure 10 shows the three complex cells and the minimum number of test vectors for each cell. The minimal test set size is the sum of the three minima. Each complex cell has several branches that meet at the two terminating nodes. For instance, Cell 1 in Figure 10 has three branches, 1-2-4, 1-4, and 1-3-4, where node 1 and node 4 are the cell's terminating nodes. Cell 3 has only two branches that terminate at node 7 and node 0. A desirable test vector should turn off a transistor or a set of parallel transistors from each branch, and turn on some other transistors from the same branch so that the end nodes of the off transistors are connected to V_{dd} and V_{ss}.

Equivalence fault collapsing applies to parallel transistors in the same branch, while dominance fault collapsing applies to transistors in different branches. For instance, in Cell 1, transistors with control labels { 7,8 } in the 1-3-4 branch constitute two equivalent transistor stuck-on faults. On the other hand, the transistor stuck-on fault of the 1-4 branch dominates transistor stuck-on faults of the other two branches, 1-2-4 and 1-3-4. With fault collapsing, the length of a branch is defined as the number of series edges in that branch. In general, a stuck-on fault for a transistor or a group of parallel transistors in the longer branch is dominated by a stuck-on fault for a transistor or a group of parallel transistors in the other shorter branches. Therefore, the minimum number of test

vectors needed to test all transistor stuck-on faults in a complex cell equals the length of the longest branch in that cell.

For example, the longest branches in Cell 1 and Cell 2 have two series edges, hence at least two vectors are necessary for each cell to detect all the transistor stuck-on faults. Cell 3 has embedded complex cells in both branches, and the minimum for an embedded cell is resolved before the minimum for the branch can be determined. The length of a branch with complex cells is the sum of the minima for all its embedded cells and the minimum for a complex cell containing several branches is the length of the longest branch. For example, in Figure 10, the length of the branch 7-8-12-9-13-14-15-0 is $3+1+2 = 6$; similarly, the length of the branch 7-10-11-16-17-18-0 is $2+3 = 5$. Hence, the minimum test set size for Cell 3 is 6. Finally, the minimum test set size of the module equals the sum of all minima of its cells, i.e., $2+2+6 = 10$.

Unlike the minimal test set for a simple cell, a test set that completely covers all transistor stuck-on faults and input short faults in a complex cell may not also cover all general short faults. For a cell with multiple branches, every branch must have one and only one inactive edge for each test vector. The selection of the inactive edge in each branch will determine if the test set so formed completely tests all possible bridging faults in addition to the transistor stuck-on faults. Figure 11, for example, shows that by selecting test vectors in a cyclic pattern to avoid adjacent branches having their inactive edges in the same relative positions, all node-to-node bridging faults between any two branches are tested by the four test vectors chosen for the complex cell in the N-network. The minimal test set, therefore, for this switch-level graph has eight test vectors. However, if there are b branches in a cell and the length of the longest branch is m, where m is smaller than b, then only m branches can have all their general short faults sensitized by a test set derived using the above approach. A bigger test set is necessary to completely cover all the general shorts. The effectiveness of the test set is still high, since nodes further apart are less likely to be shorted.

The following procedure can be used to find the size of a minimal test set for a fan-out free GSP CMOS module.

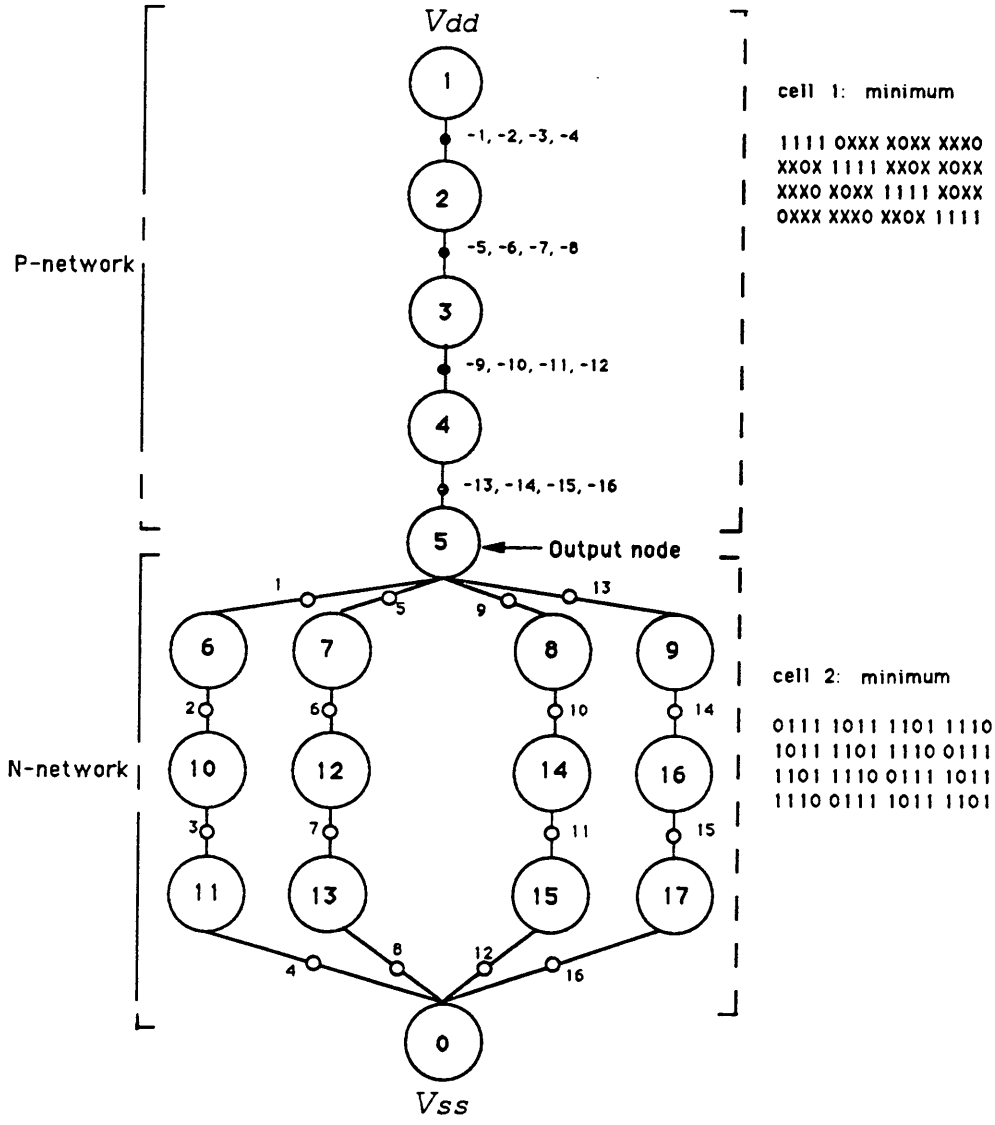

Figure 11. A complete minimal test set: cyclic arrangements

1. Divide the fan-out free GSP graph into cells.
2. Find the minimum test set size for each cell.
3. Sum up all minimum test set sizes from the cells.

A minimal test set is also dependent on the particular switch-level graph. Different implementations of the same function may require different minimal test sets of the same size. The minimal test set for the module shown in Figure 12 is the same size as that for the module shown in Figure 10, but the test vectors are different. Moreover, the minimal test set for a complex switch-level graph may not be unique.

4.5 Summary

For this research, a two-level CMOS circuit representation is used. The higher-level abstraction partitions the circuit into modules, while the switch-level abstraction preserves physical structures of individual functional modules. The discussion is restricted to combinational CMOS circuits with fully complementary modules. Single fault occurrence and supply current testing are assumed. Since the fault simulation of a CMOS circuit is the fault simulation of individual modules put together, this chapter concentrates on the fault simulation of a module. Both logic simulation and fault sensitization of a module are presented. Once the V_{dd} and V_{ss} trees are derived at the end of logic simulation, the set of tested bridging faults can be determined. The exhaustive fault list is derived from the switch-level graph and its size is $O(P)$. Fault collapsing techniques are used to reduce the number of faults to be tested in a module.

Based on the observations of fault sensitization requirements, test patterns for CMOS modules are analyzed. If certain conditions are met, a fan-out free CMOS module has a minimal test set that completely covers the faults in the exhaustive list with a minimum number of test vectors. All

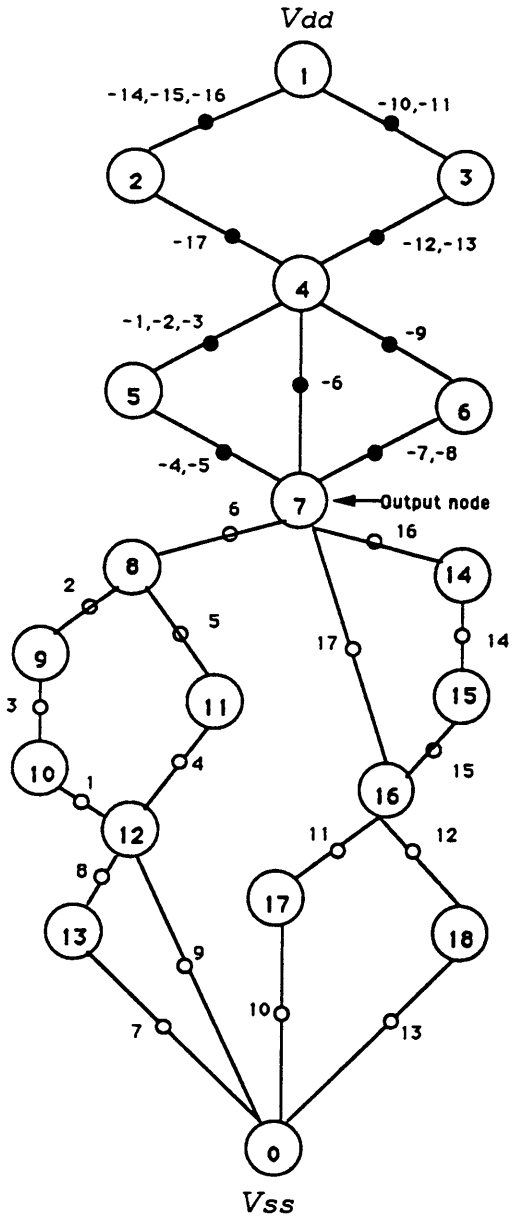

Figure 12. A different switch-level graph: $f = \{ (l_1 \cdot l_2 \cdot l_3 + l_4 \cdot l_5) \cdot l_6 \cdot (l_7 \cdot l_8 + l_9) + l_{10} \cdot l_{11} + l_{12} \cdot l_{13}) \cdot (l_{17} + l_{14} \cdot l_{15} \cdot l_{16}) \}'$.

bridging faults in modules with fan-outs may not be fully covered by any test set, even the exhaustive test set. The next chapter discusses the implementation of the fault simulation program.

Chapter 5. Fault Simulation Program

This chapter focuses on the design and implementation of the fault simulation program. Preprocessing and parallelism within machine operations are used to minimize repetition, reduce computation complexity, and increase processing flexibility. The program is implemented in C and runs under Digital Equipment Corporation's ULTRIX operating system.

5.1 Important Features of the Program

In this implementation, the network partitioning and the switch-level descriptions of module types are specified by the user and are preprocessed to yield the data structures used by the main simulation program. For large circuits, partitioning tools such as PRIMO [42], and layout extractors like ACE [43] can be used during preprocessing. Up to four preprocessing programs may be required to prepare the data structures for the main simulation program. In general, the overhead of preprocessing is small compared to the flexibility and efficiency gained by the main program. The preprocessing programs and their impact on the main program are briefly described in this section.

The preprocessing program *order.c* converts the partitioning information into an array of structures and writes the results to *order.rec* which is an input file for the main program. Each element of the array represents a module instance at a high level, and the modules they represent will be processed by the main program in the order of occurrence in the array. Prior to the switch-level simulation of a module, the main program reads from order.rec the module type, fault list type, and control labels of the current module instance. In general, the main program depends on this input file for the processing schedule and control of inputs to individual module instances. Section 5.2 describes the input requirements for this preprocessing program in more detail.

Since most VLSI CMOS circuit layouts repeat isomorphic functional units, time can be saved by merely analyzing the set of unique module types. Module instances of the same module type share fundamental structures such as sets N and E of their switch-level graphs, but have different sets of labels, L. To reduce the number of iterations for each logic simulation at the switch level, the nodes in a switch-level graph are numbered and the netlist for the switch-level graph is ordered according to the pair of nodes representing each edge. All ordered netlists are supplied to the program *mod.c* to be processed into data forms that are suitable for the main program. Essential characteristics of module types are then saved in *mod.rec* which is another input file for the main program. The file contains an array of structures, each of which represents the switch-level netlist of a module type and some additional characteristics extracted from the user-specified netlist. There is no significance to the order of elements in the array, as long as the module type number in order.rec is consistent with the module type number in this file. The main program references the file mod.rec as many times as there are module instances in the circuit, independent of the number of module types described in the file. Further discussion on the inputs for mod.c program is provided in Section 5.3.

The third preprocessing program *in.c* accepts input test vectors in the form of binary strings, formats them into data structures that support parallelism in the main program and writes the results to the file *in.rec*. When a set of test vectors is simulated sequentially, each simulation procedure for a test vector is repeated with only very minor changes, making the process inefficient. By processing a group of test vectors simultaneously, however, computational commonalities can be

exploited to increase the efficiency and reduce the cost of fault simulation. Since each bit in a test vector is a binary digit representing an input, some parallelism can be achieved by packing the same input bits of several test vectors into words and performing operations a word at a time instead of a bit at a time. If a computer has a 32-bit word, for example, a logical operation can perform thirty-two bit operations in parallel, i.e., a particular input of thirty-two test vectors are operated on in parallel. The main program refers to in.rec whenever it is ready to process a group of test vectors in parallel using bit-level logic operations. Detail of the formatting of test vectors into machine words is explained in Section 5.4.

An optional preprocessing program *list.c* creates for the main program an optional input file *list.rec* that contains fault lists specified by the user instead of the exhaustive lists generated by the main program. The input data for the program list.c consists of fault lists with each list representing a set of node-to-node faults and a set of input short faults for a particular module type. The program organizes the fault lists into an array of structures with each structure containing the data for node-to-node and input short faults of a particular fault list. As is true with the other preprocessing programs, this program could have been incorporated into the main program as a subroutine, but it is set aside as a separate program to increase the flexibility of the simulation program as a whole. For instance, if the data in list.rec is valid, and one of the other three input files needs modification, then only the program for that file is executed again prior to the main program. Section 5.5 describes list.c in more detail. This feature is not implemented in the current simulation program, but is outlined here for future accomodation.

The main fault simulation program, often referred to simply as the main program, runs on its own without user intervention once its three input files order.rec, mod.rec, in.rec, and the optional file list.rec, if needed, are correctly created by the preprocessing programs. The main program is divided into subroutines that implement logic implication, logic simulation, fault sensitization for each module instance, and computation of the overall fault coverage. Bit-level logic operations on machine words are used where they are applicable. The results of fault simulation are written to an output file named *fault.rec* which is useful for fault diagnosis and test grading. Presently, only intra-module faults are considered, although the program can be extended with some adjustments

to accommodate inter-module faults. Section 5.5 describes the structure of the main program and its subroutines in greater detail.

5.2 *Schedule Preprocessing*

This section describes the input requirements of the program order.c. Like the other three preprocessing programs, its input data may be entered line by line interactively as prompted by the program or an external file may be redirected to the program. An example of the input to the program order.c for the 3-module circuit in Figure 4 is shown below.

```
1 0 1 2
2 1 -1 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
3 -1 1 -2
```

Each line specifies a single module instance and is processed in the order of specification, with the first line having the highest priority. The first integer in each line identifies the module type, the second integer identifies the fault list, and the rest of the integers represent control labels for the module instance. The module type number should correspond to the number used in the file mod.rec. It should be noted that the module type number does not determine the order of specification. For example, if the first module and the third module in Figure 4 have the same module type whose identification number is 1, then the input for order.c becomes the following.

```
1 0 1 2
2 1 -1 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
1 -1 1 -2
```

The fault list number indicates whether or not fault simulation is needed and which fault list to use for the module instance. Three categories of fault lists are included to allow the user more flexibility in controlling the fault simulation for individual module instances. A positive integer means the module instance requests fault simulation using a user-specified fault list from the optional list.rec, a "-1" means the exhaustive fault list generated by the main program is chosen, and a zero means fault simulation is not needed. The last category is included since faults in simple module instances may be easily sensitized by test vectors for other complex module instances in the circuit, so fault simulation for the simple modules may be bypassed without significant impact on the overall fault coverage. Another reason to bypass fault simulation for some module instances, simple or complex, is to be able to focus on the fault simulation of some specific module instances. For the three modules in Figure 4, for example, Module 1 bypasses fault simulation, Module 2 uses the first fault list in list.rec and Module 3 uses the exhaustive fault list generated by the main program.

Beginning at the third integer of an input line, the integers represent the control labels for that module instance. All positive integers correspond to primary inputs of the circuit, and all negative integers correspond to outputs of some previous module instances. The output for the module instance in the *ith* line, is represented by "-i", and any module instance driven by it will have the negative integer as one of their control labels. Since only combinational CMOS circuits are considered, no feedback loops are allowed, which means that all negative integers representing control labels in a line must have their absolute values smaller than the current line number. In the third input line for the circuit in Figure 4, which represents the third module instance, the first control label of the module instance is the first primary input of the circuit, and the second control label is the output of the second module instance. This indicates that the first primary input is connected to the first input of the third module instance and the second module's output is connected to the second input of the third module instance.

For every input line received by the program, a data structure that includes other information such as the number of control labels in each line is created. At the end of preprocessing, an array of structures that represent the circuit in terms of module instances is written to the file order.rec.

The user-specified input data are, therefore, transformed by the preprocessing program into data structures that can be readily used by the main program.

5.3 *Module Type Preprocessing*

Program mod.c processes switch-level descriptions of unique module types into structures that are suitable for the main program. For each module type, the user specifies the following netlist of edges to the program mod.c:

- a pair of nodes for every edge in its switch-level graph,
- the label symbols for each edge, and
- the output node.

Before the user can supply the netlist to the program, nodes in each switch-level graph must be numbered in a certain order and edges must be prioritized. The numbering system is discussed first, followed by the netlist specification.

5.3.1 Node Numbering

For every module type, the nodes of its switch-level graph are numbered topologically with the V_{dd} node being "1" and V_{ss} being "0". Proceeding from the V_{dd} node down toward the V_{ss} node, the rest of the nodes are identified by numbers beginning at "2". Except for the V_{ss} node, nodes in the P-network are numbered prior to nodes in the N-network. Within the P-network, the source node of every edge must have a smaller number than its drain node. Node number "2" is assigned to the drain node of an edge that has V_{dd} as its source node. The output node is the last node in

the P-network and it has the highest number in the P-network. Continuing from the output node to V_{ss} in the N-network, the source nodes are assigned higher numbers than their respective drain nodes. Refer to the switch-level graph in Figure 5 and Figure 6 for examples of the numbering system.

When a node has more than one edge branching out from it to nodes of higher numbers, then it is called a *branch-out node*. Similarly, a node whose number is higher than those connected to it through multiple edges is called a *branch-in node*. The V_{ss} node is considered a branch-in node when more than one edge is connected to it even though the other nodes connected to it have higher numbers. A node can be both a branch-in and a branch-out node. The pair of terminating nodes that defines a *cell* in Section 4.4.2 is a pair of branch-out and branch-in node. The terminating branch-out node of a cell has the smallest number, while the terminating branch-in node has the highest number relative to the rest of the nodes in that cell, except for node V_{ss} . The module shown in Figure 5 has no branch-out or branch-in nodes. However, the module in Figure 6 has nodes { 1, 4, 7, 14, 16 } as branch-out nodes and { 4, 7, 13, 16, 0 } as branch-in nodes. Notice that { 4, 7, 16 } are both branch-in and branch-out nodes. This node numbering system minimizes the number of iterations needed for logic simulation.

5.3.2 Netlist Specification

Each netlist that specifies the switch-level graph of a module to the preprocessing program mod.c begins with an edge whose first node is "1", followed by edges whose first nodes are the same or higher numbers. Edges with the same first node appear in the netlist according to their second nodes. For every edge in the netlist, the first two numbers represent the two end nodes and the other numbers are the label symbols. These symbols are used for local switch-level graph representation only. The absolute value of each label symbol is mapped to the control labels in order.rec, with a negative symbol indicating that the inverse of the control label's value is used. Following the last edge, the output node is specified in a new line, and the line after that should

contain a character indicating whether or not another netlist is available. An "n" means no more netlists, and a "y" means another netlist is to follow. For example, the input to mod.c for the circuit in Figure 4 is shown in Figure 13. The first three lines describe the netlist for the first module type which is a 2-input NAND gate with "2" representing its output node. The lines between the first and the second "y" describe the second module type, and the rest of them are for the third module type.

During logic simulation, node status evaluations from the first to the last edges in the netlist are called *VDD forward-tracing* and *VSS back-tracing* for V_{dd} and V_{ss} trees, respectively. Conversely, evaluations from the last to the first edges in the netlist are called *VDD back-tracing* and *VSS forward-tracing* for V_{dd} and V_{ss} trees, respectively. For a given test vector, the switch-level netlist may be traversed forward and backward a few times before the status of every node is finalized. An important function of the mod.c program is to extract from the netlist the back-tracing requirements for both supply trees. Back-tracing flags are set or cleared based on the switch-level characteristics of the module type. The netlist and its back-tracing flags for a module type are written to mod.rec as an element of an array of structures. This implementation saves the main program from having to repeat the netlist processing each time the same module type is referenced.

The process of extracting back-tracing characteristics from a given netlist can be broken down into four parts giving four back-tracing flags. The four parts are VDD back-tracing in the P-network, VDD back-tracing in the N-network, VSS back-tracing in the P-network, and VSS back-tracing in the N-network. This division optimizes the efficiency of logic simulation in the main program.

Any node that appears as the first node more than once in the netlist is considered a branch-out node. In the P-network, when a branch-out node's number is greater than or equal to "1" but less than the output node's number, the two back-tracing flags for this network are set. Likewise, a branch-out node with a number greater than the output node's number will cause the two flags for the N-network to be set. These four flags will be used by the main program to determine if back-tracing for V_{dd} and V_{ss} trees are needed. The concept of forward-tracing and back-tracing for the

```
1  2 -1 -2
2 3 1
3 0 2

2

y

1 2  -1 -2 -3
1 3 -7 -8
1 4 -6
2 4 -4 -5
3 4 -9
4 5 -10 -11
4 6 -17
5 7 -12 -13
6 7 -14 -15 -16
7 8 1
7 9 4
7 10 10
7 11 12
8 12 2
9 13 5
10 16 11
11 16 13
12 13  3
13 14 6
14 15 7
14 0 9
15 0 8
16 17 14
16 0 17
17 18 15
18 0 16

7

y

1 2 -1
2 3 -2
3 0 1 2

3

n
```

Figure 13. An example of input data for mod.c.

V_{dd} and V_{ts} trees is discussed in more detail in Section 5.5. A pseudo-code description for mod.c is given in Figure 14.

5.4 *Input Preprocessing*

The requirements and results of the preprocessing program in.c are described in this section. This is an important preprocessing step that prepares the test vectors for bit-level parallel simulation in the main program. Input data for in.c is a list of binary strings, each with a length that equals the number of primary inputs to the circuit. The following notation is used in the discussion.

ℓ = number of primary inputs

τ = number of test vectors specified

ω = machine-dependent word length

ρ = [τ / ω], where [i] is the smallest integer $\geq$ i

χ_{ji} = a compact signal word, $j = 1,2,...,\rho$ and i = 1,2,...,ℓ.

When multiple vectors are exercised simultaneously, it is referred to as a *pass*. With test parallelism, ρ then represents the number of passes required for fault simulation and every module instance is evaluated only ρ times, rather than τ times. At each pass, through bit-level logic operations on signal words, up to ω test vectors may be active in deriving ω response graphs. Details of the processing are explained in Section 5.6.

For demonstration purposes, the length of a machine word is assumed to be 4, i.e. ω = 4. Figure 15 and Figure 16 show the input data before and after being processed into signal words. There are τ = 6 vectors in the test set; each vector is a string of binary digits ℓ_{ts}, where $t = 1,2,...,\tau$ and $s = 1,2,...,\ell$. For example, ℓ_{25} is the fifth input of the second test vector. A compact signal word

```
BEGIN mod.c
{
initialize module-type count to zero;
WHILE ( more module types to process )
   {
   clear the two P-network back-tracing flags;
   clear the two N-network back-tracing flags;
   initialize edge count and node count to zero;
   WHILE ( more edges )
      {
      increment the frequency count of the first node by 1;
      put node pair and its label symbols to the array of structures;
      increment edge count by 1;
      }
   FOR ( 1 ≤ i < output node )
      {
      IF ( frequency of node i as a first node > 1 ) THEN set P-network flags;
      }
   FOR ( output node ≤ i < node count )
      {
      IF ( frequency of node i as a first node > 1 ) THEN set N-network flags;
      }
    increment module-type count by 1;
    }
  put the array of data structures in mod.rec;
  }
  END mod.c
```

Figure 14. Pseudo-code for mod.c.

χ_{ij} encodes the ith input variable of as many as ω test vectors in the jth pass. Every bit position of a signal word corresponds to one test vector, and the positions are filled from left to right as shown in Figure 16. Eventually $\rho x \ell$ signal words are produced. A valid mask word is also created to indicate the exact number of test vectors covered by a pass. For the example shown in Figure 16, two passes are formed from the given test set, one with 4 test vectors and a full valid mask (1111), and another with only 2 test vectors and a valid mask (1100). Both signal and mask words are written to in.rec at the end of the input processing program. An example of the input data to in.c for the circuit in Figure 4 is shown in Figure 17. Figure 18 is a pseudo-code description of in.c.

5.5 Fault List Preprocessing

There are reasons to use non-exhaustive fault lists. Shorter lists reduce computation time, and ranked lists focuses on significant faults. In some ways this preprocessing step is similar to the module-type preprocessing program mod.c. Input data to the preprocessing program list.c consists of netlists of faults with each list divided into node-to-node and input short faults. An example of a non-exhaustive fault list for Module 2 of Figure 4 is given in Figure 19.

The netlist of node-to-node faults is established first. Each fault is identified by two node numbers. At the end of the netlist, a blank line indicates the beginning of a netlist for the input short faults. For each input short, the first integer is a node number and the second integer represents a control label. The character on the last line indicates whether or not another fault list is to be processed, an "n" means no more fault list to specify and a "y" means there is another fault

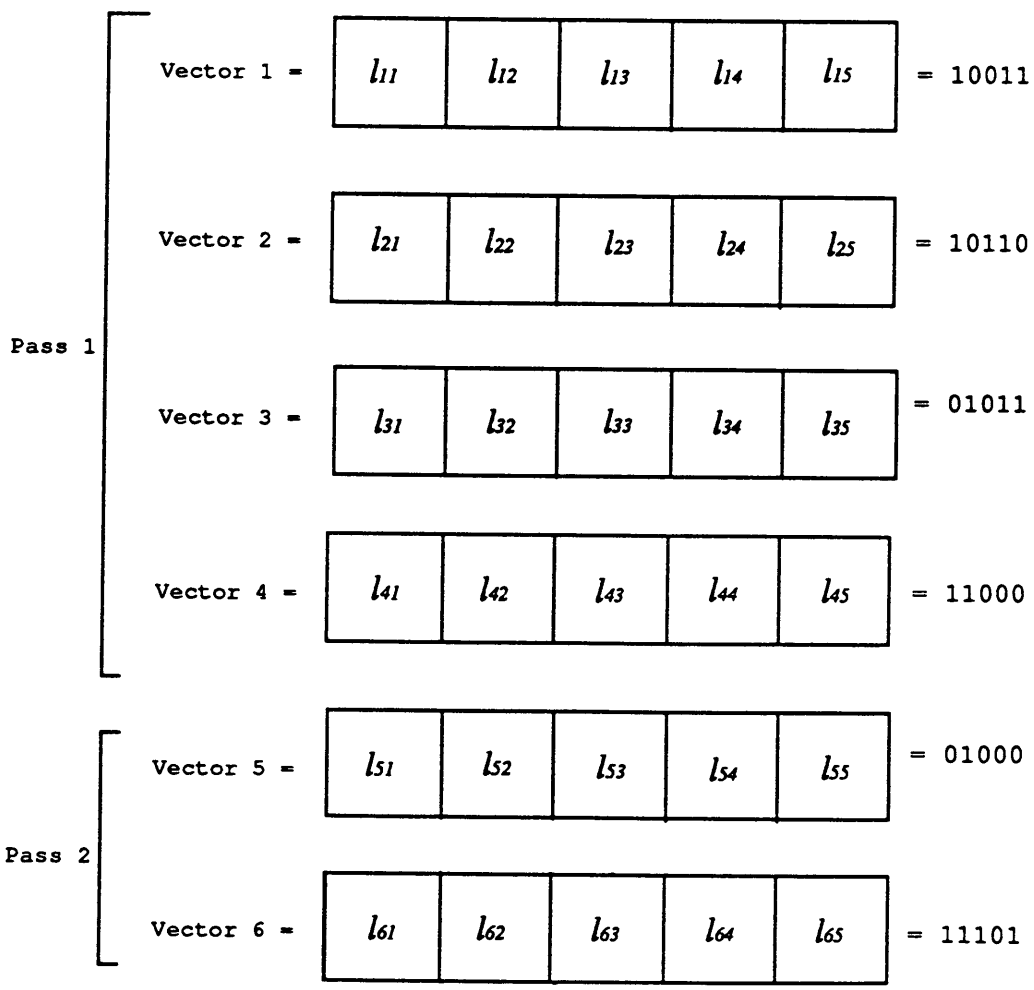

Figure 15. Test set to be formatted.

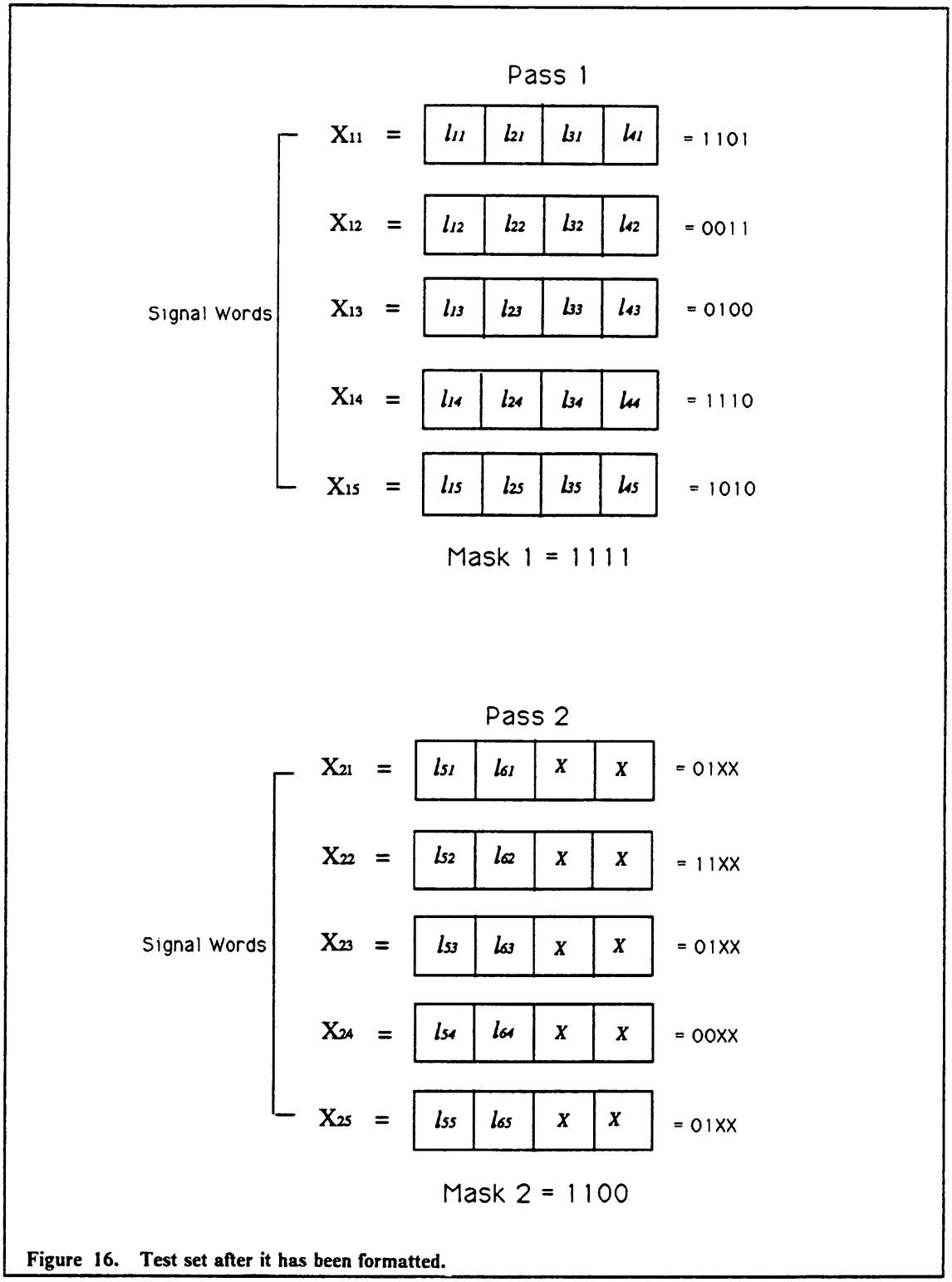

Figure 16. Test set after it has been formatted.

```
10111100110101111 1
00001010010000001 1
01100010100100111 0
00111110101011011 0
10101010101010110 0
11100010101100000 1
11011100000111000 0
10001001101001001 1
00010111011100001 0
10010101000111011 1
```

Figure 17. An example of input data for in.c.

```
BEGIN in.c
{
initialize pass count to zero, j = 0;
WHILE ( condition to process another pass is true )
   {
   initialize test vector count for present pass, v = 0;
   WHILE ( current pass is not filled, v < ω )
      {
      IF ( no more test vector ) THEN break;
      ELSE place the vector in the temporary array of strings VEC[];
      increment the vector count, v = v + 1;
      }
      ℓ = string length of VEC[0] in first pass;
      IF ( current pass is not empty, v > 0 )
         {
         FOR ( 0 ≤ i < ℓ )
             {
             initialize X_ij = 0;
             FOR ( 0 < k < v )
                 {
                 map i^th bit of VEC[k] to X_ij;
                 }
             }
         set valid mask for current pass;
         increment pass count, j = j + 1;
         }
      IF ( previous pass is not full, v < ω ) THEN break;
   }
put the array of data structures X in in.rec;
}
END in.c
```

Figure 18. Pseudo-code for in.c.

```
1 4
2 3
4 7
5 7
7 9
13 14
14 16
15 0
17 0
18 0

2 3
2 5
4 17
7 10
18 15

n
```

Figure 19. An example of input data for list.c.

list. Each fault list is an element of an array of structures in the order of specification, and the array is written to the file list.rec.

5.6 Main Program

The main program may begin when the files, order.rec, gate.rec, in.rec, and list.rec (optional) are ready. The first file to be read by the main program is in.rec that supplies signal words to initiate logic implication during each pass. Next, the file order.rec is read to obtain general information pertinent to the current module instance. The main program then proceeds to the subroutine *signal evaluation* that gets the switch-level netlist of current module from mod.rec and maps each module label symbol to the corresponding signal word. Following this, the status of all nodes in the module is evaluated by the subroutine *nodal excitation.*

When every node of the current module is at its steady state, subroutine *fault sensitization* uses bit-level logic operators on signal words and node status words to create fault signatures for faults in the specified fault list. A *fault signature* is a word that indicates the status of all test vectors in a particular pass in sensitizing a particular fault. Like a signal word, each bit in a fault signature represents a test vector. For every pass, a fault matrix consisting of fault signatures for all the faults specified is established.

Finally, the subroutine *coverage* determines the state of every fault in a fault list using the fault signatures and calculates the fault coverage of the test set for the entire circuit. All the signatures of a particular fault are OR-ed together at the bit-level to determine if the fault is tested by a test set. Details about the subroutines are given in the following sections. For simplicity, all examples cited assume that the word length ω is 4. Pseudo-code for the main program is given in Figure 20.

```
BEGIN main
{
open input files in.rec and mod.rec ;
open output file fault.rec;
WHILE ( a pass is read from in.rec )
   {
   initialize module instance count;
   initialize fault matrix count;
   open input file order.rec;
   WHILE ( a module instance is read from order.rec )
      {
      call Signal_evaluation ();
      call Nodal_excitation ();
      call Fault_sensitization ();
      increment module instance count by 1;
      }
   increment pass count by 1;
   close input file order.rec;
   }
call Coverage ();
close input files in.rec and mod.rec;
close output file fault.rec;
}
END main.
```

Figure 20. Pseudo-code for the main program.

```
BEGIN Signal_evaluation()
{
FOR ( 0 ≤ i < number of control labels for the module intance )
    {
    let num = i^th control label;
    IF ( num > 0 ) THEN label[i] = num^th input signal;
    ELSE label[i] = num^th module output signal;
    }
Read from mod.rec the netlist of current module instance;
FOR ( 0 ≤ i < total number of edges )
    {
    initialize connex[i] = 0;
    FOR ( 0 ≤ j < sum of label symbols for current edge )
        {
        let num = j^th label symbol;
        IF ( num > 0 ) THEN connex[i] v= label[num-1];
        ELSE connex[i] v= ¬label[-num-1];
        connex[i] &= valid mask;
        }
    }
}
END Signal_evaluation().
```

Figure 21. Pseudo-code for signal evaluation subroutine.

5.6.1 Signal Evaluation

The discussion of subroutine signal evaluation is based on the simulation of a single pass. During signal evaluation, control labels of the current module instance in order.rec are first updated according to the set of signals consisting of primary inputs and modular outputs. Next, the local label symbols at the switch-level graph of the module instance are mapped to the set of updated global signals. Finally, the status of every edge is evaluated based on the values of its labels. Pseudo-code for this subroutine is outlined in Figure 21.

As illustrated in the pseudo-code, the value of a negative control label is the output value of the module instance whose sequence order equals the absolute value. The value of a positive control label is the value of the primary input whose sequence order equals the positive number. Once the actual values of the control labels are known, the module's switch-level description is read from the file mod.rec. The status of every edge is evaluated by logically OR-ing the signal words of all its labels. To do this, a temporary array of words, *connex[ ]*, is created to keep track of the status of every edge as they are being updated. For each edge being evaluated, its label symbols are mapped to signal words one at a time; each time a signal word is mapped, the value is OR-ed with the current status of the edge. A negative label symbol means the inverse of the signal word whose number corresponds with the absolute value is used in the bit-level OR operation. The final state of the edge is established when all its label symbols have been examined, this final status word is AND-ed with the valid mask of the pass to ensure that only the effect of test vectors included in that pass are considered.

For a module with e edges, the size of connex[ ] is e, and each element represents the status of an edge. It is assumed that the left most bit of a word represents the first vector and its right most bit represents the ωth vector in a pass. A "1" bit in the edge status word means the edge is active for a particular test vector and a "0" means inactive. For instance, if edge i has the final status word connex[i] = 0101, then edge i is not active in the response graphs of test vectors 1 and 3, but is

active in the response graphs of test vectors 2 and 4. Recall that only active edges have their nodes connected and share a common state.

For a single module, the computational complexity for a pass in this subroutine is directly proportional to e, the number of edges in E, and ℓ, the number of labels in L. However, as discussed in Chapter 4, the number of edges in a switch-level graph is also linearly proportional to the number of labels for that graph. For large modules, the computational complexity per pass is $O(\ell)$.

5.6.2 Nodal Excitation

The objective of the nodal excitation subroutine is to evaluate the node status of a module instance using bit-level logic operators on the status words of edges. Two arrays of binary strings, VDD[ ] and VSS[ ], are specified to trace the V_{dd} and V_{ss} trees of each test vector in a pass. Each array has n elements, one for each node of the switch-level graph, and each element is a word that represents the node status of all the test vectors in a pass, with the status of the first vector represented by the left most bit of the word. At the end of this subroutine, as many as ω response graphs are exercised, and their V_{dd} and V_{ss} trees are represented by the VDD[ ] and VSS[ ] arrays.

By default, four elements of the arrays are assigned specific values to reflect the nature of the V_{dd} and V_{ss} nodes, they are VDD[1] = VSS[0] = 1111, and VDD[0] = VSS[1] = 0000. It should be mentioned that the size of ω is assumed to be 4 and a "1" bit indicates the node is connected to a particular supply node in the response graph of a particular test vector, while a "0" means it is not connected. VDD[1] represents the status of the V_{dd} node with respect to itself, likewise VSS[0] represents the status of the V_{ss} node with respect to itself, and the two nodes are not connected for fault-free logic simulation; their status are therefore as shown and do not change during nodal evaluation. The status of all the other nodes are initialized at the beginning of the subroutine as VDD[n] = VSS[n] = 0000, for $n \neq 0,1$.

After initialization of the arrays VDD[ ] and VSS[ ], the tracing of the V_{dd} tree begins in the P-network. Starting from the first edge in the netlist, the second (drain) node of every edge is evaluated based on the status of the edge and the status of the first (source) node of that edge. For a particular test vector in a pass, the second node is included in the V_{dd} tree if and only if the first node is in it and the edge represented by the nodes is active. This means that for ω test vectors of a pass, VDD[n] = VDD[n] $\vee$ (VDD[m] & connex[x]), where $\vee$ is the bit-level OR operator, & is the bit-level AND operator, m is less than n, and x is the number of the edge represented by the node pair (m,n). When the forward-tracing of the V_{dd} tree has considered all edges in the P-network, the back-tracing flag for the V_{dd} tree in the P-network is checked to decide if VDD back-tracing in the P-network is necessary. If the back-tracing flag is set, then starting from the last edge to the second edge in the P-network, node status is evaluated in the reverse order. Using the same notation, the node status evaluation for the xth edge becomes, VDD[m] = VDD[m] $\vee$ (VDD[n] & connex[x]).

If the back-tracing flag is not set or when the back-tracing in the P-network is completed, the forward-tracing for the V_{dd} tree proceeds to the N-network. Unlike the edges in the P-network, an edge in the N-network has the drain node as the first node and the source node as the second node. For every edge whose second node is not the V_{ss} node, the status of the second node is updated based on the status of the first node and the status of the edge. When the forward-tracing finally reaches the last edge of the netlist, the VDD back-tracing flag for the N-network is checked to decide if back-tracing in the N-network is necessary. Back-tracing of the V_{dd} tree in the N-network starts from the last edge in the netlist and ends at the first edge in the N-network. At the end of the forward-tracing and/or the back-tracing of V_{dd} tree in both networks, the array VDD[ ] is complete with the information of V_{dd} connection to every node in the netlist for every test vector in the current pass.

The V_{ss} tree tracing is the reverse of the V_{dd} tree tracing with minor changes, hence the discussion is brief and focuses on the differences between the two. The first half of the forward tracing of V_{ss} tree begins at the last edge and ends at the first edge in the N-network. The status of the second (source) node is evaluated based on the status of the first (drain) node and the status of the edge,

much like the back-tracing of V_{dd} tree in the N-network, i.e., VSS[m] = VSS[m] ∨ (VSS[n] & connex[x]). At the end of forward-tracing in the N-network, if the VSS back-tracing flag in the N-network is set, back-tracing is initiated; otherwise, forward-tracing continues into the next half, which is the P-network. The V_{ss} tree back-tracing updates the status of the second node based on the first node's status and the edge's status. A typical evaluation may look like this:

VSS[n] = VSS [n] ∨ (VSS[m] & connex[x])..in

Forward-tracing of the V_{ss} tree in the P-network assumes the same principle as the forward-tracing in the N-network, except that any edge whose first node is the V_{dd} node is not evaluated. The VSS back-tracing flag in the P-network is checked when the first edge in the P-network is reached. If back-tracing is necessary, it begins at the edge whose first node's number is 2 and ends before the last edge in the P-network.

Forward and back-tracings traverse every edge of the switch-level netlist to ensure the completeness of the VDD[ ] and VSS[ ] arrays. Since the V_{dd} and the V_{ss} trees are mutually exclusive, a node with a "1" bit in VDD[ ] cannot at the same time have a "1" bit in VSS[ ] for the same test vector. However, a node with a bit "0" in both VSS[ ] and VDD[ ] means that it is a floating node for that test vector. For instance, if VDD[n] = 0100 and VSS[n] = 1001, then only test vector 2 connects node n to node V_{dd}, and only test vectors 1 and 4 have node n connected to node V_{ss} . Notice that the status of test vectors 1, 2 and 4 in VDD[n] and VSS[n] are complementary, but the third bit has a "0" in both cases, so node n is a floating node for test vector 3.

Bit-level logic operations are performed on compact words from the arrays connex[ ], VDD[ ], and VSS[ ] to achieve parallel logic simulation within a pass. For a module with e edges, without any back-tracing, a total of $2(e-1)$ bit-level operations are needed for each pass. In the worst case, four back-tracings are needed, each covering only a half of the netlist (P-network or N-network). The worst case then requires $4(e-1)$ bit-level operations. Recall from Chapter 4 that $e < 2(\sum_{i=1}^{\ell} m_i)$. For a fan-out free module $m_i = 1$ and $e < 2\ell$, therefore, the computational complexity is $O(\ell)$. A pseudo-code description of this subroutine is shown in Figure 22.

```
BEGIN Nodal_excitation()
{
FOR ( 2 ≤ i ≤ highest node number of current module )
    {
    intialize VDD[i] and VSS[i] to 0;
    }
FOR ( 0 ≤ i < sum of edges in current module )
    {
    IF ( 2nd-node is not V_SS ) THEN
       {
       VDD[2nd-node] v= ( VDD[1st-node] & connex[i] );
       IF ( 1st-node is the largest number next to the output
            AND VDD back-tracing flag in P-network is set )
       THEN VDD back-tracing in P-network;
       }
    ELSE IF ( 1st-node is the largest number AND VDD back-tracing flag in N-network is set )
    THEN VDD back-tracing in N-network;
    }
FOR ( sum of edges in current module > i ≥ 0 )
    {
    IF ( 1st-node is not V_dd ) THEN
      {
      VDD[1st-node] v= ( VDD[2nd-node] & connex[i] );
      IF ( 2nd-node is the smallest number next to the output
           AND VSS back-tracing flag in N-network is set )
      THEN VSS back-tracing in N-network;
      }
    ELSE IF ( 2nd-node is 2 AND VSS back-tracing flag in P-network is set )
    THEN VSS back-tracing in P-network;
    }
}
END Nodal_excitation().
```

Figure 22. Pseudo-code for nodal excitation subroutine.

5.6.3 Fault Sensitization

Results from the previous two subroutines, signal evaluation and nodal excitation, are applied to the fault sensitization subroutine. Depending on the fault-list type specified, three alternative procedures are possible within this subroutine. The first alternative, when the fault-list type is specified by "-1", leads to the generation and use of an exhaustive fault list for the current module. The second alternative requires the opening and reading of the input file list.rec. The specification for this fault-list type is a positive integer that corresponds to the order of specification of the desired fault list in the file list.rec. The last alternative is specified by the integer "0", meaning no fault sensitization is needed for the current module. Regardless of the nature of the fault list, exhaustive or not, the fault sensitization process is the same and the following general description applies once the fault is identified.

The fault sensitization process recognizes two types of faults, general node-to-node faults and input short faults. The fault signature for the bridging fault between two nodes, μ and ν, is denoted $F_n[\mu, \upsilon]$, and the fault signature for the input short fault between node μ and label ℓ is $F_i[\mu,\ell]$. Each fault signature indicates which of the ω test vectors in a pass test the fault.

Assuming these bit-level operators, $\neg$ for logical not, & for logical AND, $\vee$ for logical OR, and $\wedge$ for exclusive OR, the fault signature for a node-to-node bridging fault between μ and ν for an arbitrary pass is derived as follows.

$$F_n[\mu, \upsilon] = (VDD[\mu] \ \& \ VSS[\upsilon]) \ \vee \ (VDD[\upsilon] \ \& \ VSS[\mu])$$

Assuming, for example

$$VDD[\mu] = 1010$$

$$VSS[\mu] = 0100$$

$$VDD[\upsilon] = 1100$$

$$VSS[v] = 0011$$

then

$$F_n[\mu, v] = (1010 \ \& 0011) \ \vee \ (0100 \ \& 1100) = (0010) \ \vee \ (0100) = 0110$$

So vectors 2 and 3 test the bridging fault between node μ and node v.

For an input short fault between node μ and the input ℓ, the fault signature F_i[j, ℓ] for the jth pass is derived as follows, where $\chi_{j\ell}$ is the signal word for input ℓ during the jth pass.

$$F_i[\mu, \ell] = (\chi_{j\ell} \ \wedge \ VDD[\mu]) \ \& (\neg \chi_{j\ell} \ \wedge \ VSS[\mu])$$

Assuming, for example

$$VDD[\mu] = 1010$$

$$VSS[\mu] = 0100$$

$$\chi_{j\ell} = 1100$$

then

$$F_i[\mu, \ell] = (1100 \ \wedge \ 1010) \ \& (0011 \ \wedge \ 0100) = 0110 \ \& 0111 = 0110$$

The above fault signature indicates that test vectors 2 and 3 test the input short fault between input ℓ and node μ, and test vectors 1 and 4 fail to test the fault.

For a test set that spans more than one pass, fault signatures of the same fault from all passes are OR-ed together to determine if the fault is tested by the test set. Two non-equivalent faults will have at least one differing pair of fault signatures. Fault signatures for an untested fault will all be zero, i.e., a zero value for every vector of every pass. But, as long as one of its fault signatures

is non-zero, a fault is tested. Based on the number of tested and untested faults from fault lists of all simulated module instances, the fault coverage for a test set is derived.

A module's fault signatures in a pass is collected in an r-by-ω fault matrix, r being the length of the fault list, and is written to the output file fault.rec. Each row of the matrix represents a fault signature in binary form, and each column represents a test vector. All fault matrices for a particular module instance generated at different passes are merged for test grading and fault diagnostics. By looking down a particular column of all fault matrices for a module, all faults sensitized by the particular test vector have "1" entries. If, however, the test vector fails to test any fault in that module, then the corresponding column would have only "0" entries. Looking across a particular row of all fault matrices corresponding to a particular module, the test vectors that sensitized the particular fault have "1" entries, and if all entries of that row are "0", then the fault is not tested by the test set.

For a module instance of ℓ control labels, the computation complexity of an exhaustive fault list in this subroutine is $O(\ell^2)$. The complexity for a shorter fault list may be significantly lower, depending on the size of the fault list. Pseudo-code for this subroutine that uses the exhaustive fault list to explain the general sensitization process is shown in Figure 23 and Figure 24.

5.7 *Summary*

The program developed in this research conducts fault simulation of bridging faults assuming supply current testing, with the option of an exhaustive fault list or a user specified non-exhaustive list. At present, only the exhaustive fault list option is implemented in the main program. To enhance execution speed, computation repetitions are minimized and bit-level parallelism is implemented. One-time preprocessing converts user-specified input data into compact structures for the main program. At the end of fault simulation, the fault coverage is evaluated and fault matrices

```
BEGIN Fault_sensitization()
{
SWITCH ( fault-list type )
  {
   CASE ( fault-list type = -1 ):
   {
   FOR ( 0 ≤ μ < highest node number of current module )
       {
       FOR ( (i+1) ≤ ν ≤ highest node number )
           {
           IF ( this is the first pass ) THEN
              {
              initialize test status, T_n[μ,ν]=0;
              }
           num = (VDD[μ]&VSS[ν])∨(VSS[μ]&VDD[ν]);
           F_n[μ,ν] = num;
           T_n[μ,ν] ∨= num;
           }
       }
   FOR ( 0 ≤ μ ≤ highest node number )
       {
       FOR ( 0 ≤ ℓ < number of control labels for the module )
           {
           IF ( this is the first pass ) THEN
              {
              initialize test status, T_f[μ,ℓ]=0;
              }
```

Figure 23. Pseudo-code for fault sensitization subroutine.

```
                num = (VDD[μ]∧χ_{jℓ})&(VSS[μ]∧¬χ_{jℓ});
                F_i[μ,ℓ] = num;
                T_i[μ,ℓ] ∨= num;
                }
            }
        write the fault matrix to output file, fault.rec;
        break;
        }

        CASE ( fault-list type = 0 ):
        {
        break;
        }

        CASE ( fault-list type > 0 ):
        {
        open input file, list.rec;
        read the fault list;
        fault sensitization;
        close input file, list.rec;
        write fault matrix to output file, fault.rec;
        break;
        }
       }
    }
    END Fault sensitization().
```

Figure 24. Pseudo-code for fault sensitization subroutine (continued).

are available in an output file. The worst case computation complexity in the main program for a single pass of a module instance with ℓ inputs is $O(\ell^2)$. It is possible that with a fault list that is of complexity $O(\ell)$, the single-pass computation complexity can be reduced to $O(\ell)$.

Chapter 6. Fault Simulation Experiments

The fault simulation program described in Chapter 5 is used to simulate several single-module CMOS circuits. For each circuit, fault simulation is conducted for different sets of test vectors generated by both deterministic and random methods. Results are tabulated and presented in this chapter. All three possible module types, fan-out free GSP, fan-out GSP, and fan-out non-GSP, are represented. Finally, conclusions about the effectiveness of test sets for switch-level bridging faults are drawn.

6.1 Objectives and Methodology

Although the fault simulation program allows for circuits with multiple modules, only single-module circuits are considered here. The objectives of the experiments are to:

- verify the fault simulation algorithm and its implementation as described in Chapter 4 and Chapter 5,
- verify the fault generation analysis and test generation method presented in Chapter 4, and

- compare and contrast the effectiveness of different test generation schemes for bridging faults.

Five single-module circuits are chosen to represent the three categories of CMOS modules, and the exhaustive fault list generated by the simulation program is used in each case. The characteristics and fault simulation results for each circuit are presented separately. It should be noted that some of the results are restricted to the particular transistor-level implementation chosen for the experiments. The procedure for the simulation of each circuit is as follows.

1. A gate-level representation of the circuit is specified to Mahjong [12] to obtain a test set based on line stuck-at faults.

2. A switch-level graph for the single module of the circuit is specified.

3. A Minimal Complete Leakage Test Set (MCLTS) is derived using the graph-theoretic method of Malaiya and Su [31].

4. A Minimal or Near-Minimal test set is hand generated for the switch-level graph using the method described in Chapter 4.

5. Random test sets that are the same size as the Minimal or Near-Minimal test set and the gate-level test set are generated.

6. The files in.rec, gate.rec, and order.rec are generated by the preprocessing programs.

7. The fault simulation program is run using each of the test sets from steps 1, 3, 4, and 5.

8. The fault coverage and the fault matrices for each test set are analyzed.

There are two tables presented for each circuit studied. The first table contains pre-simulation circuit information such as the Boolean function, transistor-level implementation, size of the switch-level graph, and the deterministic test sets. Simulation results are summarized in the second

table, which displays the number of faults not tested for each fault type and the overall fault coverage for each test set. Test set sources are listed in the first column, followed by test set sizes in second column. The next three columns are for the input shorts, general shorts, and transistor stuck-on faults. The number in parenthesis under each fault type is the number of faults of that type in the exhaustive list, and the number in parenthesis under fault coverage is their total. Listed across the row corresponding to each test set is the number of untested faults for the corresponding fault type. The last column shows the fault coverage for each test set.

Three deterministic and two random test sets are included for each circuit studied. Except for the random test sets, the results for each test set are dependent on the particular transistor-level implementation chosen. The same test set may yield different results if a different implementation of the same function is used. The two random test sets are called Random I and Random II. The test set size of Random I equals the size of the smallest deterministic set and the test set size of Random II equals the size of the Mahjong test set. Whenever possible, ten random patterns are used for each random test set, and the results are averaged. The mean, μ, and the variance, σ^2, of untested faults for each fault type and the fault coverage are calculated for the random sets.

6.2 *Fan-out Free GSP Modules*

Two single fan-out free GSP modules are considered. Their switch-level graphs are shown in Figure 25 on page 83 and Figure 26 on page 89. The first module is taken from Malaiya and Su [31]. Table 1 on page 84 and Table 3 on page 87 summarize characteristics of the two modules, while Table 2 on page 85 and Table 4 on page 88 present the fault simulation results.

6.2.1 Circuit 1

In Circuit 1, with 6 inputs, there are 8 nodes and 10 edges in the switch-level graph as indicated in Table 1. Table 2 shows that the exhaustive fault list generated by the main program has 75 collapsed bridging faults: 48 are input shorts, 17 are general shorts, and 10 are transistor stuck-on faults. Input shorts constitute 64 percent of the faults in the list, and only about 13 percent are transistor stuck-on faults. At the gate-level, the Mahjong program considered a total of 20 collapsed line stuck-at faults which is about a fourth of the size of the exhaustive bridging fault list. Both the Mahjong and MCLTS sets have 7 test vectors, and the Minimal set has 6 test vectors; hence, the test set size of Random I is 6 and that of Random II is 7.

Both MCLTS and Minimal sets test all input shorts; the Mahjong set leaves one untested input short. On average, Random I has about 1 untested input short, and Random II has about half as many untested. All three deterministic sets tested all of the general shorts. Random II has an average of 2 untested general shorts, while Random I has an average of 1 untested general short. All transistor stuck-on faults are tested by the three deterministic test sets. Random I has a mean of 5.6 untested transistor stuck-on faults and Random II has a mean of 4.6. Considering the three types of bridging faults, transistor stuck-on faults have the highest untested rate, almost 70 percent, for the random test sets.

Finally, the fault coverages of the deterministic sets are all above 98 percent, with 100 percent for MCLT and the Minimal sets and 98.67 percent for the Mahjong set. The difference between the fault coverages of the two random test sets whose sizes differ by one vector is very small, and their fault coverages average to about 90 percent. Therefore, there is about 10 percent improvement in using deterministic test sets over random test sets for this circuit.

The gate-level Mahjong test set has 100 percent coverage for line stuck-at faults, but the coverage is 98.67 percent for switch-level bridging faults. The performance of the Minimal and MCLTS sets are better than the performance of the Mahjong set. The Minimal set does have one less test vector than the MCLTS set. The improvement in fault coverage of the MCLTS and Minimal sets

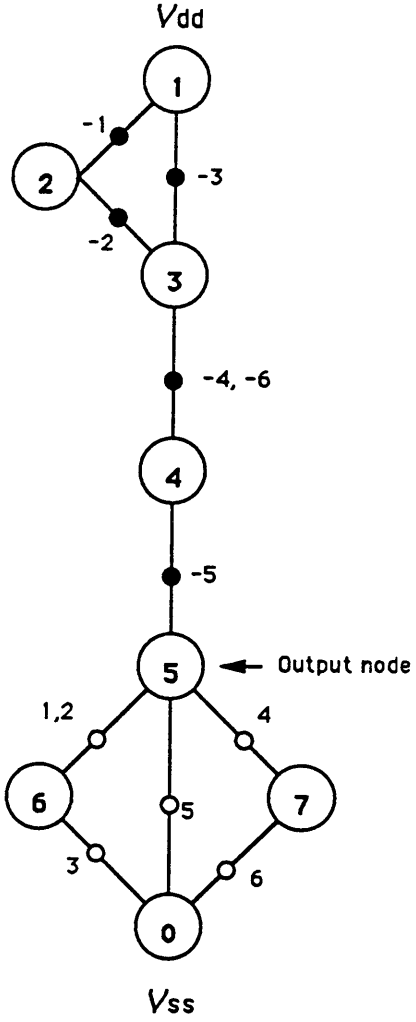

Figure 25. Circuit 1: $f = \{ (l_1 + l_2) \cdot l_3 + l_5 + l_4 \cdot l_6 \}'$.

Table 1. Test Sets for Circuit 1

Function: $f = \{ (l_1 + l_2) \cdot l_3 + l_5 + l_4 \cdot l_6 \}'$		
Description: Fan-out free, GSP, complex module with 12 transistors and 6 inputs		
Switch-level: $N = 8$; $E = 10$; $L = 6$		
Mahjong	MCLTS	Minimal
101000	101000	101100
011001	011000	011001
110100	110100	110010
110010	110001	001100
001001	000101	000101
100101	000010	010001
100001	001100	

Table 2. **Fault Simulation Results for Circuit 1**

Test set	Test set size	Input shorts (48)	General shorts (17)	Transistor Stuck-on (10)	Fault Coverage % (75)
Mahjong	7	1	0	0	98.67
MCLTS	7	0	0	0	100.00
Minimal	6	0	0	0	100.00
Random I	6	$\mu = 1.1$ $\sigma^2 = 0.885$	$\mu = 1.1$ $\sigma^2 = 1.57$	$\mu = 5.6$ $\sigma^2 = 3.44$	$\mu = 89.47$ $\sigma^2 = 22.06$
Random II	7	$\mu = 0.6$ $\sigma^2 = 0.788$	$\mu = 1.7$ $\sigma^2 = 1.632$	$\mu = 4.6$ $\sigma^2 = 1.44$	$\mu = 90.67$ $\sigma^2 = 10.66$

over the Mahjong set is less than 2 percent. If a ranked list is used and the untested input short carries a low weight, then the Mahjong test set could be essentially as good as the Minimal and MCLTS sets.

6.2.2 Circuit 2

The next circuit, Circuit 2, is another single fan-out free module. It has over three times as many transistors as Circuit 1. According to Mahjong, there are 60 line stuck-at faults for the gate-level description; the exhaustive list of bridging faults generated by the fault simulation program is eight times larger. Input shorts make up 65 percent of the exhaustive list, and only 5 percent are transistor stuck-on faults. Both the Mahjong and MCLTS test sets are the same size again, and the Minimal set is about half their size.

Like the results for Circuit 1, all three deterministic sets tested all transistor stuck-on faults. The number of untested transistor stuck-on faults for random sets averaged from 23 percent in Random II to 40 percent in Random I, an improvement of more than 10 percent compared to the results of Circuit 1. Although the Mahjong test set has 100 percent coverage for line stuck-at faults, its switch-level fault coverage is 98.99 percent. The MCLTS also has a coverage that is less than 100 percent. The Minimal set, however, has 100 percent fault coverage. With a 40 percent reduction in test set size, Random I has an average fault coverage that is 5 percent lower than Random II.

6.2.3 General Comments

Based on the results of the two fan-out free GSP modules, the following conclusions are reached. The exhaustive switch-level bridging fault list is at least four times the size of the gate-level line stuck-at fault list provided by Mahjong; the difference is larger for a higher transistor count. Input shorts constitute about 65 percent of the exhaustive fault list, with 10 percent or less being

Table 3. Test Sets for Circuit 2

Function: $f = \{ [(l_1 \cdot l_2 \cdot l_3 + l_4 \cdot l_5) \cdot l_6 \cdot (l_7 \cdot l_8 + l_9)] + [(l_{10} \cdot l_{11} + l_{12} \cdot l_{13}) \cdot (l_{17} + l_{14} \cdot l_{15} \cdot l_{16})] \}'$		
Description: Fan-out free, GSP, complex module with 34 transistors and 17 inputs		
Switch-level: $N = 19$; $E = 26$; $L = 17$		
Mahjong	MCLTS	Minimal
01101101100011011	11101111000011010	01110111011111100
11100111101001110	00111100101000011	10101111111101010
10100100101000111	11101110100011010	11001110110011101
11001101100000101	10100011011001001	11100011101101111
00111100101000011	01110110001111110	01111101010011010
00010111100010000	00001000011101110	11101110011100110
11101001101000100	01101111101011111	11101110101011100
11100101000001010	11111110011111100	01111111010101011
11101111000011010	10110111110101111	10110001111101110
11101110001001101	01101111111111010	11000011101111101
10100011011001001	10110111111111100	
00010010110001101	11001111101011111	
11101010000111011	11010111111110110	
11101011001101011	11111011110011111	
00011010111100110	11111101010101111	
00001000011101110	11111101011111010	
10100011011101010	11111011111111100	
11001001111111100	11111110001011111	

Table 4. Fault Simulation Results for Circuit 2

Test set	Test set size	Input shorts (323)	General shorts (144)	Transistor Stuck-on (26)	Fault Coverage % (493)
Mahjong	18	0	5	0	98.99
MCLTS	18	1	0	0	99.80
Minimal	10	0	0	0	100.00
Random I	10	$\mu = 1.1$ $\sigma^2 = 0.885$	$\mu = 14.9$ $\sigma^2 = 81.69$	$\mu = 11.2$ $\sigma^2 = 6.56$	$\mu = 89.47$ $\sigma^2 = 22.06$
Random II	18	$\mu = 0.7$ $\sigma^2 = 1.618$	$\mu = 1.1$ $\sigma^2 = 1.57$	$\mu = 5.6$ $\sigma^2 = 3.44$	$\mu = 94.56$ $\sigma^2 = 4.649$

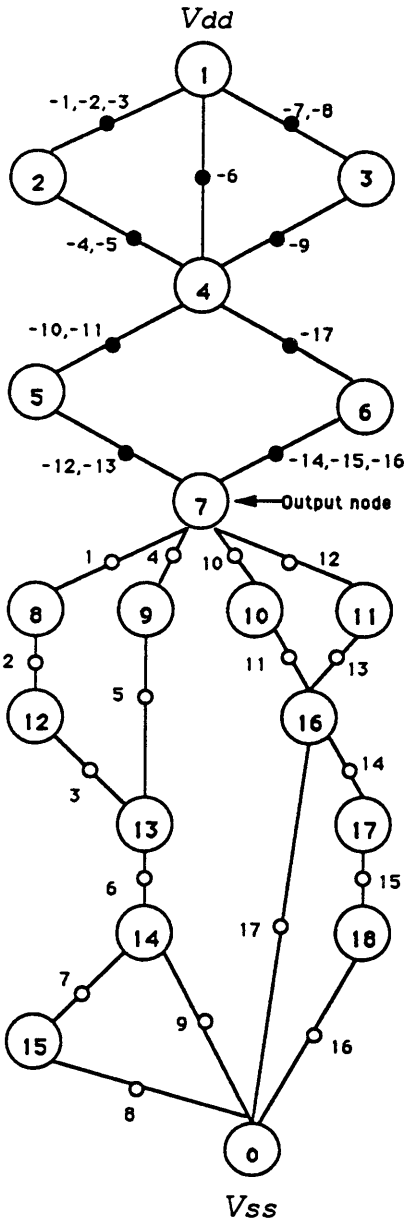

Figure 26. Circuit 2: $f = \{ [(l_1 \cdot l_2 \cdot l_3 + l_4 \cdot l_5) \cdot l_6 \cdot (l_7 \cdot l_8 + l_9)] + [(l_{10} \cdot l_{11} + l_{12} \cdot l_{13}) \cdot (l_{17} + l_{14} \cdot l_{15} \cdot l_{16})] \}'$.

transistor stuck-on faults. The number of bridging faults in the exhaustive list is approximately twice the square of the number of inputs, i.e., $2\ell^2$.

Although the Mahjong test set has 100 percent line stuck-at fault coverage, its coverage for bridging faults is less. Both the Mahjong and MCLTS sets have the same size and cover all transistor stuck-on faults even though their fault coverages may not be 100 percent. The Minimal sets are smaller by as much as 40 percent than either the Mahjong or MCLTS sets, but have 100 percent fault coverages. However, the difference in fault coverages for the three deterministics test sets is very small. Indeed, if a ranked fault list is used, and the untested faults have low weights, then the fault coverages of the Mahjong and MCLTS sets may not be significantly less than the Minimal set.

The size difference between the Minimal set and the Mahjong or MCLTS sets increases with the number of transistors in the module, from about 15 percent in Circuit 1 to almost 50 percent in Circuit 2. In general, the larger the random test set, the better the fault coverage. With the same test set size as the Mahjong or MCLTS test sets, Random II has an average fault coverage that is above 90 percent. Random I, whose size is that of the Minimal test set, has an average that is below 90 percent. Unlike the three deterministic test sets, random test sets failed to cover all transistor stuck-on faults. In fact, about 70 percent of the untested bridging faults for random test sets are transistor stuck-on faults.

6.3 Fan-Out GSP Modules

Two circuits, each having a single fan-out GSP module, are investigated. They are Circuit 3, the complement of the carry function, and Circuit 4, the complement of the sum function of a combinational 1-bit full adder. The modules are considered individually for the fault simulation experiments, and their results are combined for the consideration of a full-adder cell. Their switch-level graphs are shown in Figure 27 on page 92 and Figure 28 on page 96, respectively.

Circuit 3 has three inputs, representing the two bits to be added and one carry-in bit; the same three inputs are also the inputs for Circuit 4 with the fourth input being the output of Circuit 3. In an adder comprised of these two modules, the module of Circuit 4 is driven by the module of Circuit 3. Table 5 on page 93 and Table 7 on page 98 show the input vectors for Circuit 3 and Circuit 4, respectively. Their fault simulation results are summarized in Table 6 on page 94 and Table 8 on page 99. Another table, Table 9, combines results for the two circuits to indicate the intra-module bridging faults for a 1-bit full adder.

6.3.1 Circuit 3

For Circuit 3, there are 18 collapsed line stuck-at faults considered by Mahjong, about half the size of the exhaustive bridging fault list, and they are all tested by the Mahjong test set. According to Theorem 1 [31], a complex module with t transistors will have a MCLTS of $(\frac{t}{2}) + 1$ test vectors, therefore the MCLTS for Circuit 3 has 6 test vectors. However, using the graph-theoretic method for the test generation of the MCLTS [31], only 5 test vectors, 101, 110, 011, 001, and 010, can be derived. No test vector can satisfy the conditions for the cut-set { (5-0), (6-0) } such that the two edges are not conducting and their nodes are connected to the V_{dd} and the V_{ss} nodes. A cut-set is defined as a set of edges, removal of which will separate the graph into two parts [31]. The MCLTS is therefore not included in the fault simulation experiment for Circuit 3 and is indicated by " N/A " in Table 6.

According to the discussion of the Minimal set in Chapter 4, a Minimal set for the switch-level graph of Circuit 3 without fan-outs has 5 test vectors. Since there are fan-outs in Circuit 3, no test set of that size yields 100 percent fault coverage, and the hand generated test set becomes the Near-Minimal test set for Circuit 3. An additional test vector is added to the Near-Minimal set to achieve 100 percent fault coverage and the Minimal set, therefore, has six test vectors. In Table 5, the test set called *Extra* has one extra test vector added to the Minimal set.

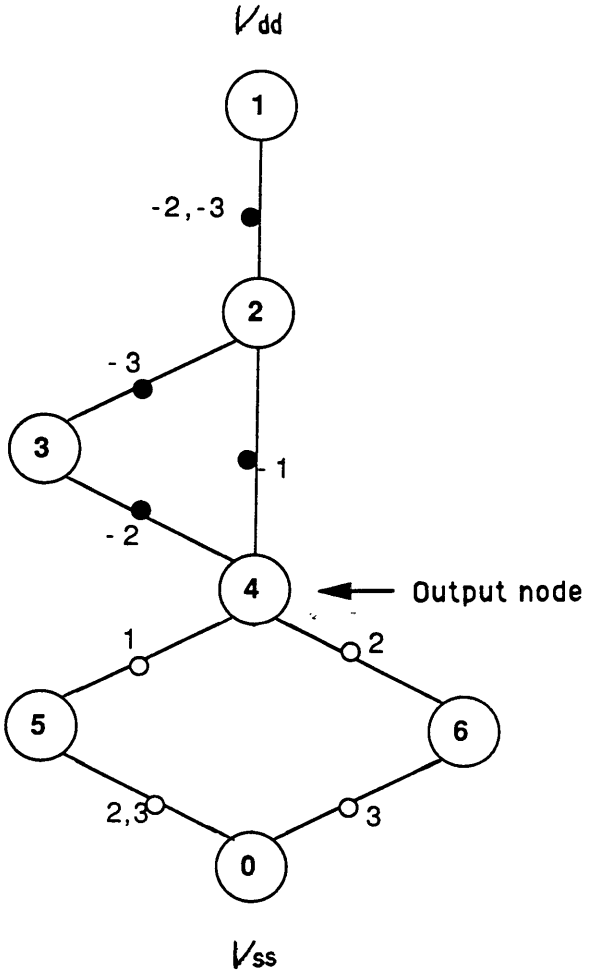

Figure 27. Circuit 3: $f = \{ (l_1 + l_2) \cdot l_3 + (l_1 \cdot l_2) \}'$.

Table 5. Test Sets for Circuit 3

Function: $f = \{ (l_1 + l_2) \cdot l_3 + (l_1 \cdot l_2) \}'$		
Description: Fan-out, GSP, complex module with 10 transistors and 3 inputs		
Switch-level: $N = 7$; $E = 8$; $L = 3$		
Mahjong	Minimal	Extra
100	100	100
011	011	011
010	010	010
101	101	101
001	001	001
110	110	110
		000

Table 6. Fault Simulation Results for Circuit 3

Test set	Test set size	Input shorts (21)	General shorts (12)	Transistor Stuck-on (8)	Fault Coverage % (41)
Mahjong	6	0	0	0	100.00
MCLTS	6	N/A	N/A	N/A	N/A
Minimal	6	0	0	0	100.00
Extra	7	0	0	0	100.00
Random I	6	$\mu = 0$ $\sigma^2 = 0$	$\mu = 0.375$ $\sigma^2 = 0.234$	$\mu = 1.625$ $\sigma^2 = 0.234$	$\mu = 95.12$ $\sigma^2 = 4.465$
Random II	7	$\mu = 0$ $\sigma^2 = 0$	$\mu = 0.25$ $\sigma^2 = 0.187$	$\mu = 0.625$ $\sigma^2 = 0.234$	$\mu = 97.86$ $\sigma^2 = 1.544$

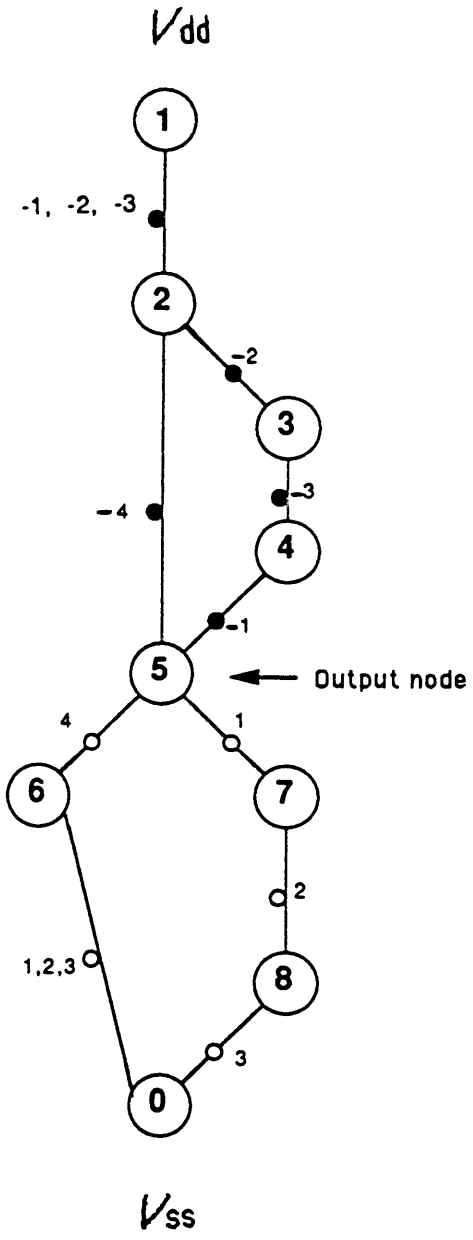

Figure 28. Circuit 4: $f = \{ (l_1 + l_2 + l_3) \cdot l_4 + (l_1 \cdot l_2 \cdot l_3) \}'$.

6.3.2 Circuit 4

The gate-level description of Circuit 4 has three inputs since it is the complement of the sum function of a 1-bit full adder. The Mahjong set for this gate-level representation has an exhaustive test set of 8 test vectors and 100 percent coverage for the line stuck-at faults. However, to be consistent with the other test sets for this circuit, a fourth input variable, which is the output of Circuit 3 with the same 3-input vector, is added to each test vector of the Mahjong set. The MCLTS is also not included in the fault simulation experiment of this circuit. According to Theorem 1, there are $(\frac{14}{2}) + 1 = 8$ test vectors for Circuit 4, but using the graph-theoretic method [31] only 7 test vectors can be derived. No vector can satisfy the conditions for the cut set { (6-0), (8-0) } shown in Figure 28. Using the method discussed in Chapter 4, the size of the Minimal set is 7 for the switch-level graph of Circuit 4 assuming no fan-out labels. However, a test set of this size fails to have 100 percent fault coverage for Circuit 4 that has fan-outs. The Near-Minimal set, therefore, has seven test vectors and the Minimal set has eight test vectors, which is also the exhaustive set.

The combined results of the intra-module bridging faults of the two circuits, Circuit 3 and Circuit 4, are shown in Table 9. The Extra test sets are included for the fault simulations of Circuit 3 and Circuit 4 to allow the calculation of intra-module bridging faults for the 1-bit full adder. Results for Set I in Table 9 are the combined results of the Minimal set in Table 6 and the Extra set in Table 8. Similarly, results for Set II in Table 9 are derived from the results of the Extra set in Table 6 and the Near-Minimal set in Table 8. At the gate-level, there are 33 collapsed line stuck-at faults for the 1-bit full adder. There are, however, a total of 112 intra-module bridging faults for the two circuits. The exhaustive Mahjong test set that has 100 percent line stuck-at fault coverage at the gate-level also has 100 percent bridging fault coverage. The MCLTS method is not applicable for these two fan-out modules.

Table 7. Test Sets for Circuit 4

Function: $f = \{ (l_1 + l_2 + l_3) \cdot l_4 + (l_1 \cdot l_2 \cdot l_3) \}'$		
Description: Fan-out, GSP, complex module with 14 transistors and 4 inputs		
Switch-level: $N = 9$; $E = 10$; $L = 4$		
Mahjong	Extra	Near-Minimal
1001	1001	1001
0110	0110	0110
0101	0101	0101
1010	1010	1010
0011	0011	0011
1100	1100	1100
0001		0001
1110		

Table 8. Fault Simulation Results for Circuit 4

Test set	Test set size	Input shorts (36)	General shorts (25)	Transistor Stuck-on (10)	Fault Coverage % (71)
Mahjong	8	0	0	0	100.00
MCLTS	8	N/A	N/A	N/A	N/A
Extra	6	0	0	2	97.18
Near-Minimal	7	0	0	1	98.59
Random I	6	$\mu = 0.25$ $\sigma^2 = 0.187$	$\mu = 1.25$ $\sigma^2 = 0.437$	$\mu = 2$ $\sigma^2 = 0$	$\mu = 95.06$ $\sigma^2 = 0.994$
Random II	7	$\mu = 0$ $\sigma^2 = 0$	$\mu = 0.5$ $\sigma^2 = 0.25$	$\mu = 1$ $\sigma^2 = 0$	$\mu = 97.88$ $\sigma^2 = 0.497$

Table 9. Intra-Module Faults for a Combinational 1-bit Full Adder

Test set	Test set size	Input shorts (57)	General shorts (37)	Transistor Stuck-on (18)	Fault Coverage % (112)
Mahjong	8	0	0	0	100.00
MCLTS	N/A	N/A	N/A	N/A	N/A
Set I	6	0	0	2	98.21
Set II	7	0	0	1	99.11
Random I	6	$\mu = 0.25$ $\sigma^2 = 0.187$	$\mu = 1.625$ $\sigma^2 = 1.234$	$\mu = 3.625$ $\sigma^2 = 0.234$	$\mu = 95.09$ $\sigma^2 = 4.185$
Random II	7	$\mu = 0$ $\sigma^2 = 0$	$\mu = 0.75$ $\sigma^2 = 0.687$	$\mu = 1.625$ $\sigma^2 = 0.234$	$\mu = 97.89$ $\sigma^2 = 1.118$

6.4 Fan-Out Non-GSP Module

Circuit 5 is an alternative CMOS circuit design for the combinational 1-bit full adder, i.e., it combines both the carry and sum modules into a single complex module. Unlike Circuit 4, which is the complement of the sum function, the output of Circuit 5 is the sum itself. Figure 29 shows the switch-level graph of Circuit 5. A brief circuit description and the list of test sets are contained in Table 10 and the fault simulation results are displayed in Table 11.

The number of line stuck-at faults considered by Mahjong is less than a third of the exhaustive bridging faults. The Mahjong test set used for this circuit is the same set used for Circuit 4, but without the fourth input variable. Since there are 18 transistors in the circuit layout, the MCLTS according to Theorem 1 [31] has a size of $(\frac{18}{2}) + 1 = 10$. However, since the circuit has only 3 inputs, the maximum size is 8; therefore, the MCLTS is not applicable for Circuit 5. The minimal test set method of Chapter 4 is not applicable to this circuit, since the switch-level graph is non-GSP. Nonetheless, the Minimal and the Extra sets for Circuit 3 are used in the simulation of Circuit 5 as Set I and Set II, respectively, for comparison purposes. The same random test sets for Circuit 3 are also used. The fault simulation results show that even the exhaustive test set does not have 100 percent fault coverage for this fan-out non-GSP circuit.

6.5 Summary

Fault simulation considering I_{dd} current testing of bridging faults for five single-module circuits ranging from fan-out free GSP to fan-out non-GSP modules has been studied. The exhaustive bridging fault lists are found to be more than twice the gate-level line stuck-at fault list generated by Mahjong; the difference is bigger for more complex modules. Transistor stuck-on faults constitute about 10 percent of the faults and are the type of fault that is hardest to detect by random

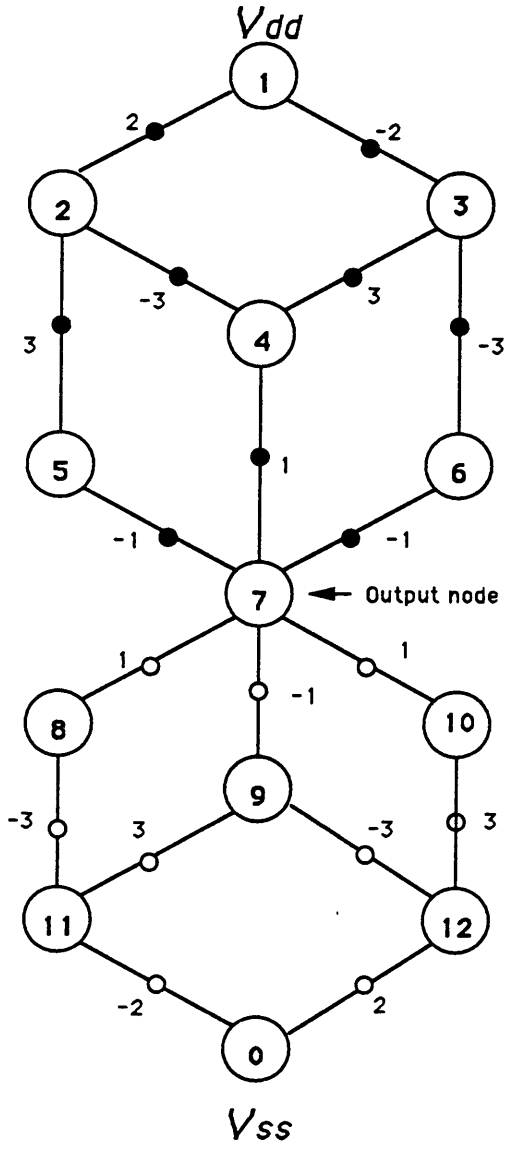

Figure 29. Circuit 5: $f = \{ (\hat{I}_1 + \hat{I}_2 + \hat{I}_3) \bullet [(I_1 + I_2) \bullet I_3 + I_1 \bullet I_2 + \hat{I}_1 \bullet \hat{I}_2 \bullet \hat{I}_3] \}'$.

Table 10. Test sets of Circuit 5

Function: $f = \{ (\hat{l}_1 + \hat{l}_2 + \hat{l}_3) \cdot [(l_1 + l_2) \cdot l_3 + l_1 \cdot l_2 + \hat{l}_1 \cdot \hat{l}_2 \cdot \hat{l}_3] \}'$		
Description: Fan-out, Non-GSP, complex module with 18 transistors and 3 inputs		
Switch-level: $N = 13$; $E = 18$; $L = 3$		
Mahjong	Set I	Set II
100	100	100
011	011	011
010	010	010
101	101	101
001	001	001
110	110	110
000		000
111		

Table 11. Fault Simulation Results for Circuit 5

Test set	Test set size	Input shorts (39)	General shorts (59)	Transistor Stuck-on (18)	Fault Coverage % (116)
Mahjong	8	2	8	0	91.38
MCLTS	N/A	N/A	N/A	N/A	N/A
Set I	6	2	11	2	87.07
Set II	7	2	8	1	90.52
Random I	6	$\mu = 2.5$ $\sigma^2 = 0.25$	$\mu = 10$ $\sigma^2 = 1.5$	$\mu = 2.5$ $\sigma^2 = 1$	$\mu = 87.06$ $\sigma^2 = 0.744$
Random II	7	$\mu = 2$ $\sigma^2 = 0$	$\mu = 8.5$ $\sigma^2 = 0.25$	$\mu = 1$ $\sigma^2 = 0$	$\mu = 90.09$ $\sigma^2 = 0.185$

test sets. It is observed that a Mahjong test set with 100 percent fault coverage for the gate-level line stuck-at faults will also have 100 percent coverage for transistor stuck-on bridging faults. However, it may not have 100 percent fault coverage for all bridging faults. This is true for all three module types, fan-out free GSP, fan-out GSP, and fan-out non-GSP. The MCLTS method of Malaiya and Su [31] is applicable only to fan-out free GSP modules. When applicable, the MCLTS test set size matches the size of the Mahjong test set. Moreover, like the Mahjong test set, it also covers all the transistor stuck-on faults. Its fault coverage performance is slightly better than the Mahjong test set, but it does not guarantee 100 percent fault coverage for all bridging faults.

A minimal test set can be generated from the switch-level graph of every fan-out free GSP module, simple or complex. As the number of transistors increases, the reduction in test set size is more pronounced. As much as a 40 percent reduction compared to the Mahjong test set is possible. For a fan-out GSP module, the switch-level graph may yield a near-minimal set rather than the minimal set, the former being a bigger set. The method for deriving a minimal or a near-minimal test set is not applicable to a fan-out non-GSP module. Even with the exhaustive test set, the fault coverage of a fan-out non-GSP module may not be 100 percent.

Apart from a smaller test set, the improvement in fault coverage contributed by the minimal test set is not significant enough to conclude that gate-level test generation is inferior to switch-level test sets. On the average, random test sets of the same size as the Mahjong test sets have fault coverages that are about 10 percent less than the Mahjong test sets. As expected, the larger the random test set, the better the fault coverage.

Chapter 7. Conclusion

This research studied fault simulation of bridging faults for combinational CMOS circuits assuming supply current testing. A two-level circuit model is used where a circuit is partitioned into connected modules, with an individual module's structural detail represented at the switch-level. Fault collapsing techniques are applied to the switch-level graphs to make bridging fault specification simple and fault simulation more efficient. The fault simulation program maintains global control over the inputs to individual modules and their processing order. A graph-based algorithm is used locally for the switch-level logic simulation and fault sensitization of individual modules. This algorithm calls for only a fault-free logic simulation. Procedures such as fault injection, simulation of faulty circuits and comparisons of good and faulty circuit results are eliminated. There is also the flexibility of choosing the type of fault list to use, an exhaustive list, a shorter but more specific list, or no list at all. Simulation efficiency is further enhanced by using preprocessing techniques and parallel processing within machine operations. The results of fault simulation are useful for test grading, fault diagnostics, and analysis of circuit testability. For large circuits, minimizing the test set size will also reduce simulation time.

This research also considered an optimum test set that has the smallest size among all test sets with 100 percent coverage for the faults in the exhaustive list. This set is referred to as the minimal test set. For a fully complementary module that is GSP and has no fan-outs, the minimal test set

can be determined based on the module's switch-level graph. For a module that is GSP but with fan-outs, there is no simple way to determined its minimal test set, but test sets with 100 percent coverage are still possible. In the worst case, a non-GSP module with fan-outs, even an exhaustive test set may not yield 100 percent fault coverage.

Results of fault simulation experiments were analyzed and the effectiveness of three test generation schemes were compared. In all cases studied, the number of bridging faults in an exhaustive list is found to be at least twice as many as the number of line stuck-at faults from Mahjong for the same module. Transistor stuck-on faults constitute the least, about 10 percent, of the bridging faults, but are the most difficult for random test sets to test. The difference between fault coverages for line stuck-at faults and for bridging faults for the gate-level test set from Mahjong is greater for the non-GSP module than the GSP modules, about 5 percent for the GSP modules and less than 10 percent for the non-GSP module. A Mahjong test set with a 100 percent fault coverage for line stuck-at faults will also have a 100 percent coverage for transistor stuck-on faults. At another level of comparison, the bridging fault coverages for the Mahjong test sets are lower than those of switch-level test sets by about 3 percent. The difference may not be significant, especially when the untested faults carry low weights. Random test sets have average fault coverages close to 90 percent, and at least half of their untested faults are transistor stuck-on faults.

The above conclusion is based on experimental results for five single-module circuits, the largest of which has 34 transistors. Only intra-module faults were examined. Fault simulation studies of multi-module circuits with more complex modules are necessary to verify the effectiveness of gate-level test sets or the superiority of switch-level test sets for testing intra-module and inter-module bridging faults. Another important factor to be included in the evaluation of fault coverage and, therefore, the effectiveness of test sets, is the weight of individual bridging faults. The fault coverage for an exhaustive fault list may be lower than that for a shorter weighted list and vice versa. Therefore, without a reliable weighted fault list, much time may be spent simulating insignificant faults while ignoring those that are critical. Future considerations for the fault simulation program should include simulating transmission gates and other important CMOS design styles such as Pseudo-CMOS.

This research has developed a switch-level fault simulation algorithm for bridging faults of combinational CMOS circuits and has implemented the algorithm as a fault simulation program that is accurate and fast. Although there are limitations and restrictions, the fault simulation program explores the potential of a switch-level bridging fault simulator that supports current-domain testing and performs logic simulation, fault generation, fault diagnostics, circuit testability and test grading. The analysis of switch-level test generation for bridging faults presents an interesting and potentially significant approach to test generation for CMOS circuits. In general, the work of this research provides a base for further research and development of a general switch-level fault simulator that is accurate and fast to meet the increasing demand for quality and reliability control for CMOS circuits.

Bibliography

1. G. Mott and J. Newkirk, "Eliminating Failures in the Field", *VLSI Systems Design,* Vol. 7, No. 10, pp. 88-90, October 1986.

2. R. Wallace, "ASICs Drive Demand for Sophisticated Test System", *Electronic Engineering Times,* pp. 31-32, May 30, 1988.

3. Y. M. El-Ziq, "Modeling and Testing of MOS Physical Failures", *Automatic Test Generation Workshop,* pp. 59-62, 1983.

4. P. Banerjee and J. A. Abraham, "Characterization and Testing of Physical Failures in MOS Logic Circuits", *IEEE Design and Test of Computers,* Vol. 1, No. 3, pp. 76-86, August 1984.

5. F. J. Ferguson and J. P. Shen, "Extraction and Simulation of Realistic CMOS Faults using Inductive Fault Analysis", *Proceedings International Test Conference,* pp. 475-484, 1988.

6. J. P. Shen, W. Maly and F. J. Ferguson, "Inductive Fault Analysis of MOS Integrated Circuits", *IEEE Design and Test of Computers,* Vol. 2, No. 6, pp. 13-26, December 1985.

7. M. Abramovici, B. Krishnamurthy, R. Mathews, B. Rogers, M. Schulz, S. Seth and J. Waicukauski, "What is the Path to Fast Fault Simulation?", *Proceedings International Test Conference,* pp. 183-190, 1988.

8. D. Smith, "Finding Fault: An Update on Fault Simulation", *VLSI System Design,* Vol. 8, No. 11, pp. 28-65, October 1987.

9. J. P. Hayes, "An Introduction to Switch-Level Modeling", *IEEE Design and Test of Computers,* Vol. 4, No. 4, pp. 18-25, August 1987.

10. S. L. Lusky and T. Sridhar, "Detectable CMOS Faults in Switch-Level Simulation", *Proceedings International Test Conference,* pp. 875-883, 1985.

11. R. E. Bryant, "A Switch-Level Model and Simulatior for MOS Digital Systems", *IEEE Transactions on Computers,* Vol. C-33, No. 2, pp. 160-177, February 1984.

12. R. S. Wai and T. H. K. Ma, *MAHJONG*, Department of EECS, University of California, Berkeley, CA, pp. 1-4, 1986.

13. D. Johnson, "Mixed-Mode Simulator Accurately Models Real-World Designs", *Computer Design,* Vol. 28, No. 14, pp. 65-67, August 1, 1988.

14. W. A. Rogers and J. A. Abraham, "CHIEFS: A Concurrent, Hierarchical and Extensible Fault Simulator", *Proceedings International Test Conference,* pp. 710-715, 1985.

15. J. R. Armstrong, "Chip-level Modeling with HDLs", *IEEE Design and Test of Computers,* Vol. 5, No. 1, pp. 8-18, February 1988.

16. A. Lowenstein and G. Winter, "VHDL's Impact on Test", *IEEE Design and Test of Computers,* Vol. 3, No. 2, pp. 48-53, April 1986.

17. S. Ghosh, "Behavioral-Level Fault Simulation", *IEEE Design and Test of Computers,* Vol. 5, No. 3, pp. 31-35, June 1988.

18. T. Chakraborty and S. Ghosh, "On Behavior Fault Modeling for Combinational Digital Designs", *Proceedings International Test Conference,* pp. 593-598, 1988.

19. M. A. d'Abreu, "Gate-Level Simulation", *IEEE Design and Test of Computers,* Vol. 2, No. 6, pp. 63-71, December 1985.

20. M. M. Breuer and A. D. Friedman, *Diagnosis and Reliable Design of Digital Systems,* Computer Scince Press, Woodland Hills, CA, pp. 224-242, 1976.

21. T. Kirkland and M. R. Mercer, "Automatic Test Pattern Generation", *IEEE Design and Test of Computers,* Vol. 5, No. 3, pp. 43-54, June 1988.

22. R. E. Bryant, "A Survey of Switch-Level Algorithms", *IEEE Design and Test of Computers,* Vol. 4, No. 4, pp. 26-40, August 1987.

23. R. Chandramouli and H. Sucar, "Defect Analysis and Fault Modeling in MOS Technology", *Proceedings International Test Conference,* pp. 313-321, 1985.

24. J. P. Hayes, "A Unified Switching Theory with Applications to VLSI Design", *Proceedings of the IEEE,* Vol. 70, No. 10, pp. 1140-1151, October 1982.

25. W. Y. Koe and S. F. Midkiff, "Circuit Simulation of CMOS Faults", *Proceedings IEEE Southeastcon,* Vol. I, No. 2, pp. 87-91, 1988.

26. J. P. Hayes, "Pseudo-Boolean Logic Circuits", *IEEE Transactions on Computers,* Vol. C-35, No. 7, pp. 602-605, July 1986.

27. R. E. Bryant, "Algorithmic Aspects of Symbolic Switch Network Analysis", *IEEE Transactions on Computer-Aided Design,* Vol. CAD-6, No. 4, pp. 618-633, July 1987.

28. R. L. Wadsack, "Fault Modeling and Logic Simulation of CMOS and MOS Integrated Circuits", *Bell System Technical Journal,* Vol. 57, No. 5, pp. 1449-1473, May-June 1978.

29. C. F. Hawkins and J. M. Soden, "Reliability and Electrical Propeties of Gate Oxide Shorts in CMOS ICs", *International Test Conference,* pp. 443-451, 1986.

30. J. M. Acken, "Testing for Bridging Faults (Shorts) in CMOS Circuits", *Proceedings 20th Design Automation Conference,* pp.717-712, 1983.

31. Y. K. Malaiya and Y. H. Su, "A New Fault Model and Testing Technique for CMOS Devices", *IEEE Test Conference,* pp. 25-33, 1982.

32. R. Rajsuman, Y. K. Malaiya and A. P. Jayasumana, "On Accuracy of Switch-Level Modeling of Bridging Faults in Complex Gates", *Proceedings 24th Design Automation Conference,* pp. 244-250, 1987.

33. L. K. Horning, J. M. Soden, R. R. Fritzemeier and C. F. Hawkins, "Measurements of Quiescent Power Supply Current for CMOS ICs in Production Testing", *Proceedings International Test Conference,* pp. 300-309, 1987.

34. Y. M. Malaiya and A. P. Jayasumana, and R. Rajsuman, "A Detailed Examination of Bridging Faults", *Proceedings International Conference on Computer Design,* pp. 78-82, 1986.

35. Y. K. Malaiya, "Analyzing Data for CMOS Leakage Faults", *Proceedings International Test Conference,* pp. 25-34, 1982.

36. R. E. Bryant and M. D. Schuster, "Fault Simulation of MOS Digital Circuits", *VLSI Design,* Vol. 4, No. 10, pp. 24-30, October 1983.

37. M. Kawai and J. P. Hayes, "An Experimental MOS Fault Simulation Program CSASIM", *Proceedings 21st Design Automation Conference,* pp. 2-9, 1984.

38. C. Y. Lo, H. N. Nham, and A. K. Bose, "Algorithms for an Advanced Fault Simulation System in MOTIS", *IEEE Transactions on Computer-Aided Design,* Vol. CAD-6, No. 2, pp. 232-240, March 1987.

39. Z. Barzilai et al., "Fast Pass-Transistor Simulation for Custom MOS Circuits", *IEEE Design and Test of Computers,* Vol. 1, No. 1, pp. 71-80, February 1984.

40. Y. K. Malaiya, "Testing Stuck-on Faults in CMOS Integrated Circuits", *Proceedings International Conference on Computer-Aided Design,* pp. 248-250, 1984.

41. R. E. Bryant, "Boolean Analysis of MOS Circuits", *IEEE Transactions on Computer-Aided Design,* Vol. CAD-6, No. 4, pp. 634-649, July 1987.

42. A. Gupta, "ACE: A Circuit Extractor", *Proceedings 20th Design Automation Conference,* pp. 721-725, 1983.

43. V. B. Rao and T. N. Trick, "Network Partitioning and Ordering for MOS VLSI Circuits", *IEEE Transactions on Computer-Aided Design*, Vol. CAD-6, No. 1, pp. 128-143, January 1987.

The vita has been removed from the scanned document